

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>	V Discovery	917
<u>13</u>	28 Discovery methods: an overview	919
<u>14</u>	29 Latent variable models	921
<u>15</u>	30 Hidden Markov models	961
<u>16</u>	31 State-space models	991
<u>17</u>	32 Graph learning	1019
<u>18</u>	33 Non-parametric Bayesian models	1029
<u>19</u>	34 Representation learning (Unfinished)	1061
<u>20</u>	35 Interpretability	1063
<u>21</u>		
<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

3 Statistics 63

27	3.1	Introduction	63
28	3.1.1	Frequentist statistics	63
29	3.1.2	Bayesian statistics	63
30	3.1.3	Arguments for the Bayesian approach	64
31	3.1.4	Arguments against the Bayesian approach	64
32	3.1.5	Why not just use MAP estimation?	65
33	3.2	Closed-form analysis using conjugate priors	69
34	3.2.1	The binomial model	69
35	3.2.2	The multinomial model	77
36	3.2.3	The univariate Gaussian model	79
37	3.2.4	The multivariate Gaussian model	84
38	3.2.5	Conjugate-exponential models	90
39	3.3	Beyond conjugate priors	92
40	3.3.1	Robust (heavy-tailed) priors	92
41	3.3.2	Priors for variance parameters	93
42	3.4	Noninformative priors	94
43	3.4.1	Maximum entropy priors	94
44	3.4.2	Jeffreys priors	95
45	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	8.3.4	Two-slice smoothed marginals	323
2	8.3.5	Time and space complexity	324
3	8.3.6	The Viterbi algorithm	325
4	8.3.7	Forwards filtering, backwards sampling	328
5	8.3.8	Application to discretized state spaces	328
6	8.4	Inference for linear-Gaussian SSMs	329
7	8.4.1	The Kalman filter	329
8	8.4.2	Kalman filtering for linear regression (recursive least squares)	334
9	8.4.3	Predictive coding as Kalman filtering	336
10	8.4.4	The Kalman (RTS) smoother	338
11	8.5	Inference based on local linearization	339
12	8.5.1	Taylor series expansion	339
13	8.5.2	The extended Kalman filter (EKF)	342
14	8.5.3	The extended Kalman smoother	345
15	8.5.4	Exponential-family EKF	345
16	8.6	Inference based on the unscented transform	347
17	8.6.1	The unscented transform	348
18	8.6.2	The unscented Kalman filter (UKF)	349
19	8.6.3	The unscented Kalman smoother	351
20	8.7	Other variants of the Kalman filter	352
21	8.7.1	Ensemble Kalman filter	352
22	8.7.2	Robust Kalman filters	353
23	8.7.3	Gaussian filtering	353
24	8.8	Assumed density filtering	356
25	8.8.1	The ADF algorithm	356
26	8.8.2	Connection with Gaussian filtering	357
27	8.8.3	The Gaussian sum filter for switching SSMs	357
28	8.8.4	ADF for training logistic regression	360
29	9	Message passing inference	365
30	9.1	Introduction	365
31	9.2	Belief propagation on trees	366
32	9.2.1	BP for polytrees	366
33	9.2.2	BP for undirected graphs with pairwise potentials	369
34	9.2.3	BP for factor graphs	370
35	9.2.4	Max product belief propagation	371
36	9.2.5	Gaussian and non-Gaussian belief propagation	373
37	9.3	Loopy belief propagation	373
38	9.3.1	Convergence	374
39	9.3.2	Accuracy	376
40	9.3.3	Connection with variational inference	377
41	9.3.4	Generalized belief propagation	377
42	9.3.5	Application: error correcting codes	377
43	9.3.6	Application: Affinity propagation	379
44	9.3.7	Emulating BP with graph neural nets	380

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	Preconditionining	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			

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			
36	III Prediction	547	
37			
38	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			
7	15	Generalized linear models	565
8	15.1	Introduction	565
9	15.1.1	Examples	565
10	15.1.2	GLMs with non-canonical link functions	568
11	15.1.3	Maximum likelihood estimation	568
12	15.1.4	Bayesian inference	569
13	15.2	Linear regression	570
14	15.2.1	Conjugate priors	570
15	15.2.2	Uninformative priors	572
16	15.2.3	Informative priors	574
17	15.2.4	Spike and slab prior	576
18	15.2.5	Laplace prior (Bayesian lasso)	577
19	15.2.6	Horseshoe prior	578
20	15.2.7	Automatic relevancy determination	579
21	15.3	Logistic regression	581
22	15.3.1	Binary logistic regression	582
23	15.3.2	Multinomial logistic regression	582
24	15.3.3	Priors	583
25	15.3.4	Posteriors	584
26	15.3.5	Laplace approximation	584
27	15.3.6	MCMC inference	587
28	15.3.7	Variational inference	588
29	15.4	Probit regression	588
30	15.4.1	Latent variable interpretation	588
31	15.4.2	Maximum likelihood estimation	589
32	15.4.3	Bayesian inference	590
33	15.4.4	Ordinal probit regression	591
34	15.4.5	Multinomial probit models	592
35	15.5	Multi-level GLMs	592
36	15.5.1	Generalized linear mixed models (GLMMs)	592
37	15.5.2	Model fitting	593
38	15.5.3	Example: radon regression	593
39			
40	16	Deep neural networks	597
41	16.1	Introduction	597
42	16.2	Building blocks of differentiable circuits	597
43	16.2.1	Linear layers	598
44	16.2.2	Non-linearities	598
45	16.2.3	Convolutional layers	599
46			
47			

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		
2	17.6.4	Variational methods 648
3	17.7	Hierarchical Bayesian neural networks 648
4	17.7.1	Solving multiple related classification problems 649
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	18.6	Learning the kernel 687
30	18.6.1	Empirical Bayes for the kernel parameters 687
31	18.6.2	Bayesian inference for the kernel parameters 690
32	18.6.3	Multiple kernel learning for additive kernels 691
33	18.6.4	Automatic search for compositional kernels 693
34	18.6.5	Spectral mixture kernel learning 695
35	18.6.6	Deep kernel learning 697
36	18.6.7	Functional kernel learning 698
37	18.7	GPs and DNNs 699
38	18.7.1	Kernels derived from random DNNs (NN-GP) 700
39	18.7.2	Kernels derived from trained DNNs (neural tangent kernel) 703
40	18.7.3	Deep GPs 705
41		
42	19 Structured prediction 711	
43	19.1	Introduction 711
44	19.2	Conditional random fields (CRFs) 711
45	19.2.1	1d CRFs 711
46		
47		

1	19.2.2	2d CRFs	715
2	19.2.3	Parameter estimation	717
3	19.2.4	Other approaches	718
4	19.3	Time series forecasting	719
5	19.3.1	Structural time series models	719
6	19.3.2	Prophet	725
7	19.3.3	Gaussian processes for timeseries forecasting	726
8	19.3.4	Neural forecasting methods	727
9	19.3.5	Causal impact of a time series intervention	728
10			
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			
41	22 Variational autoencoders	785	
42	22.1	Introduction	785
43	22.2	VAE basics	785
44	22.2.1	Modeling assumptions	786
45	22.2.2	Evidence lower bound	787
46	22.2.3	Optimization	788
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			

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

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

<u>1</u>	<u>33.2.1</u>	Definition	1030
<u>2</u>	<u>33.2.2</u>	Stick breaking construction of the DP	1032
<u>3</u>	<u>33.2.3</u>	The Chinese restaurant process (CRP)	1033
<u>4</u>	<u>33.2.4</u>	Dirichlet process mixture models	1035
<u>5</u>	<u>33.3</u>	Generalizations of the Dirichlet process	1040
<u>6</u>	<u>33.3.1</u>	Pitman-Yor process	1041
<u>7</u>	<u>33.3.2</u>	Dependent random probability measures	1042
<u>8</u>	<u>33.4</u>	The Indian buffet process and the Beta process	1044
<u>9</u>	<u>33.5</u>	Small-variance asymptotics	1047
<u>10</u>	<u>33.6</u>	Completely random measures	1050
<u>11</u>	<u>33.7</u>	Lévy processes	1051
<u>12</u>	<u>33.8</u>	Point processes with repulsion and reinforcement	1053
<u>13</u>	<u>33.8.1</u>	Poisson process	1053
<u>14</u>	<u>33.8.2</u>	Renewal process	1054
<u>15</u>	<u>33.8.3</u>	Hawkes process	1055
<u>16</u>	<u>33.8.4</u>	Gibbs point process	1057
<u>17</u>	<u>33.8.5</u>	Determinantal point process	1058
<u>18</u>	<u>34</u>	Representation learning (Unfinished)	1061
<u>19</u>	<u>34.1</u>	CLIP	1061
<u>20</u>	<u>35</u>	Interpretability	1063
<u>21</u>	<u>35.1</u>	Introduction	1063
<u>22</u>	<u>35.1.1</u>	The Role of Interpretability	1064
<u>23</u>	<u>35.1.2</u>	Terminology and Framework	1065
<u>24</u>	<u>35.2</u>	Methods for Interpretable Machine Learning	1069
<u>25</u>	<u>35.2.1</u>	Inherently Interpretable Models: The Model is its Explanation	1069
<u>26</u>	<u>35.2.2</u>	Semi-Inherently Interpretable Models: Example-Based Methods	1071
<u>27</u>	<u>35.2.3</u>	Post-hoc or Joint training: The Explanation gives a Partial View of the Model	1072
<u>28</u>	<u>35.2.4</u>	Transparency and Visualization	1076
<u>29</u>	<u>35.3</u>	Properties: The Abstraction Between Context and Method	1077
<u>30</u>	<u>35.3.1</u>	Properties of Explanations from Interpretable Machine Learning	1077
<u>31</u>	<u>35.3.2</u>	Properties of Explanations from Cognitive Science	1080
<u>32</u>	<u>35.4</u>	Evaluation of Interpretable Machine Learning Models	1081
<u>33</u>	<u>35.4.1</u>	Computational Evaluation: Does the Method have Desired Properties?	1082
<u>34</u>	<u>35.4.2</u>	User Study-based Evaluation: Does the Method Help a User Perform a Task?	1086
<u>35</u>	<u>35.5</u>	Discussion: How to Think about Interpretable Machine Learning	1090
<u>36</u>	<u>VI</u>	Decision making	1097
<u>37</u>	<u>36</u>	Multi-step decision problems	1099
<u>38</u>	<u>36.1</u>	Introduction	1099
<u>39</u>	<u>36.2</u>	Decision (influence) diagrams	1099
<u>40</u>			
<u>41</u>			
<u>42</u>			
<u>43</u>			
<u>44</u>			
<u>45</u>			
<u>46</u>			
<u>47</u>			

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

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

<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 • Roy Frostig (Google), who wrote [Section 6.2 \(Automatic differentiation\)](#).
- ² • Andrew Wilson (NYU), who helped write [Chapter 17 \(Bayesian neural networks\)](#) and [Chapter 18 \(Gaussian processes\)](#).
- ³ • George Papamakarios (Deepmind) and Balaji Lakshminarayanan (Google), who wrote [Chapter 24 \(Normalizing Flows\)](#).
- ⁴ • Yang Song (Stanford) and Durk Kingma (Google), who helped write [Chapter 25 \(Energy-based models\)](#).
- ⁵ • Mihaela Rosca (Deepmind / UCL), Shakir Mohamed (Deepmind) and Balaji Lakshminarayanan (Google), who wrote [Chapter 27 \(Generative adversarial networks\)](#).
- ⁶ • Vinayak Rao (Purdue), who wrote [Chapter 33 \(Non-parametric Bayesian models\)](#).
- ⁷ • Ben Poole (Google) and Simon Kornblith (Google), who wrote [Chapter 34 \(Representation learning \(Unfinished\)\)](#).
- ⁸ • Been Kim (Google) and Finale Doshi-Velez (Harvard), who wrote [Chapter 35 \(Interpretability\)](#).
- ⁹ • Lihong Li (Amazon, work done at Google), who helped write [Section 36.4 \(Contextual bandits\)](#) and [Chapter 37 \(Reinforcement learning\)](#).
- ¹⁰ • Victor Veitch (Google / U. Chicago) and Alexander D'Amour (Google), who wrote [Chapter 38 \(Causality\)](#).
- ¹¹
- ¹²
- ¹³
- ¹⁴
- ¹⁵
- ¹⁶
- ¹⁷
- ¹⁸
- ¹⁹

²⁰**Acknowledgements**

²¹I 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 Farnsworth (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
- ³³
- ³⁴
- ³⁵
- ³⁶
- ³⁷

³⁸**About the cover**

³⁹The cover illustrates a variational autoencoder ([Chapter 22](#)) being used to map from a 2d Gaussian [40](#) to image space.[41](#)

⁴²Kevin Patrick Murphy
⁴³Palo Alto, California
⁴⁴March 2022.
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
Bernoulli	1	$x \in \{0, 1\}$	
Binomial	N	$x \in \{0, 1, \dots, N\}$	
Categorical	1	$x \in \{1, \dots, K\}$	
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.

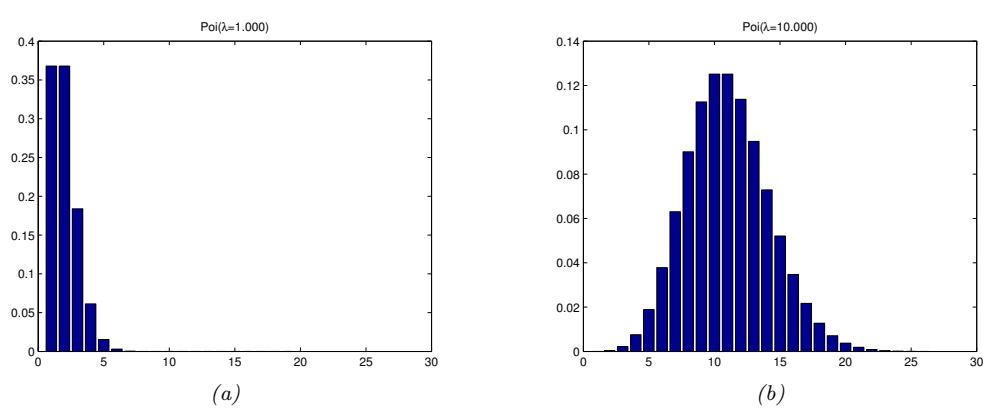


Figure 2.1: Illustration of some Poisson distributions for $\lambda = 1$ (left) and $\lambda = 10$ (right). We have truncated the x-axis to 30 for clarity, but the support of the distribution is over all the non-negative integers. Generated by [poisson_dist_plot.py](#).

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 $\Phi(x)$.

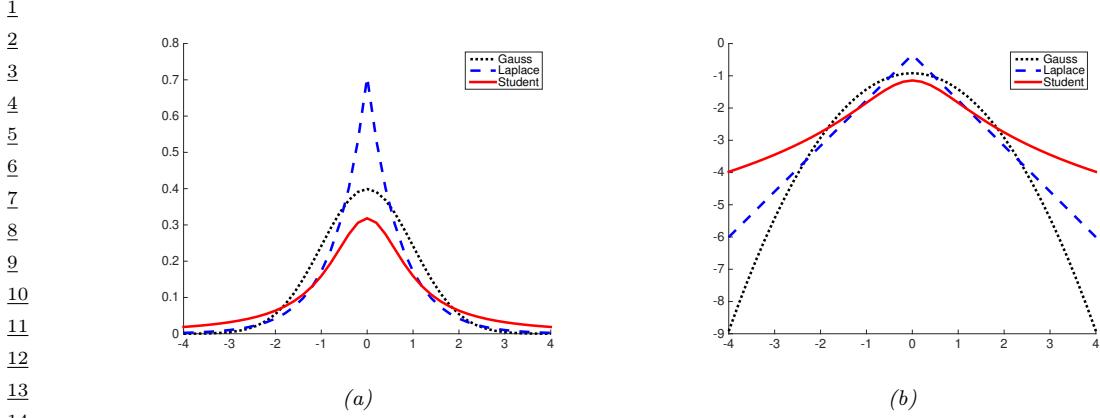


Figure 2.2: (a) The pdf's for a $\mathcal{N}(0, 1)$, $T_1(0, 1)$ and $\text{Lap}(0, 1/\sqrt{2})$. The mean is 0 and the variance is 1 for both the Gaussian and Laplace. The mean and variance of the Student distribution is undefined when $\nu = 1$. (b) Log of these pdf's. Note that the Student distribution 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:

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

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

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:

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

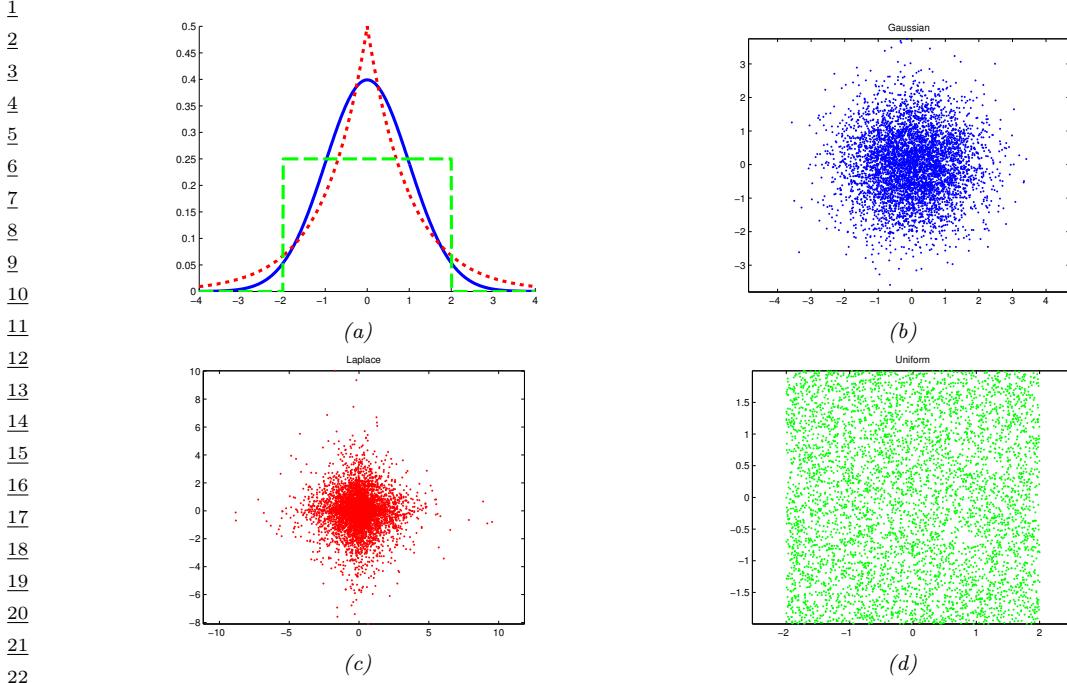


Figure 2.3: Illustration of Gaussian (blue), sub-Gaussian (uniform, green) and super-Gaussian (Laplace, red) distributions in 1d and 2d. Generated by `sub_super_gauss_plot.py`.

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

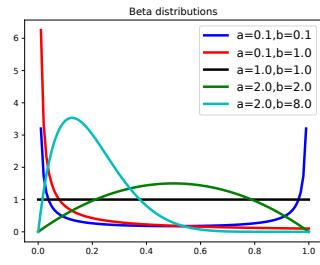
$$\text{kurt}(z) \triangleq \frac{\mu_4}{\sigma^4} = \frac{\mathbb{E}[(Z - \mu)^4]}{(\mathbb{E}[(Z - \mu)^2])^2} \quad (2.21)$$

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.

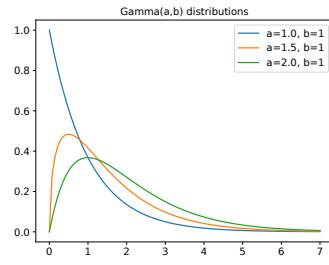
A super-Gaussian distribution (e.g., the Laplace) has positive excess kurtosis, and hence heavier tails than the Gaussian. A sub-Gaussian distribution, such as the uniform, has negative excess kurtosis, and hence lighter tails than the Gaussian. See Figure 2.3 for an illustration.

². This term is from [Kru13].

1
2
3
4
5
6
7
8
9
10



(a)



(b)

11
12
13
14
15

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`.

16

2.2.2.7 Beta distribution

19
20
21
22

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

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

23
24

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

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

27
28

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

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

31

See Figure 2.4a for plots of some beta distributions.

32
33
34
35
36

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:

37
38
39

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

40

41
42
43
44

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 $b > 0$:

45
46
47

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

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 $a > 2$.

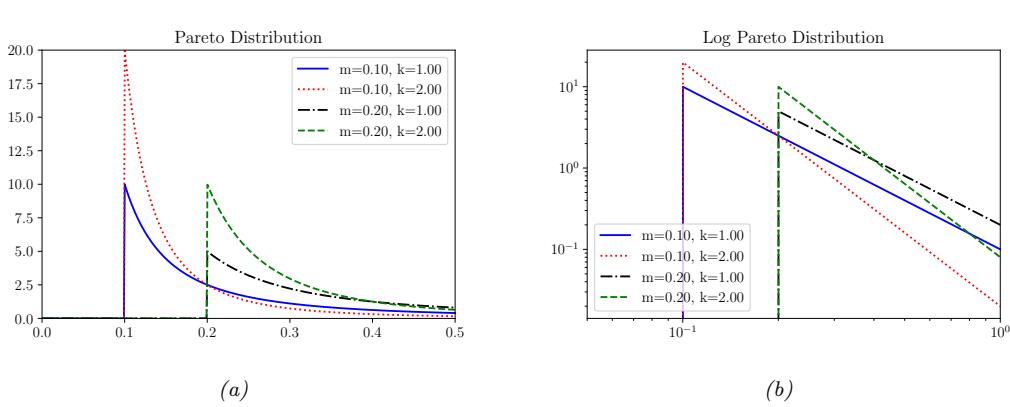


Figure 2.5: (a) The Pareto pdf $\text{Pareto}(x|k,m)$. (b) Same distribution on a log-log plot. Generated by `pareto_dist_plot.py`.

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

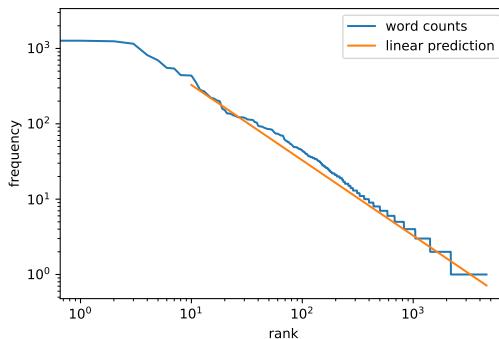


Figure 2.6: A log-log plot of the frequency vs the rank for the words in H. G. Wells’ The Time Machine. Generated by `zipfs_law_plot.py`. Adapted from a figure from [Zha+20a, Sec 8.3].

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

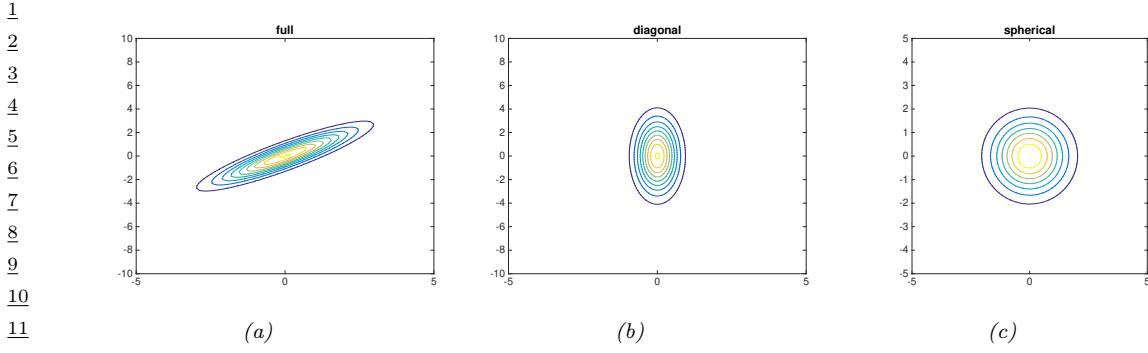


Figure 2.7: Visualization of a 2d Gaussian density in terms of level sets of constant probability density. (a) A full covariance matrix has elliptical contours. (b) A diagonal covariance matrix is an **axis aligned** ellipse. (c) A spherical covariance matrix has a circular shape. Generated by `gauss_plot_2d.py`.

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

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

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 $\rho \triangleq \frac{\sigma_{12}^2}{\sigma_1 \sigma_2}$.

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) as follows:

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

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

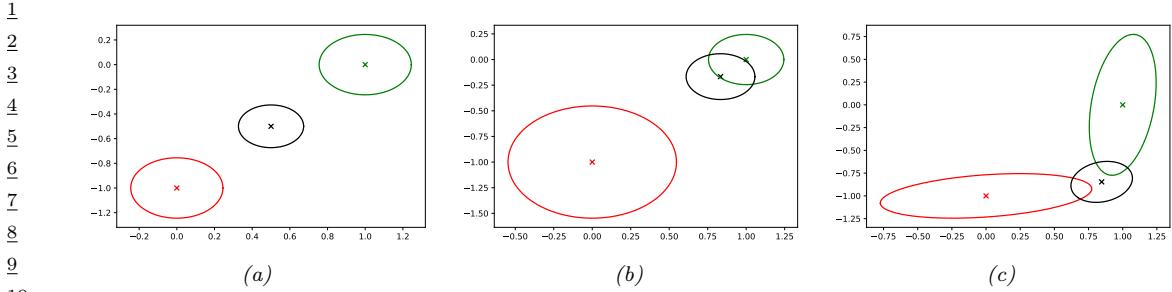


Figure 2.8: We observe $\mathbf{x} = (0, -1)$ (red cross) and $\mathbf{y} = (1, 0)$ (green cross) and estimate $\mathbb{E}[\mathbf{z}|\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}]$ (black cross). (a) Equally reliable sensors, so the posterior mean estimate is in between the two circles. (b) Sensor 2 is 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:

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

$$\boldsymbol{\Sigma}_{z|1}^{-1} = \boldsymbol{\Sigma}_z^{-1} + \mathbf{W}_1^T \boldsymbol{\Sigma}_{11}^{-1} \mathbf{W}_1 \quad (2.68)$$

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

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:

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

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

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

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

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

\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:

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

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

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

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

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

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

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

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

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:

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

2.3.7.6 Product of Gaussians

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

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

where

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

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

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

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

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

In the scalar case, this becomes

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

2.4 Some other multivariate continuous distributions

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

2.4.1 Multivariate Student distribution

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

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

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

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

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

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

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

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:

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

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

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 $\mathbf{x}_n \sim \mathcal{N}(0, \Sigma)$. One can show that the scatter matrix, $\mathbf{S} = \sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^\top$, has a Wishart distribution: $\mathbf{S} \sim \text{Wi}(\Sigma, N)$.

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.

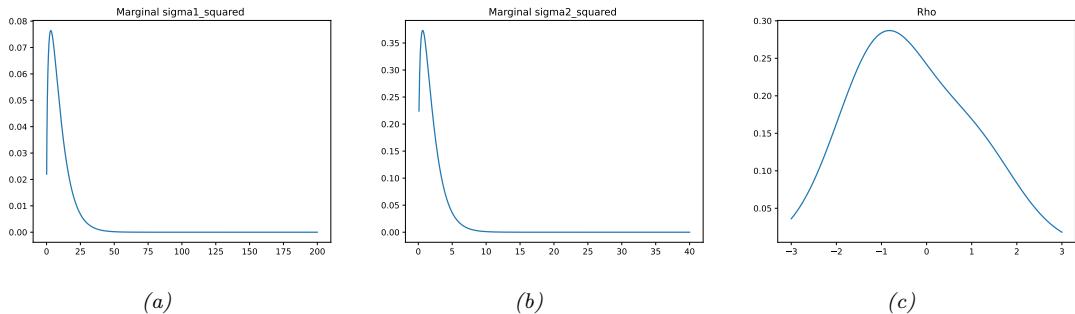
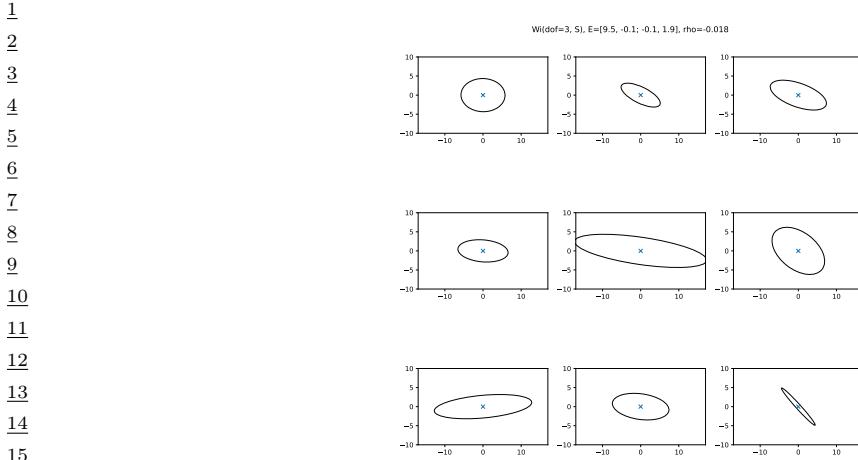
2.4.4.1 Visualizing the Wishart distribution

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

$$R_{ij} = \frac{\Sigma_{ij}}{\sqrt{\Sigma_{ii}\Sigma_{jj}}} \quad (2.114)$$

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.



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.

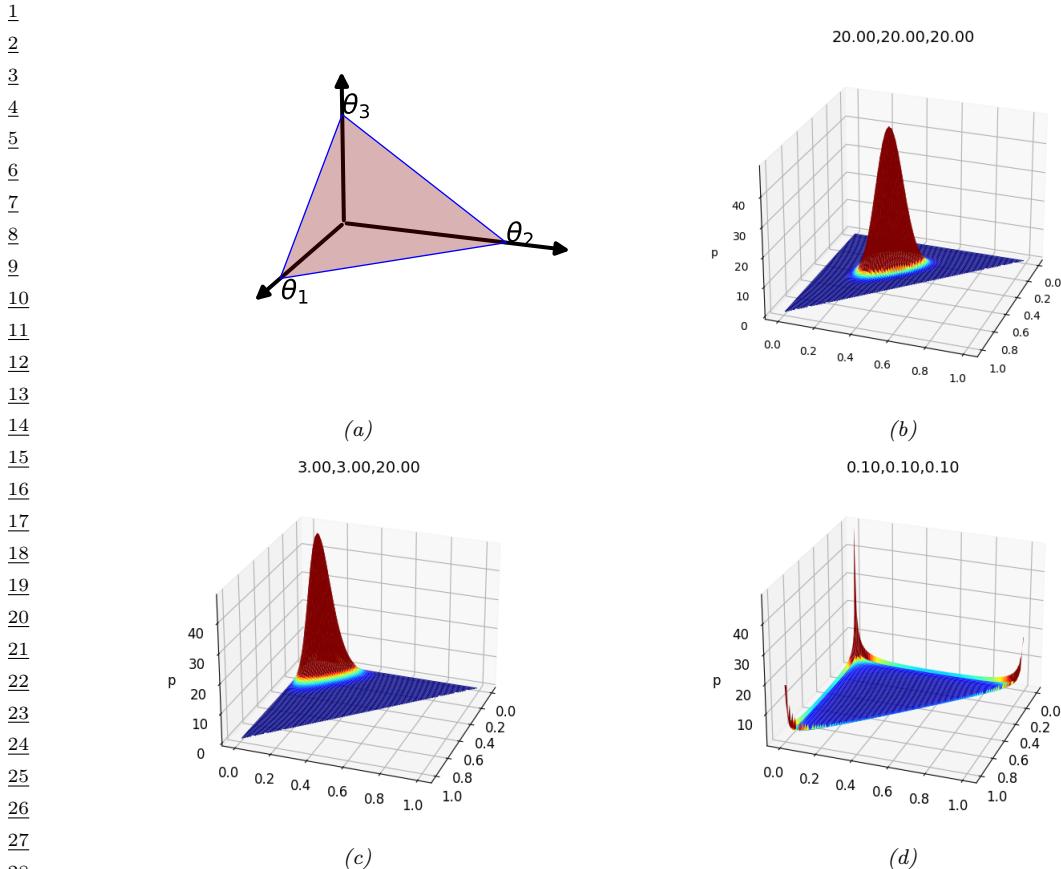


Figure 2.11: (a) The Dirichlet distribution when $K = 3$ defines a distribution over the simplex, which can be represented by the triangular surface. Points on this surface satisfy $0 \leq \theta_c \leq 1$ and $\sum_{c=1}^3 \theta_c = 1$. Generated by `dirichlet_3d_triangle_plot.py`. (b) Plot of the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

2.5 The exponential family

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:

- 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.

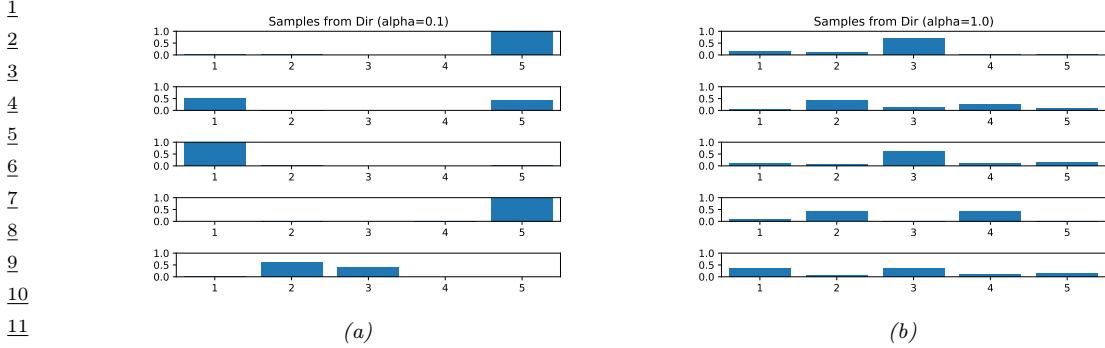


Figure 2.12: Samples from a 5-dimensional symmetric Dirichlet distribution for different parameter values. (a) $\alpha = (0.1, \dots, 0.1)$. This results in very sparse distributions, with many 0s. (b) $\alpha = (1, \dots, 1)$. This results in more uniform (and dense) distributions. Generated by `dirichlet_samples_plot.py`.

- 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, $\mathcal{T}(\mathbf{x}) = \mathbf{x}$, we say this is a **natural exponential family** or **NEF**. In this case, it can be written as

$$p(\mathbf{x}|\boldsymbol{\eta}) = h(\mathbf{x}) \exp[\boldsymbol{\eta}^\top \mathbf{x} - A(\boldsymbol{\eta})] \quad (2.125)$$

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

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

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 $h(\mathbf{x})$ and $\mathcal{T}(\mathbf{x})$ (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:

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

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

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

where $\mathcal{T}(x) = [\mathbb{I}(x=1), \mathbb{I}(x=0)]$, $\boldsymbol{\eta} = [\log(\mu), \log(1-\mu)]$, 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:

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

If the representation is overcomplete, $\boldsymbol{\eta}$ is not uniquely identifiable. It is common to use a **minimal representation**, which means there is a unique $\boldsymbol{\eta}$ 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

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

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

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

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

1 We can recover the mean parameter μ from the canonical parameter η using
2

$$\underline{3} \quad \underline{4} \quad \mu = \sigma(\eta) = \frac{1}{1 + e^{-\eta}} \quad (2.136)$$

5 which we recognize as the logistic (sigmoid) function.
6

8 2.5.2.2 Categorical distribution

9 We can represent the discrete distribution with K categories as follows (where $x_k = \mathbb{I}(x = k)$):
10

$$\underline{11} \quad \underline{12} \quad \text{Cat}(x|\boldsymbol{\mu}) = \prod_{k=1}^K \mu_k^{x_k} = \exp \left[\sum_{k=1}^K x_k \log \mu_k \right] \quad (2.137)$$

$$\underline{14} \quad \underline{15} \quad = \exp \left[\sum_{k=1}^{K-1} x_k \log \mu_k + \left(1 - \sum_{k=1}^{K-1} x_k \right) \log \left(1 - \sum_{k=1}^{K-1} \mu_k \right) \right] \quad (2.138)$$

$$\underline{17} \quad \underline{18} \quad = \exp \left[\sum_{k=1}^{K-1} x_k \log \left(\frac{\mu_k}{1 - \sum_{j=1}^{K-1} \mu_j} \right) + \log \left(1 - \sum_{k=1}^{K-1} \mu_k \right) \right] \quad (2.139)$$

$$\underline{20} \quad \underline{21} \quad = \exp \left[\sum_{k=1}^{K-1} x_k \log \left(\frac{\mu_k}{\mu_K} \right) + \log \mu_K \right] \quad (2.140)$$

23 where $\mu_K = 1 - \sum_{k=1}^{K-1} \mu_k$. We can write this in exponential family form as follows:
24

$$\underline{25} \quad \text{Cat}(x|\boldsymbol{\eta}) = \exp(\boldsymbol{\eta}^\top \mathcal{T}(\mathbf{x}) - A(\boldsymbol{\eta})) \quad (2.141)$$

$$\underline{27} \quad \boldsymbol{\eta} = [\log \frac{\mu_1}{\mu_K}, \dots, \log \frac{\mu_{K-1}}{\mu_K}] \quad (2.142)$$

$$\underline{29} \quad A(\boldsymbol{\eta}) = -\log(\mu_K) \quad (2.143)$$

$$\underline{30} \quad \mathcal{T}(x) = [\mathbb{I}(x=1), \dots, \mathbb{I}(x=K-1)] \quad (2.144)$$

$$\underline{31} \quad h(x) = 1 \quad (2.145)$$

32 We can recover the mean parameters from the canonical parameters using
33

$$\underline{34} \quad \underline{35} \quad \mu_k = \frac{e^{\eta_k}}{1 + \sum_{j=1}^{K-1} e^{\eta_j}} \quad (2.146)$$

37 If we define $\eta_K = 0$, we can rewrite this as follows:
38

$$\underline{39} \quad \underline{40} \quad \mu_k = \frac{e^{\eta_k}}{\sum_{j=1}^K e^{\eta_j}} \quad (2.147)$$

42 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
43

$$\underline{45} \quad \underline{46} \quad \mu_K = 1 - \frac{\sum_{k=1}^{K-1} e^{\eta_k}}{1 + \sum_{k=1}^{K-1} e^{\eta_k}} = \frac{1}{1 + \sum_{k=1}^{K-1} e^{\eta_k}} \quad (2.148)$$

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
10 The univariate Gaussian is usually written as follows:

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

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

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

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

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

24
25 $A(\boldsymbol{\eta}) = \frac{\mu^2}{2\sigma^2} + \log \sigma = \frac{-\eta_1^2}{4\eta_2} - \frac{1}{2} \log(-2\eta_2)$ (2.154)
26

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

29 The moment parameters are
30

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

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
37

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

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

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

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

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 $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$. The corresponding pdf is given by

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

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

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

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

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

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

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:

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

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

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

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

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

where

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

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

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

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

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

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

(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

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

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

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

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

Hence for an exponential family model we have

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

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 $A(\boldsymbol{\eta})$ 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.,

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

The set of valid moment parameters is given by

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

for some distribution p .

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

$$\mathbf{m} = \nabla_{\boldsymbol{\eta}} A(\boldsymbol{\eta}) \quad (2.197)$$

If the family is minimal, one can show that

$$\boldsymbol{\eta} = \nabla_{\mathbf{m}} A^*(\mathbf{m}) \quad (2.198)$$

where $A^*(\mathbf{m})$ is the convex conjugate of A :

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

Thus the pair of operators $(\nabla A, \nabla A^*)$ lets us go back and forth between the natural parameters $\boldsymbol{\eta} \in \Omega$ 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:

$$B_A(\boldsymbol{\lambda}_1 || \boldsymbol{\lambda}_2) = A(\boldsymbol{\lambda}_1) - A(\boldsymbol{\lambda}_2) - (\boldsymbol{\lambda}_1 - \boldsymbol{\lambda}_2)^T \nabla_{\boldsymbol{\lambda}} A(\boldsymbol{\lambda}_2) \quad (2.200)$$

$$B_{A^*}(\boldsymbol{\mu}_1 || \boldsymbol{\mu}_2) = A(\boldsymbol{\mu}_1) - A(\boldsymbol{\mu}_2) - (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \nabla_{\boldsymbol{\mu}} A(\boldsymbol{\mu}_2) \quad (2.201)$$

$$(2.202)$$

1 **2.5.5 MLE for the exponential family**

2 The likelihood of an exponential family model has the form

3

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

4

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

6

7

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

8

9 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 $\mathcal{T}(\mathcal{D}) = [\sum_n x_n, \sum_n x_n^2]$.

10 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

11

12

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

13

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

15

16

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

17

18 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:

19

20

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

21

22 For a single data case, this becomes

23

24

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

25

26 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

27

28

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

29

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

31

32

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

33

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.

¹ **Theorem 2.6.1.** If $\log p(\mathbf{x}|\boldsymbol{\theta})$ is twice differentiable, and under certain regularity conditions, the
² FIM is equal to the expected Hessian of the NLL, i.e.,
³

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

⁴ Before we prove this result, we establish the following important lemma.
⁵

⁶ **Lemma 1.** The expected value of the score function is zero, i.e.,
⁷

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

⁸ We will prove this lemma in the scalar case. First, note that since $\int p(x|\theta)dx = 1$, we have
⁹

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

¹⁰ Combining this with the identity
¹¹

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

¹² we have
¹³

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

¹⁴ Now we return to the proof of our main theorem. For simplicity, we will focus on the scalar case,
¹⁵ following the presentation of [Ric95, p263].
¹⁶

¹⁷ *Proof.* Taking derivatives of Equation (2.226), we have
¹⁸

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

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

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

¹⁹ and hence
²⁰

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

²¹ as claimed. \square
²²

²³ Now consider the Hessian of the NLL given N iid samples $\mathcal{D} = \{\mathbf{x}_n : n = 1 : N\}$:
²⁴

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

²⁵ From the above theorem, we have
²⁶

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

²⁷ We will use this result later in this chapter.
²⁸

2.6.3 Examples

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

2.6.3.1 FIM for the Binomial

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

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

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

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

The gradient of the score function is

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

Hence the Fisher information is given by

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

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

2.6.3.2 FIM for the Gaussian

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

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

The partial derivatives are given by

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

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

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

and hence

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

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

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

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

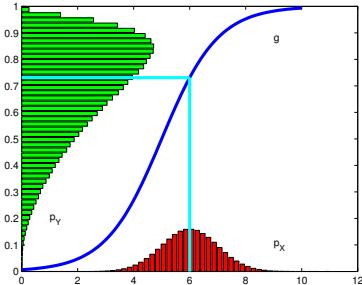
and hence

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

so

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

1
2
3
4
5
6
7
8
9
10



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

14

15

16 2.7 Transformations of random variables

17

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

20

21

2.7.1 Invertible transformations (bijections)

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

24

$$25 \quad p_y(\mathbf{y}) = p_x(g(\mathbf{y})) \left| \det [\mathbf{J}(g)(\mathbf{y})] \right| \quad (2.265)$$

26

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

29

30 2.7.2 Monte Carlo approximation

31

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

35

$$36 \quad p_{\mathcal{D}}(\mathbf{y}) = \frac{1}{S} \sum_{s=1}^S \delta(\mathbf{y} - \mathbf{y}^s) \quad (2.266)$$

37

38 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 $p(y)$
39 using Monte Carlo, as shown in Figure 2.13.

40

41

42 **2.7.3 Probability integral transform**

43

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

46

47

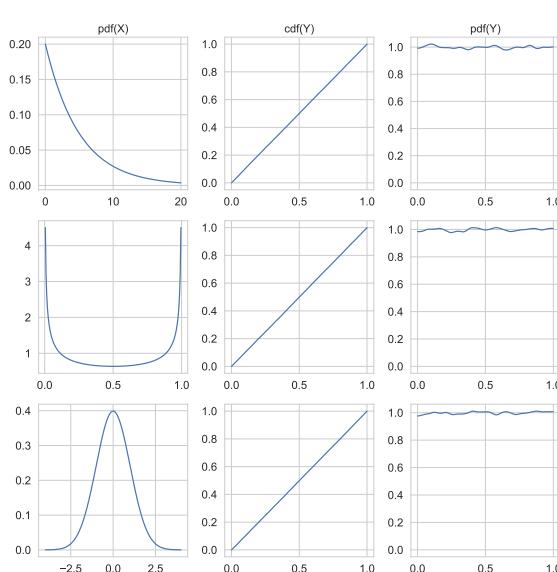


Figure 2.14: Illustration of the probability integral transform. Left column: 3 different pdf's for $p(X)$ from which we sample $x_n \sim p(x)$. Middle column: empirical cdf of $y_n = P_X(x_n)$. Right column: empirical pdf of $p(y_n)$ using a kernel density estimate. Adapted from Figure 11.17 of [MKL11]. 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 $x_n = P_X^{-1}(u_n)$.

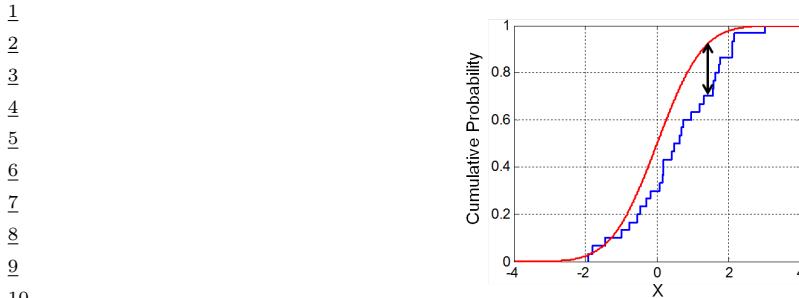


Figure 2.15: Illustration of the Kolmogorov-Smirnov statistic. The red line is a model CDF, the blue line is an empirical CDF, and the black arrow is the K-S statistic. From https://en.wikipedia.org/wiki/Kolmogorov_Smirnov_test. Used with kind permission of Wikipedia author Bscan.

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**.

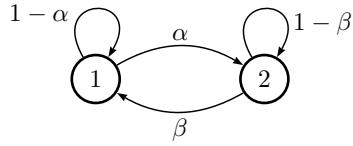
2.8.1 Parameterization

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

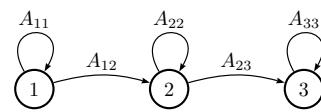
2.8.1.1 Markov transition kernels

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.



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 `bigramp_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

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

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

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

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

¹
² where we define the following counts:

$$\begin{aligned} \text{³
4 } N_j^1 &\triangleq \sum_{i=1}^N \mathbb{I}(x_{i1} = j), \quad N_{jk} \triangleq \sum_{i=1}^N \sum_{t=1}^{T_i-1} \mathbb{I}(x_{i,t} = j, x_{i,t+1} = k), \quad N_j = \sum_k N_{jk} \end{aligned} \quad (2.283)$$

⁵
⁶ Hence we can write the MLE as the normalized counts:

$$\begin{aligned} \text{⁷
8 } \hat{\pi}_j &= \frac{N_j^1}{\sum_{j'} N_{j'}^1}, \quad \hat{A}_{jk} = \frac{N_{jk}}{N_j} \end{aligned} \quad (2.284)$$

⁹
¹⁰ 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.

¹¹
¹² 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*.

¹³ ¹⁴ 2.8.3.2 Sparse data problem

¹⁵
¹⁶ 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

¹⁷
¹⁸ 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]).

¹⁹ ²⁰ 2.8.3.3 MAP estimation

²¹
²² 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

$$\begin{aligned} \text{²³
24 } \hat{A}_{jk} &= \frac{N_{jk} + \alpha}{N_j + K\alpha} \end{aligned} \quad (2.285)$$

²⁵
²⁶ If $\alpha = 1$, this is called **add-one smoothing**.

²⁷
²⁸ 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.

²⁹
³⁰ 8. See <http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html> for details.

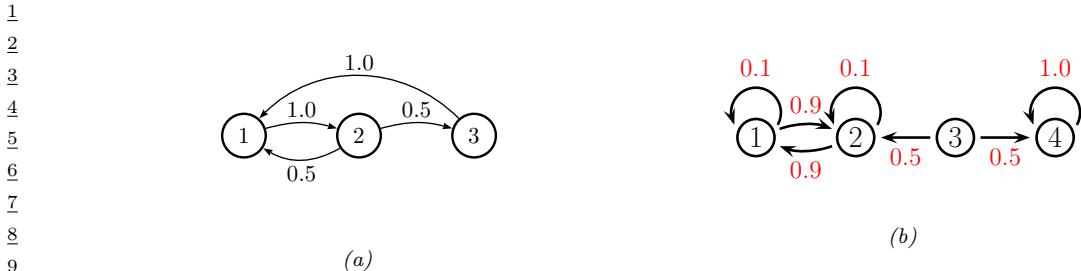


Figure 2.19: Some Markov chains. (a) A 3-state aperiodic chain. (b) A reducible 4-state chain.

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

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

¹ Hence $\pi_1(A_{12} + A_{13}) = \pi_2 A_{21} + \pi_3 A_{31}$. In general, we have

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

⁶ In other words, the probability of being in state i times the net flow out of state i must equal the
⁷ probability of being in each other state j times the net flow from that state into i . These are called
⁸ the **global balance equations**. We can then solve these equations, subject to the constraint that
⁹ $\sum_j \pi_j = 1$, to find the stationary distribution, as we discuss below.

¹¹ 2.8.4.2 Computing the stationary distribution

¹³ To find the stationary distribution, we can just solve the eigenvector equation $\mathbf{A}^T \mathbf{v} = \mathbf{v}$, and then
¹⁴ to set $\boldsymbol{\pi} = \mathbf{v}^T$, where \mathbf{v} is an eigenvector with eigenvalue 1. (We can be sure such an eigenvector
¹⁵ 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}^T
¹⁶ are the same.) Of course, since eigenvectors are unique only up to constants of proportionality, we
¹⁷ must normalize \mathbf{v} at the end to ensure it sums to one.

¹⁸ Note, however, that the eigenvectors are only guaranteed to be real-valued if all entries in the
¹⁹ matrix are strictly positive, $A_{ij} > 0$ (and hence $A_{ij} < 1$, due to the sum-to-one constraint). A more
²⁰ general approach, which can handle chains where some transition probabilities are 0 or 1 (such as
²¹ 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
²² $\boldsymbol{\pi}\mathbf{1}_{K \times 1} = 1$. 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
²³ $\mathbf{r} = [0, 0, \dots, 0, 1]$ is a $1 \times (K + 1)$ vector. However, this is overconstrained, so we will drop the last
²⁴ 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
²⁵ we have to solve this linear system:

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

³⁰ 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
³¹ $\boldsymbol{\pi} = \boldsymbol{\pi}\mathbf{A}$.

³² Unfortunately, not all chains have a stationary distribution, as we explain below.

³⁴ 2.8.4.3 When does a stationary distribution exist?

³⁶ Consider the 4-state chain in Figure 2.19(b). If we start in state 4, we will stay there forever, since
³⁷ 4 is an **absorbing state**. Thus $\boldsymbol{\pi} = (0, 0, 0, 1)$ is one possible stationary distribution. However,
³⁸ 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
³⁹ another possible stationary distribution. If we start in state 3, we could end up in either of the above
⁴⁰ stationary distributions with equal probability. The corresponding transition graph has two disjoint
⁴¹ connected components.

⁴² We see from this example that a necessary condition to have a unique stationary distribution is
⁴³ that the state transition diagram be a singly connected component, i.e., we can get from any state to
⁴⁴ any other state. Such chains are called **irreducible**.

⁴⁵ Now consider the 2-state chain in Figure 2.16(a). This is irreducible provided $\alpha, \beta > 0$. Suppose
⁴⁶ $\alpha = \beta = 0.9$. It is clear by symmetry that this chain will spend 50% of its time in each state. Thus
⁴⁷

$\pi = (0.5, 0.5)$. 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): states 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].)

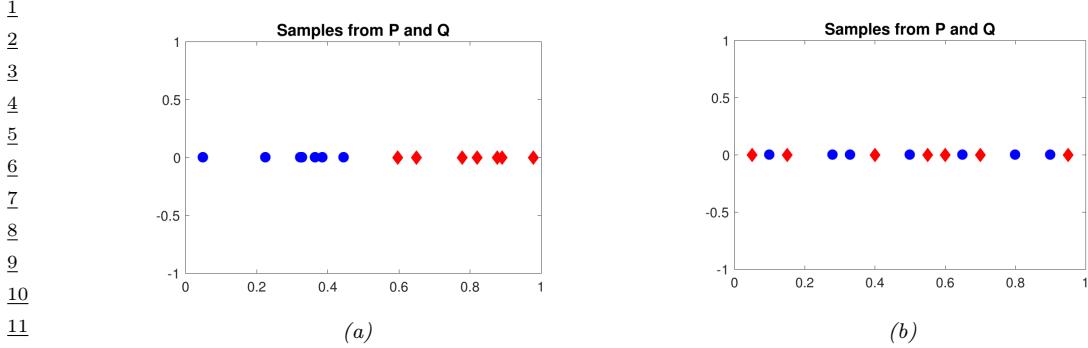


Figure 2.20: Samples from two distributions which are (a) different and (b) similar. From a figure from [GSJ19]. Used with kind permission of Arthur Getton.

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:

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

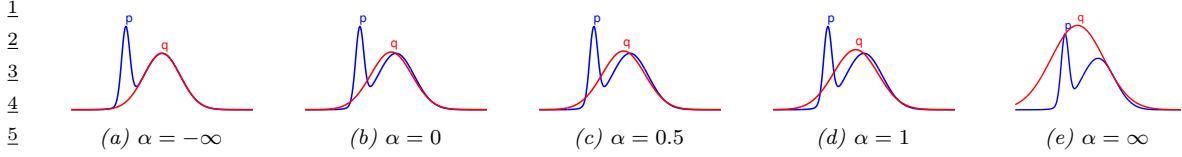


Figure 2.21: The Gaussian q which minimizes α -divergence to p (a mixture of two Gaussians), for varying α . From Figure 1 of [Min05]. Used with kind permission of Tom Minka.

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:

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

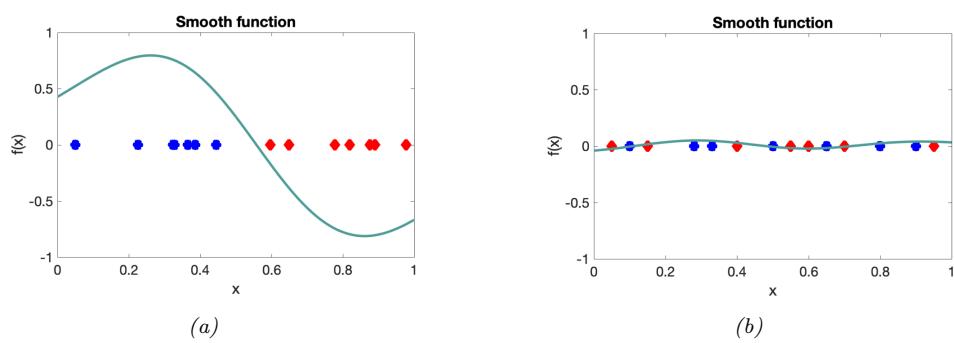


Figure 2.22: A smooth witness function for comparing two distributions which are (a) different and (b) similar. From a figure from [GSJ19]. Used with kind permission of Arthur Getton.

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

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

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

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

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

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

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 \mathbf{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:

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

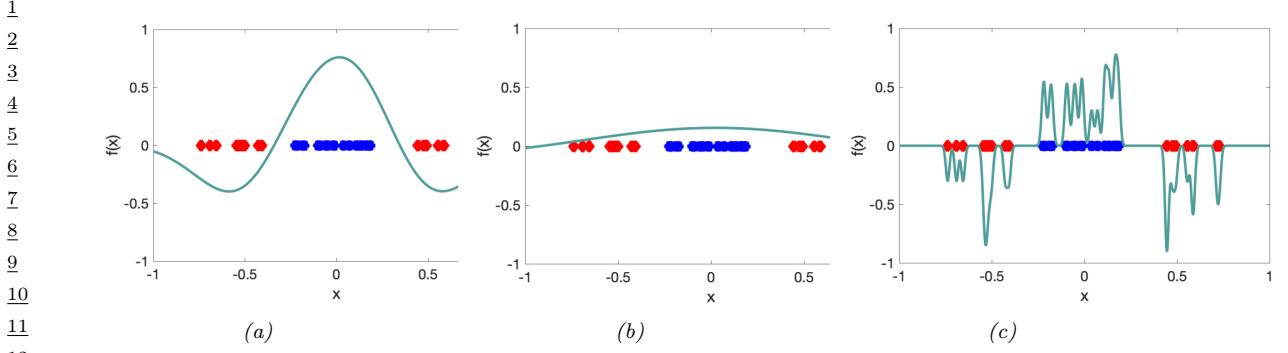


Figure 2.23: Effect of bandwidth parameter σ on the witness function defined by a Gaussian kernel. From a figure from [GSJ19]. Used with kind permission of Dougal Sutherland.

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

$$\text{witness}^2(\mathbf{v}) = (\boldsymbol{\mu}_Q(\mathbf{v}) - \boldsymbol{\mu}_P(\mathbf{v}))^2 \quad (2.311)$$

at a set of test locations $\mathbf{v}_1, \dots, \mathbf{v}_J$ 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 $\boldsymbol{\mu}_P(\mathbf{v}) = \mathbb{E}_{p(\mathbf{x})} [\mathcal{K}(\mathbf{x}, \mathbf{v})]$ can be estimated empirically in $O(N)$ time, and similarly for $\boldsymbol{\mu}_Q(\mathbf{v})$.

A normalized version of UME, known as NME, is presented in [Jit+16]. By maximizing NME wrt the locations \mathbf{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 $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathcal{K}_\sigma(\mathbf{h}(\mathbf{x}), \mathbf{h}(\mathbf{x}'))$,

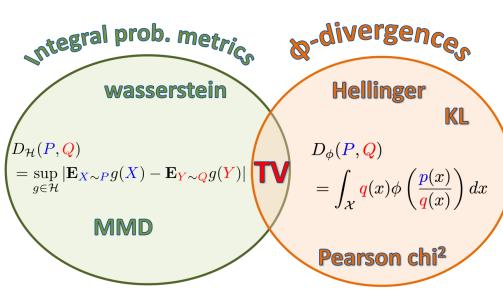


Figure 2.24: Summary of the two main kinds of divergence measures between two probability distributions P and Q . From a figure from [GSJ19]. Used with kind permission of Arthur Getton.

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

12 3.1.4.2 Is Bayes relevant in the “deep learning” era?

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

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 $p(\mathcal{D})$:

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

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

35 3.1.5.1 The MAP estimate gives no measure of uncertainty

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

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

44 where $N = N_1 + N_0$. But this is just a point estimate, with no associated uncertainty. For example,
45 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
46 is that biased? We cannot be sure, because the sample size is so small.

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, $p(\theta) = \text{Beta}(\theta | \bar{\alpha}, \bar{\beta})$, where $\bar{\alpha}$ and $\bar{\beta}$ are known as **pseudo counts**. We will show that the posterior has the form $p(\theta) = \text{Beta}(\theta | \hat{\alpha}, \hat{\beta})$, where $\hat{\alpha} = \bar{\alpha} + N_1$ and $\hat{\beta} = \bar{\beta} + N_0$. 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 be 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 $p(y | \mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(y | \mathbf{w}^T \mathbf{x}, \sigma^2)$, 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

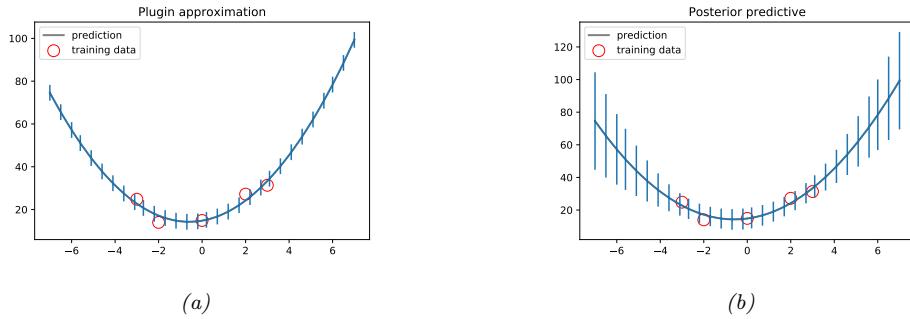


Figure 3.1: Predictions made by a polynomial regression model fit to a small dataset. (a) Plugin approximation to predictive density using the MLE. The curves shows the posterior mean, $\mathbb{E}[y|\mathbf{x}]$, and the error bars show the posterior standard deviation, $\text{std}[y|\mathbf{x}]$, around this mean. (b) Bayesian posterior predictive density, obtained by integrating out the parameters. Generated by [linreg_post_pred_plot.py](#).

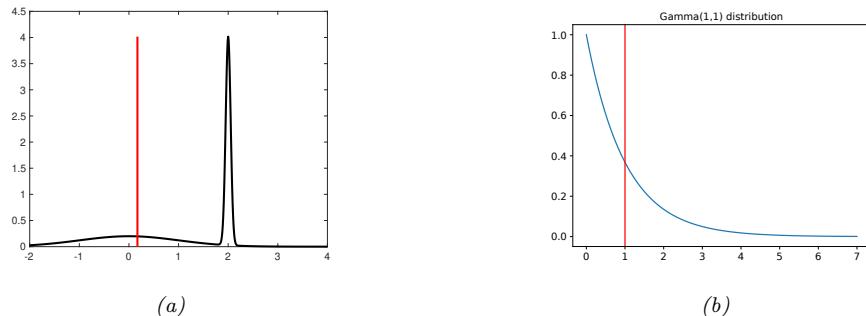


Figure 3.2: Two distributions in which the mode (highest point) is untypical of the distribution; the mean (vertical red line) is a better summary. (a) A bimodal distribution. Generated by [bimodal_dist_plot.py](#). (b) A skewed $\text{Ga}(1, 1)$ distribution. Generated by [gamma_dist_plot.py](#).

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.

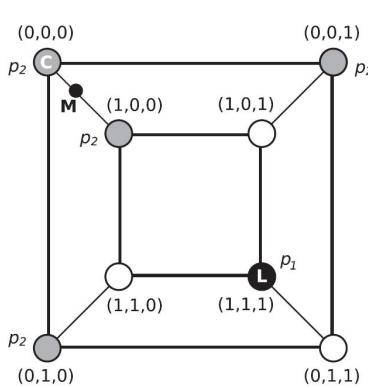


Figure 3.3: A distribution on a discrete space in which the mode (black point L , with probability p_1) is untypical of most of the probability mass (gray circles, with probability $p_2 < p_1$). The small black circle labeled M (near the top left) is the posterior mean, which is not well defined in a 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 Carvalho.

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, $\ell(\theta, \hat{\theta}) = \mathbb{I}(\theta \neq \hat{\theta})$, as we show in Section 3.8.3.1. However, this does not give any “partial credit” for estimating some of the components of θ correctly. An alternative is to use the **Hamming loss**: $\ell(\theta, \hat{\theta}) = \sum_{d=1}^D \mathbb{I}(\theta_d \neq \hat{\theta}_d)$.

1 In this case, one can show that the optimal estimator is the vector of **max marginals**

$$\hat{\boldsymbol{\theta}} = \left[\operatorname{argmax}_{\boldsymbol{\theta}_d} \int_{\boldsymbol{\theta}_{-d}} p(\boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta}_{-d} \right]_{d=1}^D \quad (3.8)$$

7 This is also called the **maximizer of posterior marginals** or **MPM** estimate. Note that computing
8 the max marginals involves marginalization and maximization, and thus depends on the whole
9 distribution; this tends to be more robust than the MAP estimate [MMP87].

11 3.1.5.5 The MAP estimate is not invariant to reparameterization

13 A more subtle problem with MAP estimation is that the result we get depends on how we parameterize
14 the probability distribution, which is not very desirable. For example, when representing a Bernoulli
15 distribution, we should be able to parameterize it in terms of probability of success, or in terms of
16 the log-odds (logit), without that affecting our beliefs.

17 For example, let $\hat{x} = \operatorname{argmax}_x p_x(x)$ be the MAP estimate for x . Now let $y = f(x)$ be a
18 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
19 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
20 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
21 Carlo approximation. The result is shown in Figure 2.13. We see that the original Gaussian for
22 $p(x)$ has become “squashed” by the sigmoid nonlinearity. In particular, we see that the mode of the
23 transformed distribution is not equal to the transform of the original mode.

25 We have seen that the MAP estimate depends on the parameterization. The MLE does not suffer
26 from this since the likelihood is a function, not a probability density. Bayesian inference does not
27 suffer from this problem either, since the change of measure is taken into account when integrating
28 over the parameter space.

30 3.2 Closed-form analysis using conjugate priors

32 In this section, we consider the problem of inferring the posterior for parameters of simple probability
33 models. These will form the foundations for many more complex models. We will use priors that are
34 **conjugate** to the likelihood. This is defined as follows: a prior $p(\boldsymbol{\theta}) \in \mathcal{F}$ is a conjugate prior for
35 a likelihood function $p(\mathcal{D}|\boldsymbol{\theta})$ if the posterior is in the same parameterized family as the prior, i.e.,
36 $p(\boldsymbol{\theta}|\mathcal{D}) \in \mathcal{F}$. In other words, \mathcal{F} is closed under Bayesian updating. If the family \mathcal{F} corresponds to
37 the exponential family (defined in Section 2.5), then the computations can be performed in closed
38 form. In the sections below, we give some common examples of this framework, which we will use
39 later in the book.

41 3.2.1 The binomial model

43 Suppose we toss a coin N times, and want to infer the probability of heads. Let $y_n = 1$ denote the
44 event that the n 'th trial was heads, $y_n = 0$ represent the event that the n 'th trial was tails, and let
45 $\mathcal{D} = \{y_n : n = 1 : N\}$ be all the data. We assume $y_n \sim \text{Ber}(\theta)$, where $\theta \in [0, 1]$ is the rate parameter
46 (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 $p(\mathbf{y}|\theta)$. 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:

$$p(\theta) \propto \theta^{\alpha-1} (1-\theta)^{\beta-1} = \text{Beta}(\theta | \alpha, \beta) \quad (3.11)$$

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:

$$p(\theta|\mathcal{D}) \propto \theta^{N_1} (1-\theta)^{N_0} \theta^{\alpha-1} (1-\theta)^{\beta-1} \quad (3.12)$$

$$\propto \text{Beta}(\theta | \alpha + N_1, \beta + N_0) \quad (3.13)$$

$$= \text{Beta}(\theta | \hat{\alpha}, \hat{\beta}) \quad (3.14)$$

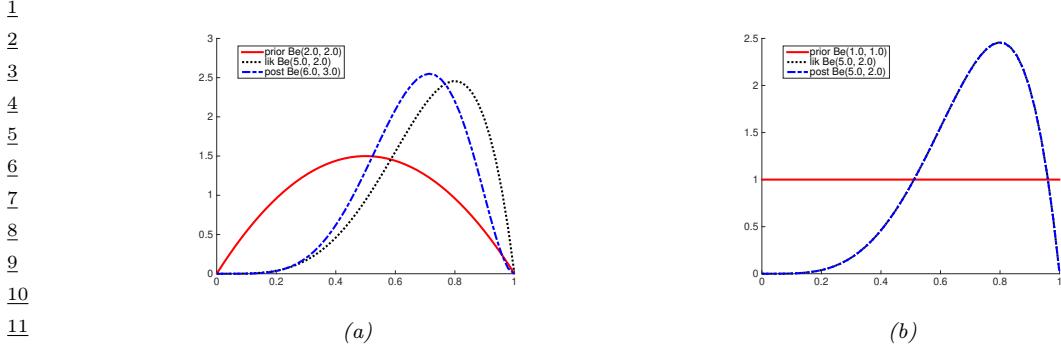


Figure 3.4: Updating a Beta prior with a Bernoulli likelihood with sufficient statistics $N_1 = 4, N_0 = 1$. (a) Beta(2,2) prior. (b) Uniform Beta(1,1) prior. Generated by `beta_binom_post_plot.py`.

where $\tilde{\alpha} \triangleq \check{\alpha} + N_1$ and $\tilde{\beta} \triangleq \check{\beta} + N_0$ 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 $\lambda = \frac{\check{N}}{N + \check{N}}$ 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

$$\mathbb{V}[\theta|\mathcal{D}] = \frac{\widehat{\alpha}\widehat{\beta}}{(\widehat{\alpha} + \widehat{\beta})^2(\widehat{\alpha} + \widehat{\beta} + 1)} = \mathbb{E}[\theta|\mathcal{D}]^2 \frac{\widehat{\beta}}{\widehat{\alpha}(1 + \widehat{\alpha} + \widehat{\beta})} \quad (3.26)$$

where $\widehat{\alpha} = \check{\alpha} + N_1$ and $\widehat{\beta} = \check{\beta} + N_0$. If $N \gg \check{\alpha} + \check{\beta}$, this simplifies to

$$\mathbb{V}[\theta|\mathcal{D}] \approx \frac{N_1 N_0}{N^3} = \frac{\widehat{\theta}(1 - \widehat{\theta})}{N} \quad (3.27)$$

where $\widehat{\theta}$ is the MLE. Hence the standard error is given by

$$\sigma = \sqrt{\mathbb{V}[\theta|\mathcal{D}]} \approx \sqrt{\frac{\widehat{\theta}(1 - \widehat{\theta})}{N}} \quad (3.28)$$

We see that the uncertainty goes down at a rate of $1/\sqrt{N}$. We also see that the uncertainty (variance) is maximized when $\widehat{\theta} = 0.5$, and is minimized when $\widehat{\theta}$ 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, $\widehat{\theta}(\mathcal{D})$, and then to plug that parameter back into the model and use $p(y|\widehat{\theta})$ 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 $\widehat{\theta} = 3/3 = 1$. 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

$$p(y = 1|\mathcal{D}) = \int_0^1 p(y = 1|\theta)p(\theta|\mathcal{D})d\theta \quad (3.29)$$

$$= \int_0^1 \theta \text{Beta}(\theta|\widehat{\alpha}, \widehat{\beta})d\theta = \mathbb{E}[\theta|\mathcal{D}] = \frac{\widehat{\alpha}}{\widehat{\alpha} + \widehat{\beta}} \quad (3.30)$$

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.

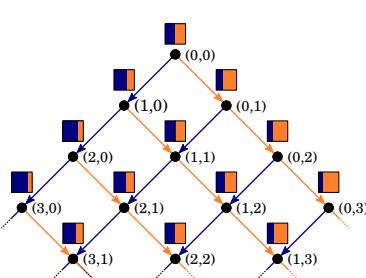


Figure 3.5: Illustration of sequential Bayesian updating for the beta-Bernoulli model. Each colored box represents the predicted distribution $p(x_t | \mathbf{h}_t)$, where $\mathbf{h}_t = (N_{1,t}, N_{0,t})$ is the sufficient statistic derived from history of observations up until time t , namely the 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 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

$$p(\theta|\mathcal{D}) \approx \delta(\theta - \hat{\theta}) \quad (3.36)$$

$$p(y|\mathcal{D}, M) = \int_0^1 \text{Bin}(y|M, \theta) p(\theta|\mathcal{D}) d\theta = \text{Bin}(y|M, \hat{\theta}) \quad (3.37)$$

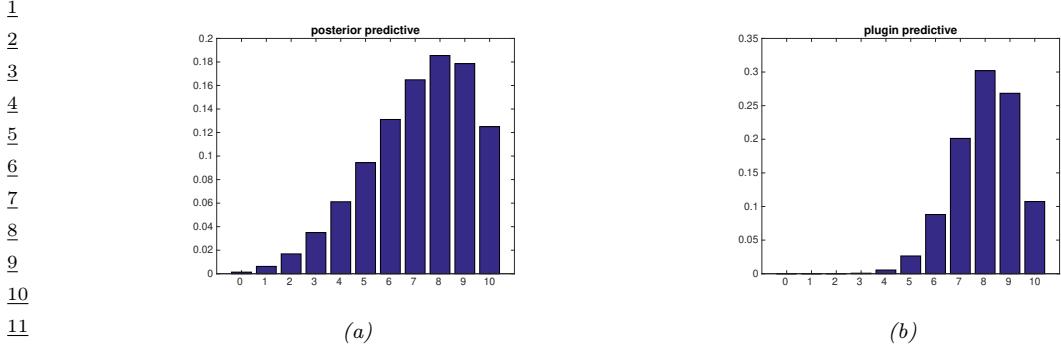


Figure 3.6: (a) Posterior predictive distributions for 10 future trials after seeing $N_1 = 4$ heads and $N_0 = 1$ tails. (b) Plug-in approximation based on the same data. In both cases, we use a uniform prior. Generated by [beta_binom_post_pred_plot.py](#).

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

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})} \quad (3.39)$$

$$= \frac{1}{p(\mathcal{D})} \left[\frac{1}{B(a, b)} \boldsymbol{\theta}^{a-1} (1-\boldsymbol{\theta})^{b-1} \right] \left[\binom{N}{N_1} \boldsymbol{\theta}^{N_1} (1-\boldsymbol{\theta})^{N_0} \right] \quad (3.40)$$

$$= \binom{N}{N_1} \frac{1}{p(\mathcal{D})} \frac{1}{B(a, b)} [\boldsymbol{\theta}^{a+N_1-1} (1-\boldsymbol{\theta})^{b+N_0-1}] \quad (3.41)$$

1 So

2

$$\frac{1}{B(a + N_1, b + N_0)} = \binom{N}{N_1} \frac{1}{p(\mathcal{D})} \frac{1}{B(a, b)} \quad (3.42)$$

3

$$p(\mathcal{D}) = \binom{N}{N_1} \frac{B(a + N_1, b + N_0)}{B(a, b)} \quad (3.43)$$

4 The marginal likelihood for the beta-Bernoulli model is the same as above, except it is missing the
5 $\binom{N}{N_1}$ term.

6 3.2.1.11 Mixtures of conjugate priors

7 The beta distribution is a conjugate prior for the binomial likelihood, which enables us to easily
8 compute the posterior in closed form, as we have seen. However, this prior is rather restrictive. For
9 example, suppose we want to predict the outcome of a coin toss at a casino, and we believe that the
10 coin may be fair, but may equally likely be biased towards heads. This prior cannot be represented
11 by a beta distribution. Fortunately, it can be represented as a **mixture of beta distributions**.
12 For example, we might use

13 $p(\theta) = 0.5 \text{ Beta}(\theta|20, 20) + 0.5 \text{ Beta}(\theta|30, 10) \quad (3.44)$

14 If θ comes from the first distribution, the coin is fair, but if it comes from the second, it is biased
15 towards heads.

16 We can represent a mixture by introducing a latent indicator variable h , where $h = k$ means that
17 θ comes from mixture component k . The prior has the form

18 $p(\theta) = \sum_k p(h = k)p(\theta|h = k) \quad (3.45)$

19 where each $p(\theta|h = k)$ is conjugate, and $p(h = k)$ are called the (prior) mixing weights. One can
20 show that the posterior can also be written as a mixture of conjugate distributions as follows:

21 $p(\theta|\mathcal{D}) = \sum_k p(h = k|\mathcal{D})p(\theta|\mathcal{D}, h = k) \quad (3.46)$

22 where $p(h = k|\mathcal{D})$ are the posterior mixing weights given by

23 $p(h = k|\mathcal{D}) = \frac{p(h = k)p(\mathcal{D}|h = k)}{\sum_{k'} p(h = k')p(\mathcal{D}|h = k')} \quad (3.47)$

24 Here the quantity $p(\mathcal{D}|h = k)$ is the marginal likelihood for mixture component k (see Section 3.2.1.10).

25 Returning to our example above, if we have the prior in Equation (3.44), and we observe $N_1 = 20$
26 heads and $N_0 = 10$ tails, then, using Equation (3.43), the posterior becomes

27 $p(\theta|\mathcal{D}) = 0.346 \text{ Beta}(\theta|40, 30) + 0.654 \text{ Beta}(\theta|30, 20) \quad (3.48)$

28 See Figure 3.7 for an illustration.

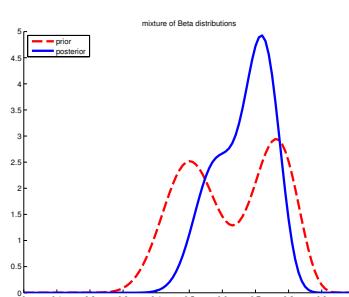


Figure 3.7: A mixture of two Beta distributions. Generated by `mixbetademo.py`.

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

$$B(\bar{\alpha}) \triangleq \frac{\prod_{k=1}^K \Gamma(\bar{\alpha}_k)}{\Gamma(\sum_{k=1}^K \bar{\alpha}_k)} \quad (3.52)$$

3.2.2.3 Posterior

We can combine the multinomial likelihood and Dirichlet prior to compute the posterior, as follows:

$$p(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta) \text{Dir}(\theta|\bar{\alpha}) \quad (3.53)$$

$$= \left[\prod_k \theta_k^{N_k} \right] \left[\prod_k \theta_k^{\bar{\alpha}_k - 1} \right] \quad (3.54)$$

$$= \text{Dir}(\theta|\bar{\alpha}_1 + N_1, \dots, \bar{\alpha}_K + N_K) \quad (3.55)$$

$$= \text{Dir}(\theta|\hat{\alpha}) \quad (3.56)$$

where $\hat{\alpha}_k = \bar{\alpha}_k + N_k$ 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 $\bar{\alpha}_k = 1$, 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

$$p(y=k|\mathcal{D}) = \int p(y=k|\theta)p(\theta|\mathcal{D})d\theta \quad (3.60)$$

$$= \int \theta_k p(\theta_k|\mathcal{D})d\theta_k = \mathbb{E}[\theta_k|\mathcal{D}] = \frac{\hat{\alpha}_k}{\sum_{k'} \hat{\alpha}_{k'}} \quad (3.61)$$

In other words, the posterior predictive distribution is given by

$$p(y|\mathcal{D}) = \text{Cat}(y|\bar{\theta}) \quad (3.62)$$

where $\bar{\theta} \triangleq \mathbb{E}[\theta|\mathcal{D}]$ 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 $\bar{\alpha}_c = 2$.

$\frac{1}{1}$ Equation (3.61) gives the probability of a single future event, conditioned on past observations
 $\frac{2}{2}$ $\mathbf{y} = (y_1, \dots, y_N)$. In some cases, we want to know the probability of observing a batch of future data,
 $\frac{3}{3}$ say $\tilde{\mathbf{y}} = (\tilde{y}_1, \dots, \tilde{y}_M)$. We can compute this as follows:
 $\frac{4}{4}$

$$\frac{5}{5} p(\tilde{\mathbf{y}}|\mathbf{y}) = \frac{p(\tilde{\mathbf{y}}, \mathbf{y})}{p(\mathbf{y})} \quad (3.63)$$

$\frac{6}{6}$ The denominator is the marginal likelihood of the training data, and the numerator is the marginal
 $\frac{7}{7}$ likelihood of the training and future test data. We discuss how to compute such marginal likelihoods
 $\frac{8}{8}$ in Section 3.2.2.5.

$\frac{9}{9}$ 3.2.2.5 Marginal likelihood

$\frac{10}{10}$ By the same reasoning as in Section 3.2.1.10, one can show that the marginal likelihood for the
 $\frac{11}{11}$ Dirichlet-categorical model is given by

$$\frac{12}{12} p(\mathcal{D}) = \frac{B(\mathbf{N} + \boldsymbol{\alpha})}{B(\boldsymbol{\alpha})} \quad (3.64)$$

$\frac{13}{13}$ where

$$\frac{14}{14} B(\boldsymbol{\alpha}) = \frac{\prod_{k=1}^K \Gamma(\alpha_k)}{\Gamma(\sum_k \alpha_k)} \quad (3.65)$$

$\frac{15}{15}$ Hence we can rewrite the above result in the following form, which is what is usually presented in
 $\frac{16}{16}$ the literature:

$$\frac{17}{17} p(\mathcal{D}) = \frac{\Gamma(\sum_k \alpha_k)}{\Gamma(N + \sum_k \alpha_k)} \prod_k \frac{\Gamma(N_k + \alpha_k)}{\Gamma(\alpha_k)} \quad (3.66)$$

$\frac{18}{18}$ 3.2.3 The univariate Gaussian model

$\frac{19}{19}$ In this section, we derive the posterior $p(\mu, \sigma^2 | \mathcal{D})$ for a univariate Gaussian. For simplicity, we consider
 $\frac{20}{20}$ this in three steps: inferring just μ , inferring just σ^2 , and then inferring both. See Section 3.2.4 for
 $\frac{21}{21}$ the multivariate case.

$\frac{22}{22}$ 3.2.3.1 Posterior of μ given σ^2

$\frac{23}{23}$ If σ^2 is a known constant, the likelihood for μ has the form

$$\frac{24}{24} p(\mathcal{D}|\mu) \propto \exp \left(-\frac{1}{2\sigma^2} \sum_{n=1}^N (y_n - \mu)^2 \right) \quad (3.67)$$

One can show that the conjugate prior is another Gaussian, $\mathcal{N}(\mu | \tilde{m}, \tilde{\tau}^2)$. Applying Bayes' rule for Gaussians (Equation (2.59)), we find that the corresponding posterior is given by

$$p(\mu | \mathcal{D}, \sigma^2) = \mathcal{N}(\mu | \tilde{m}, \tilde{\tau}^2) \quad (3.68)$$

$$\tilde{\tau}^2 = \frac{1}{\frac{N}{\sigma^2} + \frac{1}{\tilde{\tau}^2}} = \frac{\sigma^2 \tilde{\tau}^2}{N \tilde{\tau}^2 + \sigma^2} \quad (3.69)$$

$$\tilde{m} = \tilde{\tau}^2 \left(\frac{\tilde{m}}{\tilde{\tau}^2} + \frac{N \bar{y}}{\sigma^2} \right) = \frac{\sigma^2}{N \tilde{\tau}^2 + \sigma^2} \tilde{m} + \frac{N \tilde{\tau}^2}{N \tilde{\tau}^2 + \sigma^2} \bar{y} \quad (3.70)$$

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 $\tilde{\lambda} = 1/\tilde{\tau}^2$ be the precision of the prior. We can then rewrite the posterior as follows:

$$p(\mu | \mathcal{D}, \lambda) = \mathcal{N}(\mu | \tilde{m}, \tilde{\lambda}^{-1}) \quad (3.71)$$

$$\tilde{\lambda} = \tilde{\lambda} + N\lambda \quad (3.72)$$

$$\tilde{m} = \frac{N\lambda\bar{y} + \tilde{\lambda}\tilde{m}}{\tilde{\lambda}} = \frac{N\lambda}{N\lambda + \tilde{\lambda}} \bar{y} + \frac{\tilde{\lambda}}{N\lambda + \tilde{\lambda}} \tilde{m} \quad (3.73)$$

These equations are quite intuitive: the posterior precision $\tilde{\lambda}$ is the prior precision $\tilde{\lambda}$ plus N units of measurement precision λ . Also, the posterior mean \tilde{m} is a convex combination of the empirical mean \bar{y} and the prior mean \tilde{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 ($\tilde{\lambda}$ is small relative to λ), we put more weight on the empirical mean. If the prior is strong relative to the signal strength ($\tilde{\lambda}$ 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:

$$\tilde{m} = \frac{\tilde{\lambda}}{\tilde{\lambda} + \lambda} \tilde{m} + \frac{\lambda}{\tilde{\lambda} + \lambda} y \quad (3.74)$$

$$= \tilde{m} + \frac{\lambda}{\tilde{\lambda}} (y - \tilde{m}) \quad (3.75)$$

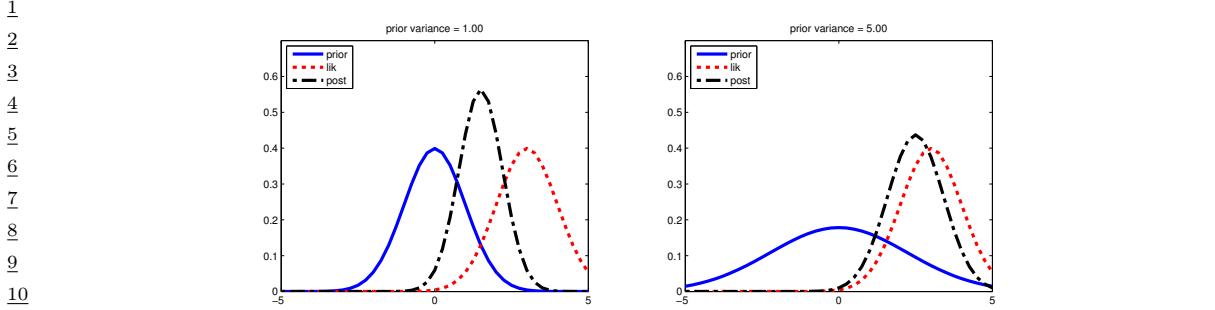
$$= y - \frac{\lambda}{\tilde{\lambda}} (y - \tilde{m}) \quad (3.76)$$

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 $w = \tilde{\lambda}/\lambda$. Then we have

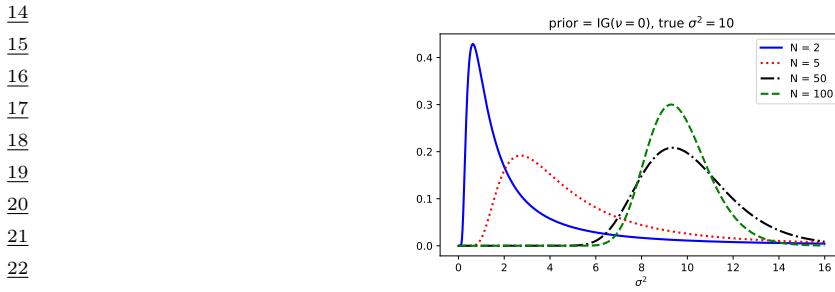
$$\tilde{m} = y - w(y - \tilde{m}) = (1 - w)y + w \tilde{m} \quad (3.77)$$

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.

3.2. CLOSED-FORM ANALYSIS USING CONJUGATE PRIORS



11
12 Figure 3.8: Inferring the mean of a univariate Gaussian with known σ^2 . (a) Using strong prior, $p(\mu) =$
12 $\mathcal{N}(\mu|0, 1)$. (b) Using weak prior, $p(\mu) = \mathcal{N}(\mu|0, 5)$. Generated by [gaussInferParamsMean1d.py](#).



24 Figure 3.9: Sequential updating of the posterior for σ^2 starting from an uninformative prior. The data
25 was generated from a Gaussian with known mean $\mu = 5$ and unknown variance $\sigma^2 = 10$. Generated by
26 [gauss_seq_update_sigma_1d.py](#)

29 3.2.3.2 Posterior of σ^2 given μ

30 If μ is a known constant, the likelihood for σ^2 has the form

$$32 \quad p(\mathcal{D}|\sigma^2) \propto (\sigma^2)^{-N/2} \exp\left(-\frac{1}{2\sigma^2} \sum_{n=1}^N (y_n - \mu)^2\right) \quad (3.78)$$

35 where we can no longer ignore the $1/(\sigma^2)$ term in front. The standard conjugate prior is the inverse
36 Gamma distribution (Section 2.2.2.8), given by

$$37 \quad 38 \quad \text{IG}(\sigma^2 | \check{a}, \check{b}) = \frac{\check{b}^{\check{a}}}{\Gamma(\check{a})} (\sigma^2)^{-(\check{a}+1)} \exp(-\frac{\check{b}}{\sigma^2}) \quad (3.79)$$

40 Multiplying the likelihood and the prior, we see that the posterior is also IG:

$$41 \quad 42 \quad p(\sigma^2 | \mu, \mathcal{D}) = \text{IG}(\sigma^2 | \hat{a}, \hat{b}) \quad (3.80)$$

$$43 \quad \hat{a} = \check{a} + N/2 \quad (3.81)$$

$$44 \quad 45 \quad \hat{b} = \check{b} + \frac{1}{2} \sum_{n=1}^N (y_n - \mu)^2 \quad (3.82)$$

¹ See Figure 3.9 for an illustration.

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**:

$$\chi^{-2}(\sigma^2 | \tilde{\nu}, \tilde{\tau}^2) = \text{IG}(\sigma^2 | \frac{\tilde{\nu}}{2}, \frac{\tilde{\nu} \tilde{\tau}^2}{2}) \propto (\sigma^2)^{-\tilde{\nu}/2-1} \exp(-\frac{\tilde{\nu} \tilde{\tau}^2}{2\sigma^2}) \quad (3.83)$$

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

$$p(\sigma^2 | \mathcal{D}, \mu) = \chi^{-2}(\sigma^2 | \hat{\nu}, \hat{\tau}^2) \quad (3.84)$$

$$\hat{\nu} = \tilde{\nu} + N \quad (3.85)$$

$$\hat{\tau}^2 = \frac{\tilde{\nu} \tilde{\tau}^2 + \sum_{n=1}^N (y_n - \mu)^2}{\hat{\nu}} \quad (3.86)$$

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.

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**:

$$\text{NIG}(\mu, \sigma^2 | \tilde{m}, \tilde{\kappa}, \tilde{a}, \tilde{b}) \triangleq \mathcal{N}(\mu | \tilde{m}, \sigma^2 / \tilde{\kappa}) \text{IG}(\sigma^2 | \tilde{a}, \tilde{b}) \quad (3.87)$$

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

$$NI\chi^2(\mu, \sigma^2 | \tilde{m}, \tilde{\kappa}, \tilde{\nu}, \tilde{\tau}^2) \triangleq \mathcal{N}(\mu | \tilde{m}, \sigma^2 / \tilde{\kappa}) \chi^{-2}(\sigma^2 | \tilde{\nu}, \tilde{\tau}^2) \quad (3.88)$$

$$\propto (\frac{1}{\sigma^2})^{(\tilde{\nu}+3)/2} \exp\left(-\frac{\tilde{\nu} \tilde{\tau}^2 + \tilde{\kappa} (\mu - \tilde{m})^2}{2\sigma^2}\right) \quad (3.89)$$

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.

³⁷ One can show (based on Section 3.2.4.3) that the posterior is given by

$$p(\mu, \sigma^2 | \mathcal{D}) = NI\chi^2(\mu, \sigma^2 | \hat{m}, \hat{\kappa}, \hat{\nu}, \hat{\tau}^2) \quad (3.90)$$

$$\hat{m} = \frac{\tilde{\kappa} \tilde{m} + N \bar{x}}{\tilde{\kappa}} \quad (3.91)$$

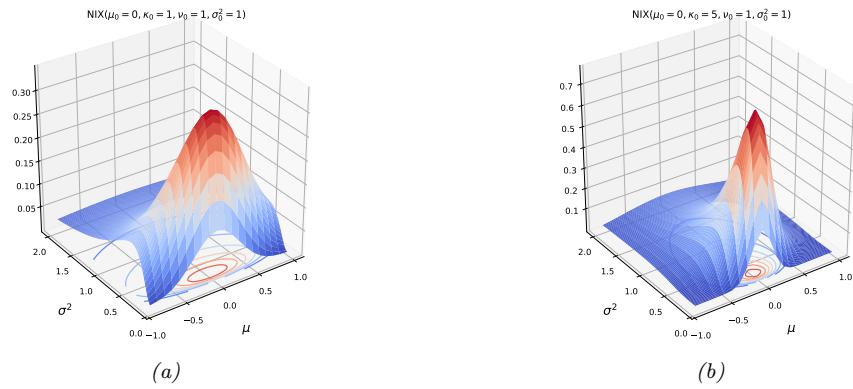
$$\hat{\kappa} = \tilde{\kappa} + N \quad (3.92)$$

$$\hat{\nu} = \tilde{\nu} + N \quad (3.93)$$

$$\hat{\nu} \hat{\tau}^2 = \tilde{\nu} \tilde{\tau}^2 + \sum_{n=1}^N (y_n - \bar{y})^2 + \frac{N \tilde{\kappa}}{\tilde{\kappa} + N} (\tilde{m} - \bar{y})^2 \quad (3.94)$$

3.2. CLOSED-FORM ANALYSIS USING CONJUGATE PRIORS

1
2
3
4
5
6
7
8
9
10
11
12
13
14



15 *Figure 3.10: The $NI\chi^2(\mu, \sigma^2 | m, \kappa, \nu, \sigma^2)$ distribution. m is the prior mean and k is how strongly we believe
16 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
17 that the contour plot (underneath the surface) is shaped like a “squashed egg”. (b) We increase the strength of
18 our belief in the mean by setting $\kappa = 5$, so the distribution for μ around $m = 0$ becomes narrower. Generated
19 by [nix_plots.py](#).*

20
21

22 The interpretation of this is as follows. For μ , the posterior mean \hat{m} is a convex combination of
23 the prior mean \tilde{m} and the MLE \bar{x} ; the strength of this posterior, $\hat{\kappa}$, is the prior strength κ plus the
24 number of data points N . For σ^2 , we work instead with the sum of squares: the posterior sum of
25 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
26 a term due to the discrepancy between the prior mean \tilde{m} and the MLE \bar{y} . The strength of this
27 posterior, $\hat{\nu}$, is the prior strength ν plus the number of data points N ;

28

The posterior marginal for σ^2 is just

$$29 \quad p(\sigma^2 | \mathcal{D}) = \int p(\mu, \sigma^2 | \mathcal{D}) d\mu = \chi^{-2}(\sigma^2 | \hat{\nu}, \hat{\tau}^2) \quad (3.95)$$

30

with the posterior mean given by $\mathbb{E} [\sigma^2 | \mathcal{D}] = \frac{\hat{\nu}}{\hat{\nu}-2} \hat{\tau}^2$.

31

The posterior marginal for μ has a Student distribution, which follows from the fact that the
32 Student distribution is a (scaled) mixture of Gaussians:

33

$$34 \quad p(\mu | \mathcal{D}) = \int p(\mu, \sigma^2 | \mathcal{D}) d\sigma^2 = \mathcal{T}(\mu | \hat{m}, \hat{\tau}^2 / \hat{\kappa}, \hat{\nu}) \quad (3.96)$$

35

with the posterior mean given by $\mathbb{E} [\mu | \mathcal{D}] = \hat{m}$.

36

3.2.3.4 Posterior of μ and σ^2 : uninformative prior

37

If we “know nothing” about the parameters a priori, we can use an uninformative prior. We discuss how
38 to create such priors in Section 3.4. A common approach is to use a Jeffreys prior. In Section 3.4.2.3,
39 we show that the Jeffreys prior for a location and scale parameter has the form

40

$$41 \quad p(\mu, \sigma^2) \propto p(\mu)p(\sigma^2) \propto \sigma^{-2} \quad (3.97)$$

42

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.

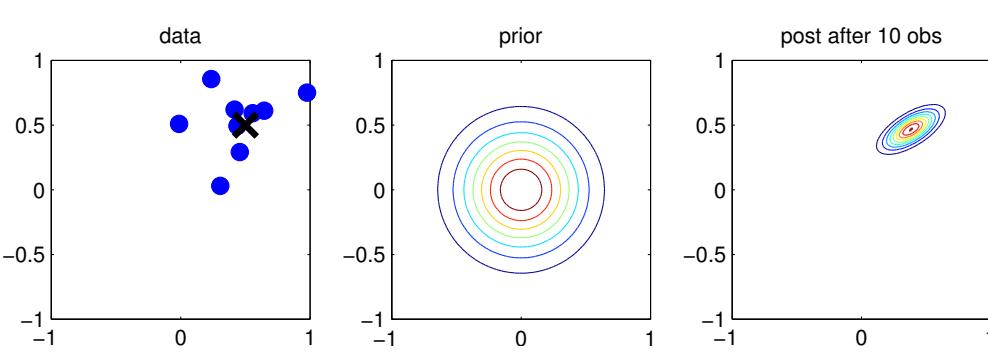


Figure 3.11: Illustration of Bayesian inference for the mean of a 2d Gaussian. (a) The data is generated from $\mathbf{y}_n \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu} = [0.5, 0.5]^T$ and $\boldsymbol{\Sigma} = 0.1[2, 1; 1, 1]$. (b) The prior is $p(\boldsymbol{\mu}) = \mathcal{N}(\boldsymbol{\mu} | \mathbf{0}, 0.1\mathbf{I}_2)$. (c) We show the posterior after 10 data points have been observed. Generated by [gaussInferParamsMean2d.py](#).

3.2.4.2 Posterior of $\boldsymbol{\Sigma}$ given $\boldsymbol{\mu}$

We now discuss how to compute $p(\boldsymbol{\Sigma} | \mathcal{D}, \boldsymbol{\mu})$.

Likelihood

We can rewrite the likelihood as follows:

$$p(\mathcal{D} | \boldsymbol{\mu}, \boldsymbol{\Sigma}) \propto |\boldsymbol{\Sigma}|^{-\frac{N}{2}} \exp\left(-\frac{1}{2}\text{tr}(\mathbf{S}_{\boldsymbol{\mu}} \boldsymbol{\Sigma}^{-1})\right) \quad (3.109)$$

where

$$\mathbf{S}_{\boldsymbol{\mu}} \triangleq \sum_{n=1}^N (\mathbf{y}_n - \boldsymbol{\mu})(\mathbf{y}_n - \boldsymbol{\mu})^T \quad (3.110)$$

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:

$$\text{IW}(\boldsymbol{\Sigma} | \check{\mathbf{S}}, \check{\nu}) \propto |\boldsymbol{\Sigma}|^{-(\check{\nu}+D+1)/2} \exp\left(-\frac{1}{2}\text{tr}(\check{\mathbf{S}} \boldsymbol{\Sigma}^{-1})\right) \quad (3.111)$$

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

$$\begin{aligned} p(\boldsymbol{\Sigma}|\mathcal{D}, \boldsymbol{\mu}) &\propto |\boldsymbol{\Sigma}|^{-\frac{N}{2}} \exp\left(-\frac{1}{2}\text{tr}(\boldsymbol{\Sigma}^{-1}\mathbf{S}_\mu)\right) |\boldsymbol{\Sigma}|^{-(\nu+D+1)/2} \\ &\quad \exp\left(-\frac{1}{2}\text{tr}(\boldsymbol{\Sigma}^{-1}\check{\mathbf{S}})\right) \end{aligned} \tag{3.112}$$

4

$$= |\boldsymbol{\Sigma}|^{-\frac{N+(\nu+D+1)}{2}} \exp\left(-\frac{1}{2}\text{tr}[\boldsymbol{\Sigma}^{-1}(\mathbf{S}_\mu + \check{\mathbf{S}})]\right) \tag{3.113}$$

5

$$= \text{IW}(\boldsymbol{\Sigma}|\hat{\mathbf{S}}, \hat{\nu}) \tag{3.114}$$

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 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 .

9 **3.2.4.3 Posterior of $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$**

10 In this section, we compute $p(\boldsymbol{\mu}, \boldsymbol{\Sigma}|\mathcal{D})$ using a conjugate prior.

11 **Likelihood**

12 The likelihood is given by

13

$$p(\mathcal{D}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \propto |\boldsymbol{\Sigma}|^{-\frac{N}{2}} \exp\left(-\frac{1}{2}\sum_{n=1}^N (\mathbf{y}_n - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{y}_n - \boldsymbol{\mu})\right) \tag{3.117}$$

14 One can show that

15

$$\sum_{n=1}^N (\mathbf{y}_n - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{y}_n - \boldsymbol{\mu}) = \text{tr}(\boldsymbol{\Sigma}^{-1} \mathbf{S}_{\bar{\mathbf{y}}}) + N(\bar{\mathbf{y}} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\bar{\mathbf{y}} - \boldsymbol{\mu}) \tag{3.118}$$

16 where

17

$$\mathbf{S}_{\bar{\mathbf{y}}} \triangleq \sum_{n=1}^N (\mathbf{y}_n - \bar{\mathbf{y}})(\mathbf{y}_n - \bar{\mathbf{y}})^\top = \mathbf{Y}^\top \mathbf{C}_N \mathbf{Y} \tag{3.119}$$

18 is empirical scatter matrix, and \mathbf{C}_N is the **centering matrix**

19

$$\mathbf{C}_N \triangleq \mathbf{I}_N - \frac{1}{N} \mathbf{1}_N \mathbf{1}_N^\top \tag{3.120}$$

20 Hence we can rewrite the likelihood as follows:

21

$$p(\mathcal{D}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \propto |\boldsymbol{\Sigma}|^{-\frac{N}{2}} \exp\left(-\frac{N}{2}(\boldsymbol{\mu} - \bar{\mathbf{y}})^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{\mu} - \bar{\mathbf{y}})\right) \exp\left(-\frac{1}{2}\text{tr}(\boldsymbol{\Sigma}^{-1} \mathbf{S}_{\bar{\mathbf{y}}})\right) \tag{3.121}$$

22 We will use this form below.

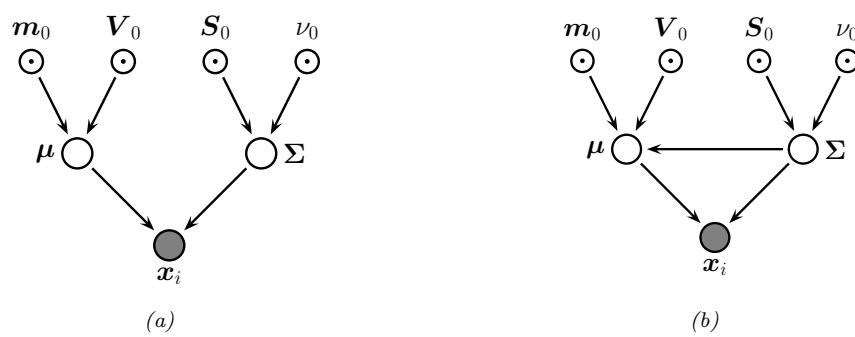


Figure 3.12: Graphical models representing different kinds of assumptions about the parameter priors. (a) A semi-conjugate prior for a Gaussian. (b) A conjugate prior for a Gaussian.

Prior

The obvious prior to use is the following

$$p(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathcal{N}(\boldsymbol{\mu} | \tilde{\boldsymbol{\mu}}, \tilde{\mathbf{C}}) \text{IW}(\boldsymbol{\Sigma} | \tilde{\mathbf{S}}, \tilde{\nu}) \quad (3.122)$$

where IW is the inverse Wishart distribution. Unfortunately, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ 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, $p(\boldsymbol{\mu} | \boldsymbol{\Sigma})$ and $p(\boldsymbol{\Sigma} | \boldsymbol{\mu})$, are individually conjugate. To create a fully conjugate prior, we need to use a prior where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are dependent on each other. We will use a joint distribution of the form $p(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = p(\boldsymbol{\mu} | \boldsymbol{\Sigma})p(\boldsymbol{\Sigma})$. 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:

$$\text{NIW}(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \tilde{\boldsymbol{\mu}}, \tilde{\kappa}, \tilde{\nu}, \tilde{\mathbf{S}}) \triangleq \mathcal{N}(\boldsymbol{\mu} | \tilde{\boldsymbol{\mu}}, \frac{1}{\tilde{\kappa}} \boldsymbol{\Sigma}) \times \text{IW}(\boldsymbol{\Sigma} | \tilde{\mathbf{S}}, \tilde{\nu}) \quad (3.123)$$

$$\begin{aligned} &= \frac{1}{Z_{\text{NIW}}} |\boldsymbol{\Sigma}|^{-\frac{1}{2}} \exp \left(-\frac{\tilde{\kappa}}{2} (\boldsymbol{\mu} - \tilde{\boldsymbol{\mu}})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{\mu} - \tilde{\boldsymbol{\mu}}) \right) \\ &\times |\boldsymbol{\Sigma}|^{-\frac{\tilde{\nu}+D+1}{2}} \exp \left(-\frac{1}{2} \text{tr}(\boldsymbol{\Sigma}^{-1} \tilde{\mathbf{S}}) \right) \end{aligned} \quad (3.124)$$

where the normalization constant is given by

$$Z_{\text{NIW}} \triangleq 2^{\tilde{\nu}D/2} \Gamma_D(\tilde{\nu}/2) (2\pi/\tilde{\kappa})^{D/2} |\tilde{\mathbf{S}}|^{-\tilde{\nu}/2} \quad (3.125)$$

The parameters of the NIW can be interpreted as follows: $\tilde{\boldsymbol{\mu}}$ is our prior mean for $\boldsymbol{\mu}$, and $\tilde{\kappa}$ is how strongly we believe this prior; $\tilde{\mathbf{S}}$ is (proportional to) our prior mean for $\boldsymbol{\Sigma}$, and $\tilde{\nu}$ is how strongly we believe this prior.³

² Using the language of directed graphical models, we see that $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ 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 **Posterior**

2 To derive the posterior, let us define the sum-of-squares matrix

3

$$\mathbf{S}_0 \triangleq \sum_{n=1}^N \mathbf{y}_n \mathbf{y}_n^\top = \mathbf{Y}^\top \mathbf{Y} \quad (3.126)$$

4 so we can rewrite the scatter matrix as follows:

5

$$\mathbf{S}_{\bar{\mathbf{y}}} = \mathbf{S}_0 - \frac{1}{N} \left(\sum_{n=1}^N \mathbf{y}_n \right) \left(\sum_{n=1}^N \mathbf{y}_n \right)^\top = \mathbf{S}_0 - N \bar{\mathbf{y}} \bar{\mathbf{y}}^\top \quad (3.127)$$

6 Now we can multiply the likelihood and the prior to give

7

$$p(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \mathcal{D}) \propto |\boldsymbol{\Sigma}|^{-\frac{N}{2}} \exp \left(-\frac{N}{2} (\boldsymbol{\mu} - \bar{\mathbf{y}})^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{\mu} - \bar{\mathbf{y}}) \right) \exp \left(-\frac{1}{2} \text{tr}(\boldsymbol{\Sigma}^{-1} \mathbf{S}_{\bar{\mathbf{y}}}) \right) \quad (3.128)$$

8

$$\times |\boldsymbol{\Sigma}|^{-\frac{\nu+D+2}{2}} \exp \left(-\frac{\kappa}{2} (\boldsymbol{\mu} - \bar{\mathbf{m}})^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{\mu} - \bar{\mathbf{m}}) \right) \exp \left(-\frac{1}{2} \text{tr}(\boldsymbol{\Sigma}^{-1} \bar{\mathbf{S}}) \right) \quad (3.129)$$

9

$$= |\boldsymbol{\Sigma}|^{-(N+\nu+D+2)/2} \exp(-\frac{1}{2} \text{tr}(\boldsymbol{\Sigma}^{-1} \mathbf{M})) \quad (3.130)$$

10 where

11

$$\mathbf{M} \triangleq N(\boldsymbol{\mu} - \bar{\mathbf{y}})(\boldsymbol{\mu} - \bar{\mathbf{y}})^\top + \kappa (\boldsymbol{\mu} - \bar{\mathbf{m}})(\boldsymbol{\mu} - \bar{\mathbf{m}})^\top + \mathbf{S}_{\bar{\mathbf{y}}} + \bar{\mathbf{S}} \quad (3.131)$$

12

$$= (\kappa + N) \boldsymbol{\mu} \boldsymbol{\mu}^\top - \boldsymbol{\mu} (\kappa \bar{\mathbf{m}} + N \bar{\mathbf{y}})^\top - (\kappa \bar{\mathbf{m}} + N \bar{\mathbf{x}}) \boldsymbol{\mu}^\top + \kappa \bar{\mathbf{m}} \bar{\mathbf{m}}^\top + \mathbf{S}_0 + \bar{\mathbf{S}} \quad (3.132)$$

13 We can simplify the \mathbf{M} matrix using a trick called **completing the square**. Applying this to the
14 above, we have

15

$$(\kappa + N) \boldsymbol{\mu} \boldsymbol{\mu}^\top - \boldsymbol{\mu} (\kappa \bar{\mathbf{m}} + N \bar{\mathbf{y}})^\top - (\kappa \bar{\mathbf{m}} + N \bar{\mathbf{x}}) \boldsymbol{\mu}^\top \quad (3.133)$$

16

$$= (\kappa + N) \left(\boldsymbol{\mu} - \frac{\kappa \bar{\mathbf{m}} + N \bar{\mathbf{y}}}{\kappa + N} \right) \left(\boldsymbol{\mu} - \frac{\kappa \bar{\mathbf{m}} + N \bar{\mathbf{x}}}{\kappa + N} \right)^\top \quad (3.134)$$

17

$$- \frac{(\kappa \bar{\mathbf{m}} + N \bar{\mathbf{x}})(\kappa \bar{\mathbf{m}} + N \bar{\mathbf{y}})^\top}{\kappa + N} \quad (3.135)$$

18

$$= \hat{\kappa} (\boldsymbol{\mu} - \hat{\mathbf{m}})(\boldsymbol{\mu} - \hat{\mathbf{m}})^\top - \hat{\kappa} \hat{\mathbf{m}} \hat{\mathbf{m}}^\top \quad (3.136)$$

19 Hence we can rewrite the posterior as follows:

20

$$p(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \mathcal{D}) \propto |\boldsymbol{\Sigma}|^{(\bar{\nu}+D+1)/2} \exp \left(-\frac{1}{2} \text{tr} [\boldsymbol{\Sigma}^{-1} (\hat{\kappa} (\boldsymbol{\mu} - \hat{\mathbf{m}})(\boldsymbol{\mu} - \hat{\mathbf{m}})^\top + \hat{\mathbf{S}})] \right) \quad (3.137)$$

21

$$= \text{NIW}(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \hat{\mathbf{m}}, \hat{\kappa}, \hat{\nu}, \hat{\mathbf{S}}) \quad (3.138)$$

22 is large, then our uncertainty in $\boldsymbol{\mu}$ must be large too. This makes sense intuitively, since if the data has large spread, it
23 will be hard to pin down its mean.

1 where

$$\underline{3} \quad \hat{\boldsymbol{m}} = \frac{\breve{\kappa} \breve{\boldsymbol{m}} + N \bar{\boldsymbol{y}}}{\hat{\kappa}} = \frac{\breve{\kappa}}{\breve{\kappa} + N} \breve{\boldsymbol{m}} + \frac{N}{\breve{\kappa} + N} \bar{\boldsymbol{y}} \quad (3.139)$$

$$\underline{4} \quad \hat{\kappa} = \breve{\kappa} + N \quad (3.140)$$

$$\underline{5} \quad \hat{\nu} = \breve{\nu} + N \quad (3.141)$$

$$\underline{6} \quad \hat{\mathbf{S}} = \breve{\mathbf{S}} + \mathbf{S}_{\bar{\boldsymbol{y}}} + \frac{\breve{\kappa} N}{\breve{\kappa} + N} (\bar{\boldsymbol{y}} - \breve{\boldsymbol{m}})(\bar{\boldsymbol{y}} - \breve{\boldsymbol{m}})^T \quad (3.142)$$

$$\underline{7} \quad = \breve{\mathbf{S}} + \mathbf{S}_0 + \breve{\kappa} \breve{\boldsymbol{m}} \breve{\boldsymbol{m}}^T - \hat{\kappa} \hat{\boldsymbol{m}} \hat{\boldsymbol{m}}^T \quad (3.143)$$

12 This result is actually quite intuitive: the posterior mean $\hat{\boldsymbol{m}}$ is a convex combination of the prior
13 mean and the MLE; the posterior scatter matrix $\hat{\mathbf{S}}$ is the prior scatter matrix $\breve{\mathbf{S}}$ plus the empirical
14 scatter matrix $\mathbf{S}_{\bar{\boldsymbol{y}}}$ plus an extra term due to the uncertainty in the mean (which creates its own
15 virtual scatter matrix); and the posterior confidence factors $\hat{\kappa}$ and $\hat{\nu}$ are both incremented by the
16 size of the data we condition on.

Posterior marginals

We have computed the joint posterior

$$\underline{21} \quad p(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \mathcal{D}) = \mathcal{N}(\boldsymbol{\mu} | \boldsymbol{\Sigma}, \mathcal{D}) p(\boldsymbol{\Sigma} | \mathcal{D}) = \mathcal{N}(\boldsymbol{\mu} | \hat{\boldsymbol{m}}, \frac{1}{\hat{\kappa}} \boldsymbol{\Sigma}) \text{IW}(\boldsymbol{\Sigma} | \hat{\mathbf{S}}, \hat{\nu}) \quad (3.144)$$

We now discuss how to compute the posterior marginals, $p(\boldsymbol{\Sigma} | \mathcal{D})$ and $p(\boldsymbol{\mu} | \mathcal{D})$.

It is easy to see that the posterior marginal for $\boldsymbol{\Sigma}$ is

$$\underline{26} \quad p(\boldsymbol{\Sigma} | \mathcal{D}) = \int p(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \mathcal{D}) d\boldsymbol{\mu} = \text{IW}(\boldsymbol{\Sigma} | \hat{\mathbf{S}}, \hat{\nu}) \quad (3.145)$$

For the mean, one can show that

$$\underline{30} \quad p(\boldsymbol{\mu} | \mathcal{D}) = \int p(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \mathcal{D}) d\boldsymbol{\Sigma} = \mathcal{T}(\boldsymbol{\mu} | \hat{\boldsymbol{m}}, \frac{\hat{\mathbf{S}}}{\hat{\kappa} \hat{\nu}'}, \hat{\nu}') \quad (3.146)$$

where $\hat{\nu}' \triangleq \hat{\nu} - D + 1$. Intuitively this result follows because $p(\boldsymbol{\mu} | \mathcal{D})$ is an infinite mixture of Gaussians, where each mixture component has a value of $\boldsymbol{\Sigma}$ drawn from the IW distribution; by mixing these altogether, we induce a Student distribution, which has heavier tails than a single Gaussian.

Posterior predictive

Following Section 3.2.2.4, we now discuss how to predict future data by integrating out the parameters. If $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \mathcal{D}) \sim \text{NIW}(\hat{\boldsymbol{m}}, \hat{\kappa}, \hat{\nu}, \hat{\mathbf{S}})$, then one can show that the posterior predictive distribution, for a single observation vector, is as follows:

$$\underline{41} \quad p(\boldsymbol{y} | \mathcal{D}) = \int \mathcal{N}(\boldsymbol{x} | \boldsymbol{\mu}, \boldsymbol{\Sigma}) \text{NIW}(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \hat{\boldsymbol{m}}, \hat{\kappa}, \hat{\nu}, \hat{\mathbf{S}}) d\boldsymbol{\mu} d\boldsymbol{\Sigma} \quad (3.147)$$

$$\underline{44} \quad = \mathcal{T}(\boldsymbol{y} | \hat{\boldsymbol{m}}, \frac{\hat{\mathbf{S}} (\hat{\kappa} + 1)}{\hat{\kappa} \hat{\nu}'}, \hat{\nu}') \quad (3.148)$$

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

$$p(\mathcal{D}|\boldsymbol{\eta}) = h(\mathcal{D}) \exp(\boldsymbol{\eta}^\top \mathbf{s}(\mathcal{D}) - NA(\boldsymbol{\eta})) \quad (3.149)$$

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:

$$p(\boldsymbol{\eta}|\tilde{\boldsymbol{\tau}}, \tilde{\nu}) = \frac{1}{Z(\tilde{\boldsymbol{\tau}}, \tilde{\nu})} \exp(\tilde{\boldsymbol{\tau}}^\top \boldsymbol{\eta} - \tilde{\nu} A(\boldsymbol{\eta})) \quad (3.150)$$

where $\tilde{\nu}$ is the strength of the prior, and $\tilde{\boldsymbol{\tau}} / \tilde{\nu}$ is the prior mean, and $Z(\tilde{\boldsymbol{\tau}}, \tilde{\nu})$ is a normalizing factor. The parameters $\tilde{\boldsymbol{\tau}}$ can be derived from **virtual samples** representing our prior beliefs.

3.2.5.3 Posterior

The posterior is given by

$$p(\boldsymbol{\eta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\eta})p(\boldsymbol{\eta})}{p(\mathcal{D})} \quad (3.151)$$

$$= \frac{h(\mathcal{D})}{Z(\tilde{\boldsymbol{\tau}}, \tilde{\nu})p(\mathcal{D})} \exp((\tilde{\boldsymbol{\tau}} + \mathbf{s}(\mathcal{D}))^\top \boldsymbol{\eta} - (\tilde{\nu} + N)A(\boldsymbol{\eta})) \quad (3.152)$$

$$= \frac{1}{Z(\hat{\boldsymbol{\tau}}, \hat{\nu})} \exp(\hat{\boldsymbol{\tau}}^\top \boldsymbol{\eta} - \hat{\nu} A(\boldsymbol{\eta})) \quad (3.153)$$

where

$$\hat{\boldsymbol{\tau}} = \tilde{\boldsymbol{\tau}} + \mathbf{s}(\mathcal{D}) \quad (3.154)$$

$$\hat{\nu} = \tilde{\nu} + N \quad (3.155)$$

$$Z(\hat{\boldsymbol{\tau}}, \hat{\nu}) = \frac{Z(\tilde{\boldsymbol{\tau}}, \tilde{\nu})}{h(\mathcal{D})} p(\mathcal{D}) \quad (3.156)$$

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

$$p(\mathcal{D}) = \frac{Z(\tilde{\tau}, \tilde{\nu})h(\mathcal{D})}{Z(\check{\tau}, \check{\nu})} \quad (3.157)$$

The posterior mean is given by a convex combination of the prior mean and the empirical mean (which is the MLE):

$$\mathbb{E}[\boldsymbol{\eta}|\mathcal{D}] = \frac{\tilde{\tau}}{\tilde{\nu}} = \frac{\check{\tau} + \mathbf{s}(\mathcal{D})}{\check{\nu} + N} = \frac{\check{\nu}}{\check{\nu} + N} \frac{\check{\tau}}{\check{\nu}} + \frac{N}{\check{\nu} + N} \frac{\mathbf{s}(\mathcal{D})}{N} \quad (3.158)$$

$$= \lambda \mathbb{E}[\boldsymbol{\eta}] + (1 - \lambda) \hat{\boldsymbol{\eta}}_{\text{mle}} \quad (3.159)$$

where $\lambda = \frac{\check{\nu}}{\check{\nu} + N}$.

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 $\mathcal{D} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$:

$$p(\mathcal{D}'|\mathcal{D}) = \int p(\mathcal{D}'|\boldsymbol{\eta})p(\boldsymbol{\eta}|\mathcal{D})d\boldsymbol{\eta} \quad (3.160)$$

$$= \int h(\mathcal{D}') \exp(\boldsymbol{\eta}^T \mathbf{s}(\mathcal{D}') - N' A(\boldsymbol{\eta})) \frac{1}{Z(\tilde{\tau}, \tilde{\nu})} \exp(\boldsymbol{\eta}^T \tilde{\tau} - \tilde{\nu} A(\boldsymbol{\eta})) d\boldsymbol{\eta} \quad (3.161)$$

$$= h(\mathcal{D}') \frac{Z(\check{\tau} + \mathbf{s}(\mathcal{D}) + \mathbf{s}(\mathcal{D}'), \check{\nu} + N + N')}{Z(\check{\tau} + \mathbf{s}(\mathcal{D}), \check{\nu} + N)} \quad (3.162)$$

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

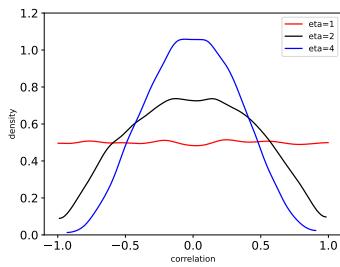
The conjugate prior for a covariance matrix is the inverse Wishart (Section 2.4.4.2). However, it can be hard to set the parameters for this in an uninformative way. One approach, discussed in [HW13], is to use a scale mixture of inverse Wisharts, where the scaling parameters have inverse gamma distributions. 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:

$$\Sigma = \text{diag}(\boldsymbol{\sigma}) \mathbf{R} \text{diag}(\boldsymbol{\sigma}) \quad (3.173)$$

⁵. See comments from Michael Betancourt at <https://github.com/pymc-devs/pymc/issues/538>.

1
2
3
4
5
6
7
8
9
10



11
12
13

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`.

14

15 For example, if $D = 2$, we have

16

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix} \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix} = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix} \quad (3.174)$$

17

We can put a factored prior on the standard deviations, following the recommendations of Section 3.3.2.
For example,

18

$$p(\sigma) = \prod_{d=1}^D \text{Expon}(\sigma_d | 1) \quad (3.175)$$

19

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

20

21

$$\text{LKJ}(\mathbf{R}|\eta) \propto |\mathbf{R}|^{\eta-1} \quad (3.176)$$

22

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.

23

24

3.4 Noninformative priors

25

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**.

26

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

27

28

3.4.1 Maximum entropy priors

29

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

30

⁶ <https://github.com/stan-dev/stan/wiki/Prior-Choice-Recommendations>.

31

32

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].

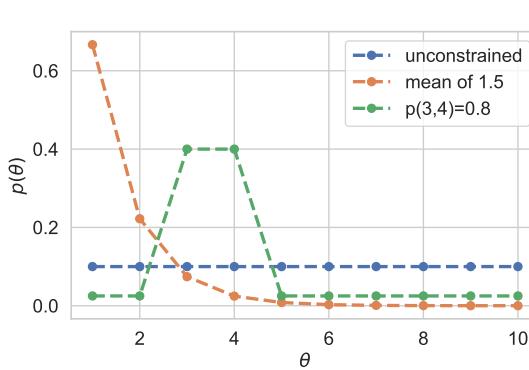


Figure 3.14: Illustration of 3 different maximum entropy priors. Adapted from Figure 1.10 of [MKL11]. Generated by `maxent_priors.py`.

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

$$p(x|\phi) \propto \left(\frac{\phi}{\phi+1} \right)^x \left(1 - \frac{\phi}{\phi+1} \right)^{n-x} = \phi^x (\phi+1)^{-x} (\phi+1)^{-n+x} = \phi^x (\phi+1)^{-x} \quad (3.185)$$

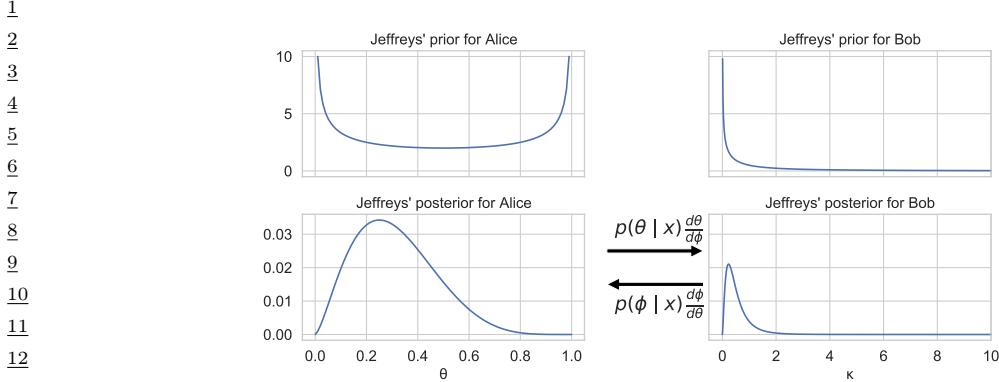


Figure 3.15: Illustration of Jeffrey's prior for Alice (who uses the rate θ) and Bob (who uses the odds $\phi = \theta/(1 - \theta)$). Adapted from Figure 1.9 of [MKL11]. Generated by `jeffreys_prior_binomial.py`.

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**

3 Consider a 1d Gaussian $x \sim \mathcal{N}(\mu, \sigma^2)$ with both parameters unknown, so $\theta = (\mu, \sigma^2)$. From
4 Equation (2.242), the Fisher information matrix is
5

6
$$\mathbf{F}(\theta) = \begin{pmatrix} 1/\sigma^2 & 0 \\ 0 & 2/\sigma^2 \end{pmatrix} \quad (3.193)$$

7

9 Since $\sqrt{\det(\mathbf{F}(\theta))} = \sqrt{\frac{2}{\sigma^4}}$, the Jeffreys prior has the form
10

11 $p(\mu, \sigma^2) \propto 1/\sigma^2 \quad (3.194)$
12

13 It turns out that we can emulate this prior with a conjugate NIX prior:
14

15 $p(\mu, \sigma^2) = NI\chi^2(\mu, \sigma^2 | \mu_0 = 0, \check{\kappa} = 0, \check{\nu} = -1, \check{\sigma}^2 = 0) \quad (3.195)$
16

17 This lets us easily reuse the results for conjugate analysis of the Gaussian in Section 3.2.3.3, as we
18 showed in Section 3.2.3.4.
19

20 **3.4.3 Invariant priors**
21

22 If we have “objective” prior knowledge about a problem in the form of invariances, we may be able to
23 encode this into a prior, as we show below.
24

25 **3.4.3.1 Translation-invariant priors**
26

27 A **location-scale family** is a family of probability distributions parameterized by a location μ and
28 scale σ . If x is an rv in this family, then $y = a + bx$ is also an rv in the same family.

29 When inferring the location parameter μ , it is intuitively reasonable to want to use a **translation-**
30 **invariant prior**, which satisfies the property that the probability mass assigned to any interval,
31 $[A, B]$ is the same as that assigned to any other shifted interval of the same width, such as $[A - c, B - c]$.
32 That is,

33
$$\int_{A-c}^{B-c} p(\mu) d\mu = \int_A^B p(\mu) d\mu \quad (3.196)$$

34

36 This can be achieved using
37

38 $p(\mu) \propto 1 \quad (3.197)$
39

40 since
41

42
$$\int_{A-c}^{B-c} 1 d\mu = (A - c) - (B - c) = (A - B) = \int_A^B 1 d\mu \quad (3.198)$$

43

45 This is the same as the Jeffreys prior for a Gaussian with unknown mean μ and fixed variance.
46 This follows since $F(\mu) = 1/\sigma^2 \propto 1$, from Equation (2.242), and hence $p(\mu) \propto 1$.

3.4.3.2 Scale-invariant prior

When inferring the scale parameter σ , we may want to use a **scale-invariant prior**, which satisfies the property that the probability mass assigned to any interval $[A, B]$ is the same as that assigned to any other interval $[A/c, B/c]$, where $c > 0$. That is,

$$\int_{A/c}^{B/c} p(\sigma) d\sigma = \int_A^B p(\sigma) d\sigma \quad (3.199)$$

This can be achieved by using

$$p(\sigma) \propto 1/\sigma \quad (3.200)$$

since then

$$\int_{A/c}^{B/c} \frac{1}{\sigma} d\sigma = [\log \sigma]_{A/c}^{B/c} = \log(B/c) - \log(A/c) = \log(B) - \log(A) = \int_A^B \frac{1}{\sigma} d\sigma \quad (3.201)$$

This is the same as the Jeffreys prior for a Gaussian with fixed mean μ and unknown scale σ . This follows since $F(\sigma) = 2/\sigma^2$, from Equation (2.242), and hence $p(\sigma) \propto 1/\sigma$.

3.4.3.3 Learning invariant priors

Whenever we have knowledge of some kind of invariance we want our model to satisfy, we can use this to encode a corresponding prior. Sometimes this is done analytically (see e.g., [Rob07, Ch.9]). When this is intractable, it may be possible to learn invariant priors by solving a variational optimization problem (see e.g., [NS18]).

3.4.4 Reference priors

One way to define a noninformative prior is as a distribution which is maximally far from all possible posteriors, when averaged over datasets. This is the basic idea behind a **reference prior** [Ber05; BBS09]. More precisely, we say that $p(\theta)$ is a reference prior if it maximizes the expected KL divergence between posterior and prior:

$$p^*(\theta) = \underset{p(\theta)}{\operatorname{argmax}} \int_{\mathcal{D}} p(\mathcal{D}) D_{\text{KL}}(p(\theta|\mathcal{D}) \| p(\theta)) d\mathcal{D} \quad (3.202)$$

where $p(\mathcal{D}) = \int p(\mathcal{D}|\theta)p(\theta)d\theta$. This is the same as maximizing the mutual information $\mathbb{I}(\theta, \mathcal{D})$.

We can eliminate the integral over datasets by noting that

$$\int p(\mathcal{D}) \int p(\theta|\mathcal{D}) \log \frac{p(\theta|\mathcal{D})}{p(\theta)} = \int p(\theta) \int p(\mathcal{D}|\theta) \log \frac{p(\mathcal{D}|\theta)}{p(\mathcal{D})} = \mathbb{E}_{\theta} [D_{\text{KL}}(p(\mathcal{D}|\theta) \| p(\mathcal{D}))] \quad (3.203)$$

where we used the fact that $\frac{p(\theta|\mathcal{D})}{p(\theta)} = \frac{p(\mathcal{D}|\theta)}{p(\mathcal{D})}$.

One can show that, in 1d, the corresponding prior is equivalent to the Jeffreys prior. In higher dimensions, we can compute the reference prior for one parameter at a time, using the chain rule. However, this can become computationally intractable. See [NS17] for a tractable approximation based on variational inference (Section 10.1).

¹ **3.5 Hierarchical priors**

³ Bayesian models require specifying a prior $p(\boldsymbol{\theta})$ for the parameters. The parameters of the prior are
⁴ called **hyperparameters**, and will be denoted by $\boldsymbol{\phi}$. If these are unknown, we can put a prior on
⁵ them; this defines a **hierarchical Bayesian model**, or **multi-level model**, which can visualize
⁶ 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
⁷ kind of minimally informative prior), so the joint distribution has the form
⁸

⁹
$$p(\boldsymbol{\phi}, \boldsymbol{\theta}, \mathcal{D}) = p(\boldsymbol{\phi})p(\boldsymbol{\theta}|\boldsymbol{\phi})p(\mathcal{D}|\boldsymbol{\theta}) \quad (3.204)$$

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

²⁶
$$p(y_j|\mathcal{D}) = \int p(y_j|\boldsymbol{\theta}_j)p(\boldsymbol{\theta}_j|\mathcal{D})d\boldsymbol{\theta}_j \quad (3.205)$$

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

³²
$$p(y_*|\mathcal{D}) = \int p(y_*|\boldsymbol{\theta}_*)p(\boldsymbol{\theta}_*|\boldsymbol{\phi})p(\boldsymbol{\phi}|\mathcal{D})d\boldsymbol{\theta}_*d\boldsymbol{\phi} \quad (3.206)$$

³⁴ We give some examples below. (More information can be found in e.g., [GH07; Gel+14a].)

³⁶ **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

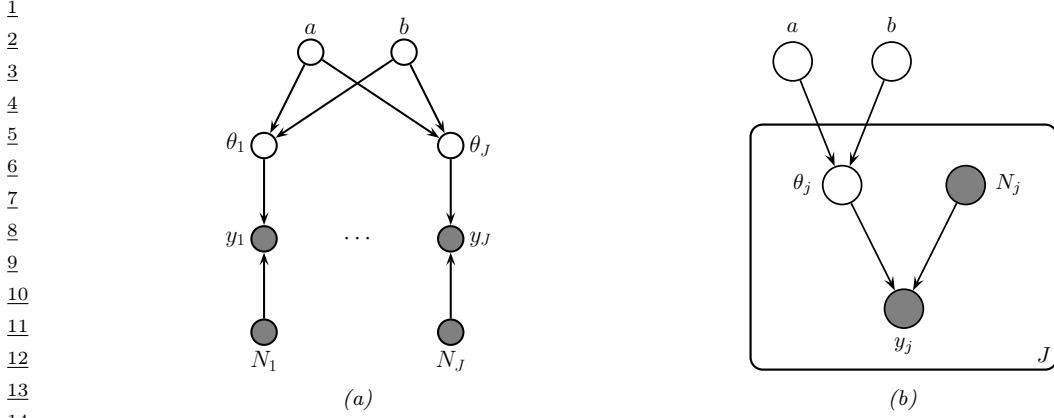


Figure 3.16: PGM for a hierarchical binomial model. (a) “Unrolled” model. (b) Same model, using plate notation.

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;

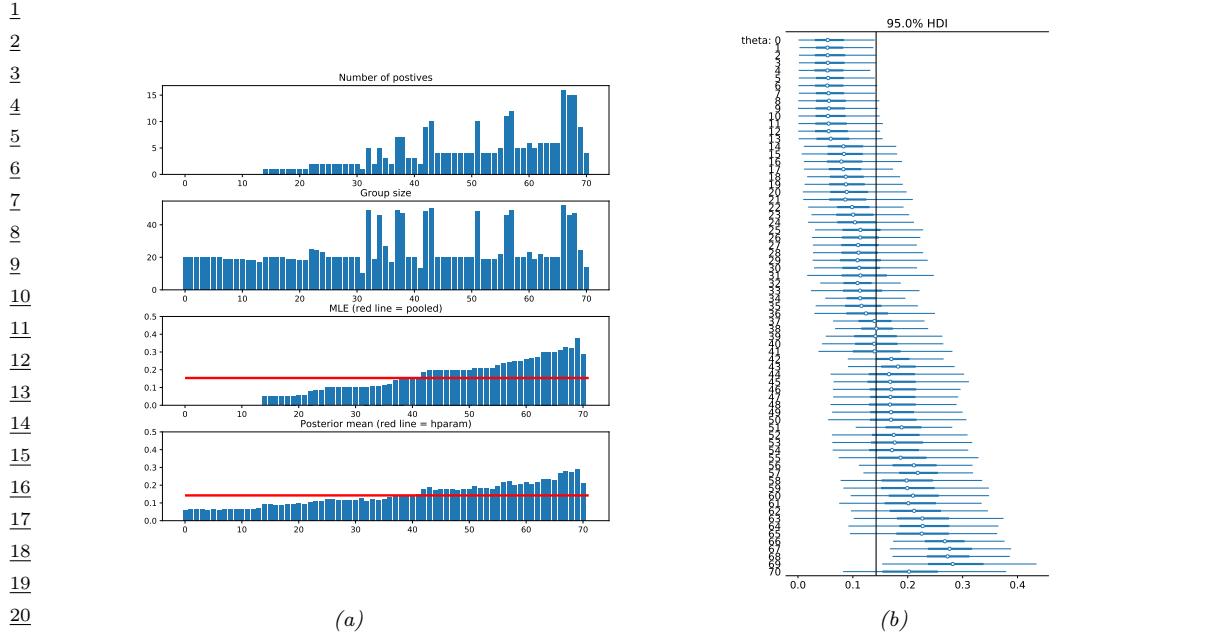


Figure 3.17: Data and inferences for the hierarchical binomial model fit using HMC. Generated by `hbayes_binom_rats_pymc3.ipynb`.

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).

31

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

$$p(\mu, \tau^2, \boldsymbol{\theta}_{1:J} | \mathcal{D}) \propto p(\mu)p(\tau^2) \prod_{j=1}^J \mathcal{N}(\theta_j | \mu, \tau^2) \mathcal{N}(y_j | \theta_j, \sigma_j^2) \quad (3.209)$$

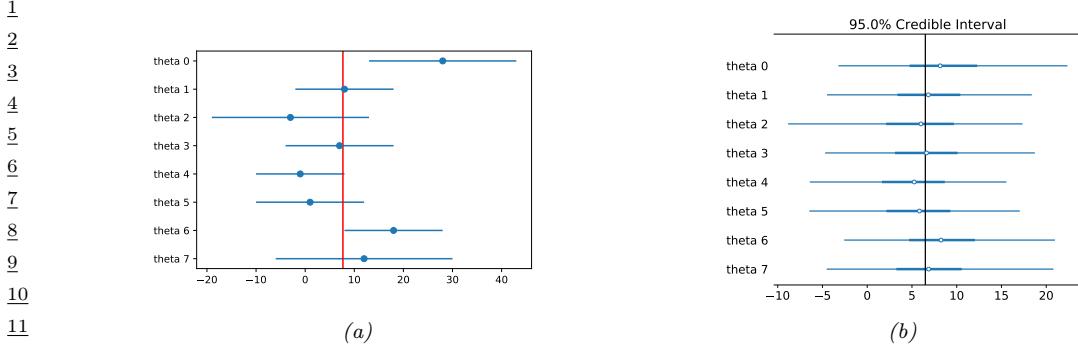


Figure 3.18: 8-schools dataset. (a) Raw data. Each row plots $y_j \pm \sigma_j$. Vertical line is the pooled estimate. (b) Posterior 95% credible intervals for θ_j . Vertical line is posterior mean $\mathbb{E}[\mu|\mathcal{D}]$. Generated by [schools8_pymc3.py](#).

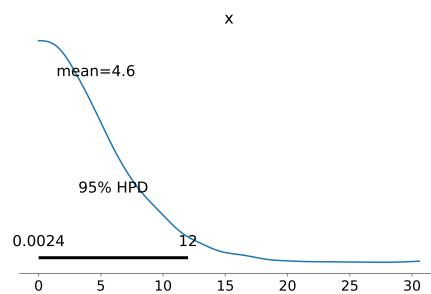


Figure 3.19: Marginal posterior density $p(\tau|\mathcal{D})$ for the 8-schools dataset. Generated by [schools8_pymc3.py](#).

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.)

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

$$\underline{9} \quad p(\mu, \tau^2|\mathcal{D}) = \delta(\mu - \bar{\mu})\delta(\tau^2 - \bar{\tau}^2) \quad (3.211)$$

¹⁰ then we can use the results from Section 3.2.3.1 to compute the marginal posteriors
¹¹

$$\underline{12} \quad p(\theta_j|\mathcal{D}) \approx p(\theta_j|\mathcal{D}_j, \bar{\mu}, \bar{\tau}^2) \quad (3.212)$$

¹⁴ 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
¹⁵ global mean $\bar{\mu} = \mathbb{E}[\mu|\mathcal{D}]$:

$$\underline{17} \quad \mathbb{E}[\theta_j|\mathcal{D}, \bar{\mu}, \bar{\tau}^2] = w_j \bar{\mu} + (1 - w_j) \hat{\theta}_j \quad (3.213)$$

¹⁹ where the amount of shrinkage towards the global mean is given by
²⁰

$$\underline{21} \quad w_j = \frac{\sigma_j^2}{\sigma_j^2 + \tau^2} \quad (3.214)$$

²⁴ Thus we see that there is more shrinkage for groups with smaller measurement precision (e.g., due to
²⁵ smaller sample size), which makes intuitive sense. There is also more shrinkage if τ^2 is smaller; of
²⁶ course τ^2 is unknown, but we can compute a posterior for it, as shown in Figure 3.19.

²⁷

²⁸ 3.5.2.2 Non-centered parameterization

²⁹

³⁰ It turns out that posterior inference in this model is difficult for many algorithms because of the
³¹ tight dependence between the variance hyper parameter τ^2 and the group means θ_j , as illustrated
³² by the **funnel shape** in Figure 3.20. In particular, consider making local move through parameter
³³ space. The algorithm can only “visit” the place where τ^2 is small (corresponding to strong shrinkage
³⁴ 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
³⁵ τ^2 is small unless all groups *simultaneously* move their θ_j estimates closer to μ .

³⁶ A standard solution to this problem is to rewrite the model using the following **non-centered**
³⁷ **parameterization**:

$$\underline{39} \quad \theta_j = \mu + \tau \eta_j \quad (3.215)$$

$$\underline{40} \quad \eta_j \sim \mathcal{N}(0, 1) \quad (3.216)$$

⁴¹ See Figure 3.21b for the corresponding graphical model. By writing θ_j as a deterministic function of
⁴² its parents plus a local noise term, we have reduced the dependence between θ_j and τ and hence the
⁴³ other θ_k variables, which can improve the computational efficiency of inference algorithms. as we
⁴⁴ discuss in Section 12.6.4. This kind of reparameterization is widely used in hierarchical Bayesian
⁴⁵ models. See Section 12.6.4 for more details.

⁴⁶

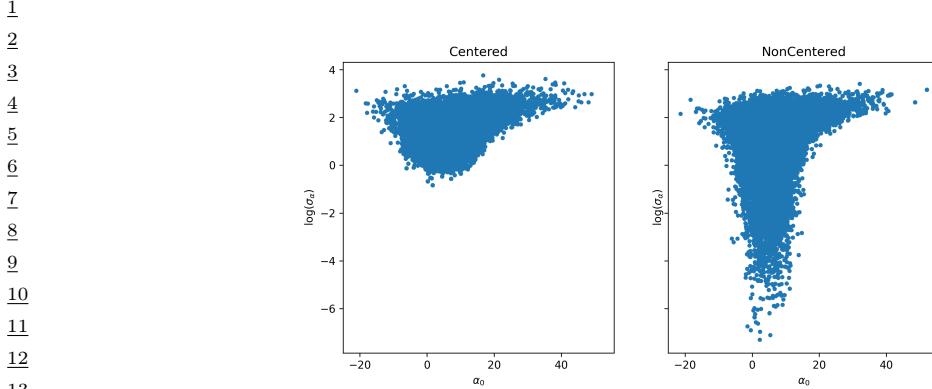


Figure 3.20: Posterior $p(\mu_0, \log(\tau) | \mathcal{D})$ for the 8 schools model using (a) centered parameterization and (b) non-centered parameterization. Generated by `schools8_pymc3.ipynb`.

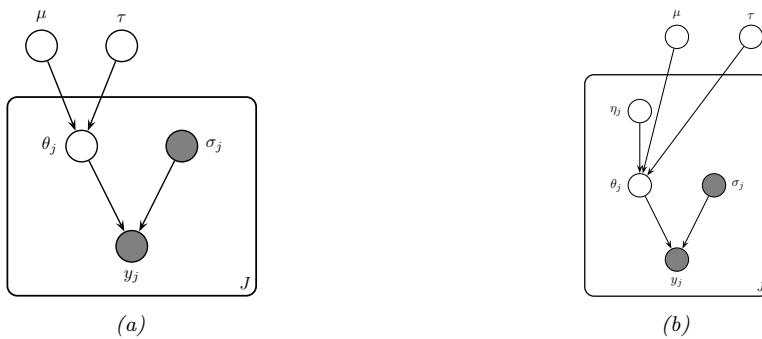


Figure 3.21: A hierarchical Gaussian Bayesian model. (a) Centered parameterization. (b) Non-centered parameterization.

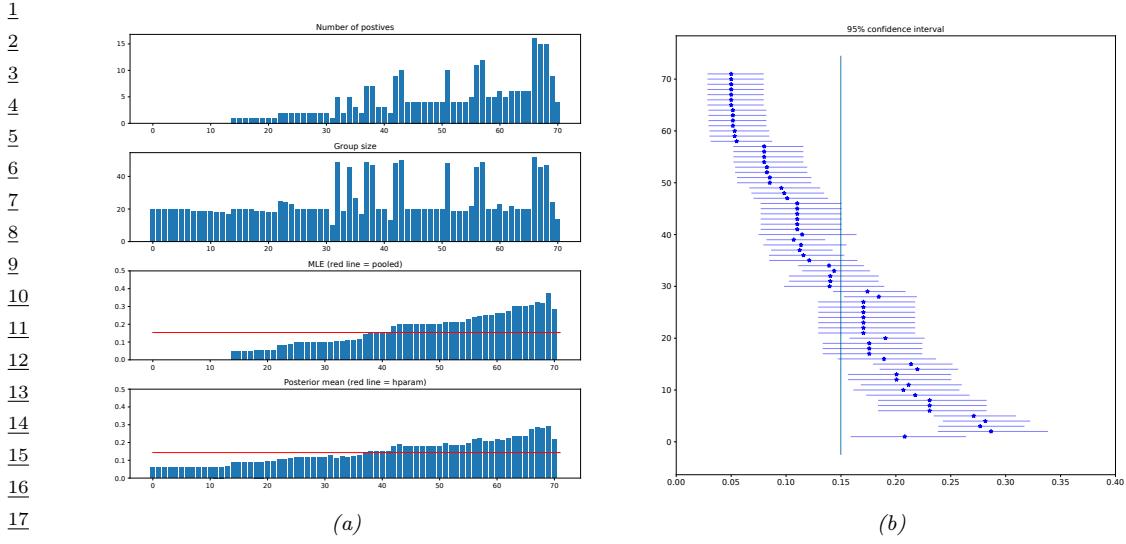
3.6 Empirical Bayes

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:

$$\hat{\phi}_{\text{mmle}}(\mathcal{D}) = \underset{\phi}{\operatorname{argmax}} p(\mathcal{D}|\phi) = \underset{\phi}{\operatorname{argmax}} \int p(\mathcal{D}|\theta)p(\theta|\phi)d\theta \quad (3.217)$$

This technique is known as **type II maximum likelihood**, since we are optimizing the hyper-parameters, 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-



19 *Figure 3.22: Data and inferences for the hierarchical binomial model fit using empirical Bayes. Generated by*
 20 *ebBinom.py.*

23 parameters.

24 Since we are estimating the prior parameters from data, this approach is **empirical Bayes (EB)**
 25 [CL96]. This violates the principle that the prior should be chosen independently of the data.
 26 However, we can view it as a computationally cheap approximation to inference in the full hierarchical
 27 Bayesian model, just as we viewed MAP estimation as an approximation to inference in the one level
 28 model $\theta \rightarrow \mathcal{D}$. In fact, we can construct a hierarchy in which the more integrals one performs, the
 29 “more Bayesian” one becomes, as shown below.

Method	Definition
Maximum likelihood	$\hat{\theta} = \operatorname{argmax}_{\theta} p(\mathcal{D} \theta)$
MAP estimation	$\hat{\theta} = \operatorname{argmax}_{\theta} p(\mathcal{D} \theta)p(\theta \phi)$
ML-II (Empirical Bayes)	$\hat{\phi} = \operatorname{argmax}_{\phi} \int p(\mathcal{D} \theta)p(\theta \phi)d\theta$
MAP-II	$\hat{\phi} = \operatorname{argmax}_{\phi} \int p(\mathcal{D} \theta)p(\theta \phi)p(\phi)d\theta$
Full Bayes	$p(\theta, \phi \mathcal{D}) \propto p(\mathcal{D} \theta)p(\theta \phi)p(\phi)$

39 Note that ML-II is less likely to overfit than “regular” maximum likelihood, because there are
 40 typically fewer hyper-parameters ϕ than there are parameters θ . We give some simple examples
 41 below, and will see some ML applications later in the book.

42

43 3.6.1 A hierarchical binomial model

45 In this section, we revisit the hierarchical binomial model from Section 3.5.1, but we use empirical
 46 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 [Min00c].

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:

$$\hat{\tau}^2 = \max\{0, v - \sigma^2\} = (v - \sigma^2)_+ \quad (3.225)$$

1 Given this, the posterior mean becomes
2

$$\hat{\theta}_j = \lambda\mu + (1 - \lambda)y_j = \mu + (1 - \lambda)(y_j - \mu) \quad (3.226)$$

3 where $\lambda_j = \lambda = \sigma^2 / (\sigma^2 + \tau^2)$.
4

5 Unfortunately, we cannot use the above method on the 8-schools dataset in Section 3.5.2.1, since it
6 uses unequal σ_j . However, we can still use the EM algorithm or other optimization based methods.
7

8 3.6.3 Hierarchical Bayes for n-gram smoothing

9 The main problem with add-one smoothing, discussed in Section 2.8.3.3, is that it assumes that
10 all n-grams are equally likely, which is not very realistic. A more sophisticated approach, called
11 **deleted interpolation** [CG96], defines the transition matrix as a convex combination of the bigram
12 frequencies $f_{jk} = N_{jk}/N_j$ and the unigram frequencies $f_k = N_k/N$:
13

$$\hat{A}_{jk} = (1 - \lambda)f_{jk} + \lambda f_k = (1 - \lambda)\frac{N_{jk}}{N_j} + \lambda\frac{N_k}{N} \quad (3.227)$$

14 The term λ is usually set by cross validation. There is also a closely related technique called **backoff**
15 **smoothing**; the idea is that if f_{jk} is too small, we “back off” to a more reliable estimate, namely f_k .
16

17 We now show that this heuristic can be interpreted as an empirical Bayes approximation to a
18 hierarchical Bayesian model for the parameter vectors corresponding to each row of the transition
19 matrix \mathbf{A} . Our presentation follows [MP95].
20

21 First, let us use an independent Dirichlet prior on each row of the transition matrix:
22

$$\mathbf{A}_j \sim \text{Dir}(\alpha_0 m_1, \dots, \alpha_0 m_K) = \text{Dir}(\alpha_0 \mathbf{m}) = \text{Dir}(\boldsymbol{\alpha}) \quad (3.228)$$

23 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
24 prior strength (see Figure 3.23). In terms of the earlier notation, we have $\boldsymbol{\theta}_j = \mathbf{A}_j$ and $\phi = (\alpha, \mathbf{m})$.
25 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
26 records the number of times we have transitioned out of state j to each of the other states. From
27 Equation (3.61), the posterior predictive density is
28

$$p(X_{t+1} = k | X_t = j, \mathcal{D}) = \frac{N_{jk} + \alpha_j m_k}{N_j + \alpha_0} = \frac{f_{jk} N_j + \alpha_j m_k}{N_j + \alpha_0} \quad (3.229)$$

$$= (1 - \lambda_j)f_{jk} + \lambda_j m_k \quad (3.230)$$

29 where
30

$$\lambda_j = \frac{\alpha_j}{N_j + \alpha_0} \quad (3.231)$$

31 This is very similar to Equation (3.227) but not identical. The main difference is that the Bayesian
32 model uses a context-dependent weight λ_j to combine m_k with the empirical frequency f_{jk} , rather
33 than a fixed weight λ . This is like *adaptive* deleted interpolation. Furthermore, rather than backing
34 off to the empirical marginal frequencies f_k , we back off to the model parameter m_k .
35

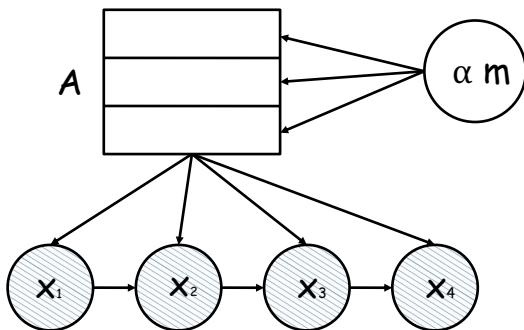


Figure 3.23: A Markov chain in which we put a different Dirichlet prior on every row of the transition matrix \mathbf{A} , but the hyperparameters of the Dirichlet are shared.

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 is the posterior over models. This is called **Bayesian model selection**. If the prior over models is
 28 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

35 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

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>.

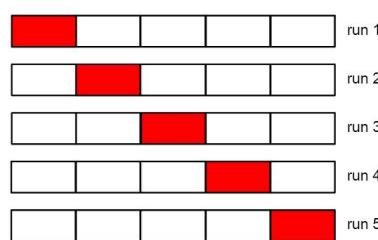


Figure 3.24: Schematic of 5-fold cross validation.

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:

$$p(\mathbf{y} | \mathbf{x}_n, \mathcal{D}_{1:n-1}, m) \approx \int p(\mathbf{y} | \mathbf{x}_n, \boldsymbol{\theta}) \delta(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_m(\mathcal{D}_{1:n-1})) d\boldsymbol{\theta} = p(\mathbf{y} | \mathbf{x}_n, \hat{\boldsymbol{\theta}}_m(\mathcal{D}_{1:n-1})) \quad (3.246)$$

1 Then Equation (3.244) simplifies to

$$\log p(\mathcal{D}|m) \approx \sum_{n=1}^N \log p(\mathbf{y}_n|\mathbf{x}_n, \hat{\boldsymbol{\theta}}(\mathcal{D}_{1:n-1}), m) \quad (3.247)$$

7 This is very similar to Equation (3.243), except it is evaluated sequentially. A complex model will
8 overfit the “early” examples and will then predict the remaining ones poorly, and thus will get low
9 marginal likelihood as well as low cross-validation score. See [FH20] for further discussion.

11 3.7.4 Pareto-Smoothed Importance Sampling LOO estimate

13 Suppose we have computed the posterior given the full dataset, $p(\boldsymbol{\theta}|\mathcal{D}, m)$. We can use this to evaluate
14 the resulting predictive distribution $p(\mathbf{y}|\mathcal{D}, m)$ on each datapoint. This gives the **log-pointwise**
15 **predictive-density** or **LPPD** score:

$$\text{LPPD} \triangleq \sum_{n=1}^N \log p(\mathbf{y}_n|\mathbf{x}_n, \mathcal{D}, m) = \sum_{n=1}^N \log \int p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}, m) p(\boldsymbol{\theta}|\mathcal{D}, m) d\boldsymbol{\theta} \quad (3.248)$$

20 We can approximate LPPD with Monte Carlo:

$$\text{LPPD}(m) \approx \sum_{n=1}^N \log \left(\frac{1}{S} \sum_{s=1}^S p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}_s, m) \right) \quad (3.249)$$

25 where $\boldsymbol{\theta}_s \sim p(\boldsymbol{\theta}|\mathcal{D}, m)$ is a posterior sample.

27 The trouble with LPPD is that it predicts the n 'th data point \mathbf{y}_n using all the data, including \mathbf{y}_n .
28 What we would like to compute is the **expected LPPD (ELPD)** on *future data*, \mathbf{y}_* :

$$\text{ELPD} \triangleq \mathbb{E}_{\mathbf{y}_*} \log p(\mathbf{y}_*|\mathbf{x}_*, \mathcal{D}, m) = \int p(\mathbf{y}_*|\mathbf{x}_*, \boldsymbol{\theta}, m) \log \int p(\mathbf{y}_*|\mathbf{x}_*, \boldsymbol{\theta}, m) p(\boldsymbol{\theta}|\mathcal{D}, m) d\boldsymbol{\theta} \quad (3.250)$$

32 Of course, the future data is unknown, but we can use a LOO approximation:

$$\text{ELPD}_{\text{LOO}} \triangleq \sum_{n=1}^N \log p(\mathbf{y}_n|\mathbf{x}_n, \mathcal{D}_{-n}, m) = \sum_{n=1}^N \log \int p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}, m) p(\boldsymbol{\theta}|\mathcal{D}_{-n}, m) d\boldsymbol{\theta} \quad (3.251)$$

37 This is a Bayesian version of Equation (3.243). We can approximate this integral using Monte Carlo:

$$\text{ELPD}_{\text{LOO}} \approx \sum_{n=1}^N \log \left(\frac{1}{S} \sum_{s=1}^S p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}_{s,-n}, m) \right) \quad (3.252)$$

43 where $\boldsymbol{\theta}_{s,-n} \sim p(\boldsymbol{\theta}|\mathcal{D}_{-n}, m)$.

44 The above procedure requires computing N different posteriors, leaving one data point out at a
45 time, which is slow. A faster alternative is to compute $p(\boldsymbol{\theta}|\mathcal{D}, m)$ once, and then use importance
46 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 An alternative approach to cross validation is to score models using the negative log likelihood (or
³¹ LPPD) on the training set plus a **complexity penalty** term:

$$\frac{34}{35} \mathcal{L}(m) = -\log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, m) + C(m) \quad (3.258)$$

³⁶ This is called an **information criterion**. Different methods use different complexity terms $C(m)$,
³⁷ as we discuss below. Note also that it is conventional, when working with information criteria, to
³⁸ 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 We can think about the problem of scoring different models in terms of information theory (Chapter 5).
⁴⁵ The goal is for the sender to communicate the data to the receiver. First the sender needs to specify
⁴⁶

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

$$\mathcal{L}_{\text{AIC}}(m) = -2 \log p(\mathcal{D} | \hat{\theta}, m) + 2D_m \quad (3.268)$$

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
³⁰

$$\mathcal{L}_{\text{WAIC}}(m) = -2\text{LPPD}(m) + 2C(m) \quad (3.270)$$

³³ Interestingly, it can be shown that the PSIS LOO estimate in Section 3.7.4 is asymptotically equivalent to WAIC [VGG17].

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.

From a Bayesian perspective, this can seem a bit odd, since if we knew there was a better model, why don't we just use that? Here we assume that we do not have a specific alternative model in mind (so we are not performing model selection, unlike Section 3.7.1) Instead we are just trying to see if the data we observe is "typical" of what we might expect if our model were correct. This is called **model checking**.

⁴⁵ In particular, suppose we knew the true parameters θ , which we use to generate S synthetic
⁴⁶ datasets, $\tilde{\mathcal{D}}^s = \{\mathbf{y}_n^s \sim p(\cdot|\theta) : n = 1 : N\}$; these represent “plausible hallucinations” of the model. To
⁴⁷

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^*).

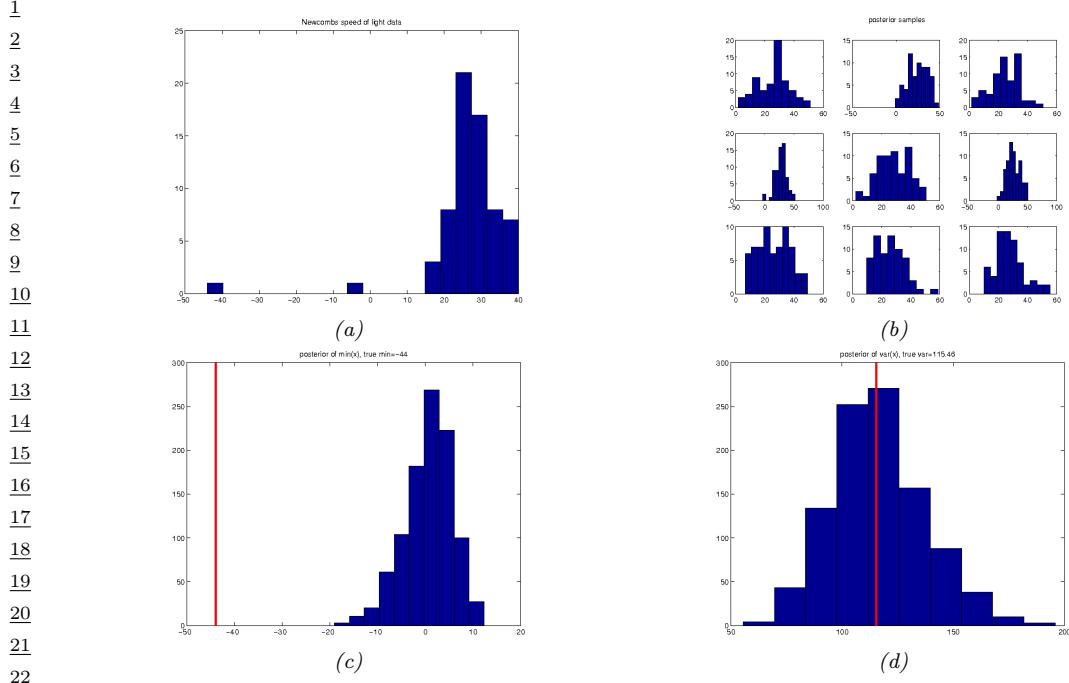


Figure 3.25: (a) Histogram of Newcomb's data. (b) Histograms of data sampled from Gaussian model. (c) Histogram of test statistic on data sampled from the model, which represents $p(\text{test}(\tilde{\mathcal{D}}^s)|\mathcal{D})$, where $\text{test}(\mathcal{D}) = \min\{y \in \mathcal{D}\}$. The vertical line is the test statistic on the true data, $\text{test}(\mathcal{D})$. (d) Same as (c) except $\text{test}(\mathcal{D}) = \mathbb{V}\{y \in \mathcal{D}\}$. Generated by `newcomb_plugin_demo.py`.

27

28

29

30 We can approximate the Bayesian p-value using Monte Carlo integration, as follows:

31

$$32 \quad p_B = \int \mathbb{I}(\text{test}(\tilde{\mathcal{D}}) > \text{test}(\mathcal{D})) p(\tilde{\mathcal{D}}|\theta) p(\theta|\mathcal{D}) d\theta \approx \frac{1}{S} \sum_{s=1}^S \mathbb{I}(\text{test}(\tilde{\mathcal{D}}^s) > \text{test}(\mathcal{D})) \quad (3.275)$$

33

34 Any extreme value for p_B (i.e., a value near 0 or 1) means that the observed data is unlikely under
 35 the model, as assessed via test statistic test. However, if $\text{test}(\mathcal{D})$ is a sufficient statistic of the model,
 36 it is likely to be well estimated, and the p-value will be near 0.5. For example, in the speed of light
 37 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
 38 a p-value of 0.48. (See Figure 3.25(d).) This shows that the Gaussian model is capable of representing
 39 the variance in the data, even though it is not capable of representing the support (range) of the
 40 data.

41 The above example illustrates the very important point that we should not try to assess whether
 42 the data comes from a given model (for which the answer is nearly always that it does not), but
 43 rather, we should just try to assess whether the model captures the features we care about. See
 44 [Gel+04, ch.6] for a more extensive discussion of this topic.

45

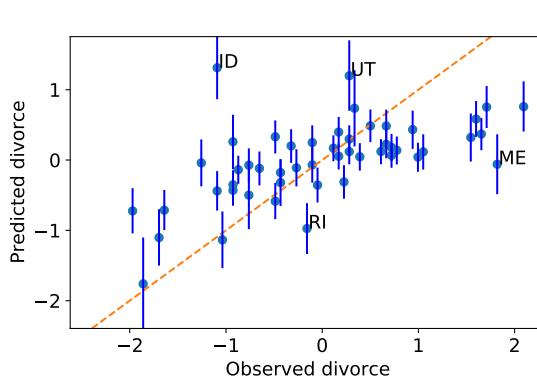


Figure 3.26: Posterior predictive distribution for divorce rate vs actual divorce rate for 50 US states. Both axes are standardized (i.e., z-scores). A few outliers are annotated. Adapted from Figure 5.5 of [McE20]. Generated by `linreg_divorce_ppc_numpyro.py`.

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/3iMTZgV>.

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
0	0	0	0.01	0.84	8.00	0
0	1	0	0.01	0.14	8.00	0
1	0	1	0.80	47.73	8.00	1
1	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:

$$\pi(\mathbf{x}) = \operatorname{argmax}_{y \in \mathcal{Y}} p(y|\mathbf{x}) \quad (3.282)$$

20 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.

25 **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

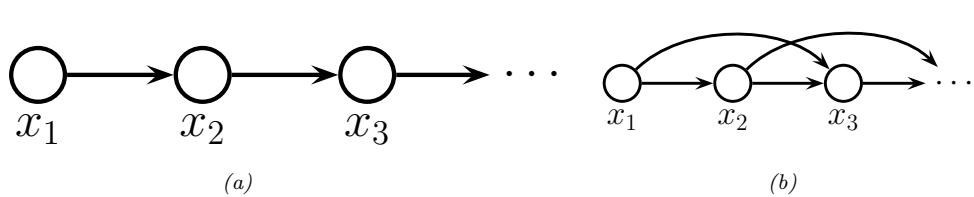


Figure 4.1: Illustration of first and second order Markov models.

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, x_t . (We formalize this in Section 4.2.3.) This means that the 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

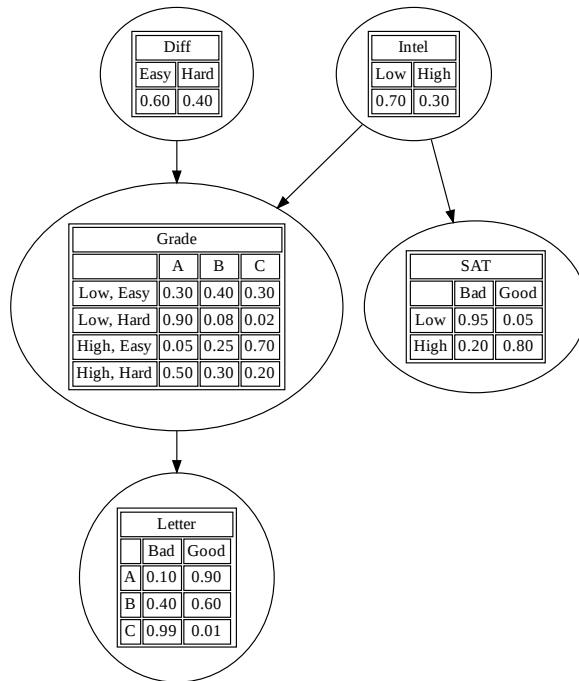


Figure 4.2: The (simplified) student network. “Diff” is the difficulty of the class. “Intel” is the intelligence of the student. “Grade” is the grade of the student in this class. “SAT” is the score of the student on the SAT exam. “Letter” is whether the teacher writes a good or bad letter of recommendation. 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, $p(D = \text{Hard}|G = C) = 0.63$). 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

2 4.2.2.3 Gaussian Bayes nets

3

4 Consider a PGM-D where all the variables are real-valued, and all the CPDs have the following form,
5 known as a **linear Gaussian CPD**:

6

$$7 p(x_i | \mathbf{x}_{\text{pa}(i)}) = \mathcal{N}(x_i | \mu_i + \mathbf{w}_i^\top \mathbf{x}_{\text{pa}(i)}, \sigma_i^2) \quad (4.8)$$

8

9 As we show below, multiplying all these CPDs together results in a large joint Gaussian distribution of
10 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**
11 or a **Gaussian Bayes net**.

12 We now explain how to derive $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, following [SK89, App. B]. For convenience, we will rewrite
13 the CPDs in the following form:

14

$$15 x_i = \mu_i + \sum_{j \in \text{pa}(i)} w_{i,j}(x_j - \mu_j) + \sigma_i z_i \quad (4.9)$$

16

17 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
18 of the $j \rightarrow i$ edge, and μ_i is the local mean.¹

19 It is easy to see that the global mean is just the concatenation of the local means, $\boldsymbol{\mu} = (\mu_1, \dots, \mu_V)$.
20 We now derive the global covariance, $\boldsymbol{\Sigma}$. Let $\mathbf{S} \triangleq \text{diag}(\boldsymbol{\sigma})$ be a diagonal matrix containing the
21 standard deviations. We can rewrite Equation (4.9) in matrix-vector form as follows:

22

$$23 (\mathbf{x} - \boldsymbol{\mu}) = \mathbf{W}(\mathbf{x} - \boldsymbol{\mu}) + \mathbf{S}\mathbf{z} \quad (4.10)$$

24

25 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
26 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
27 the topological ordering), we have that $\mathbf{I} - \mathbf{W}$ is lower triangular with 1s on the diagonal. Hence

28

$$29 \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_V \end{pmatrix} = \begin{pmatrix} 1 & & & & \\ -w_{2,1} & 1 & & & \\ -w_{3,2} & -w_{3,1} & 1 & & \\ \vdots & & & \ddots & \\ -w_{V,1} & -w_{V,2} & \dots & -w_{V,V-1} & 1 \end{pmatrix} \begin{pmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \\ \vdots \\ x_V - \mu_V \end{pmatrix} \quad (4.11)$$

30

31 Since $\mathbf{I} - \mathbf{W}$ is always invertible, we can write

32

$$33 \mathbf{x} - \boldsymbol{\mu} = (\mathbf{I} - \mathbf{W})^{-1} \mathbf{e} \triangleq \mathbf{U}\mathbf{e} = \mathbf{U}\mathbf{S}\mathbf{z} \quad (4.12)$$

34

35 where we defined $\mathbf{U} = (\mathbf{I} - \mathbf{W})^{-1}$. Hence the covariance is given by

36

$$37 \boldsymbol{\Sigma} = \text{Cov}[\mathbf{x}] = \text{Cov}[\mathbf{x} - \boldsymbol{\mu}] \quad (4.13)$$

38

39

$$40 = \text{Cov}[\mathbf{U}\mathbf{S}\mathbf{z}] = \mathbf{U}\mathbf{S} \text{Cov}[\mathbf{z}] \mathbf{S}\mathbf{U}^\top = \mathbf{U}\mathbf{S}^2\mathbf{U}^\top \quad (4.14)$$

41

42 since $\text{Cov}[\mathbf{z}] = \mathbf{I}$.

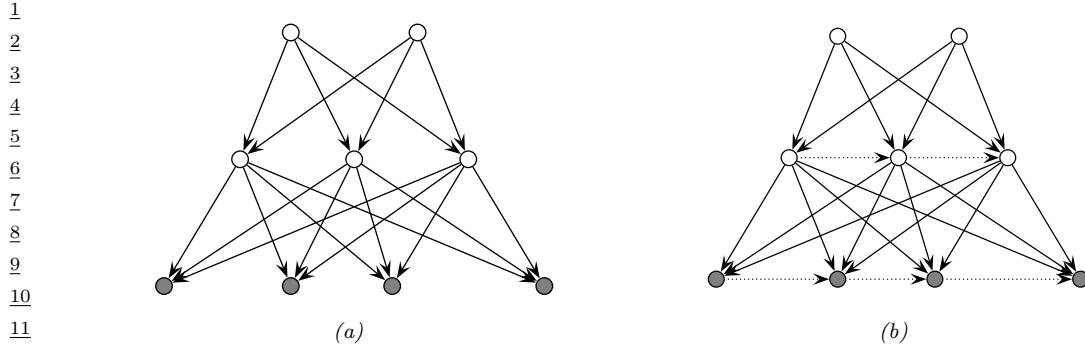


Figure 4.3: (a) Hierarchical latent variable model with 2 layers. (b) Same as (a) but with autoregressive connections within each layer. The observed \mathbf{x} variables are the shaded leaf nodes at the bottom. The unshaded nodes are the hidden \mathbf{z} variables.

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.

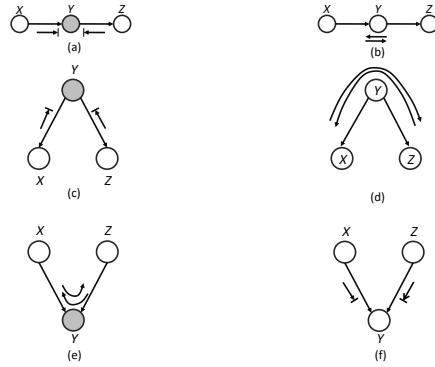
We can justify the 3 rules of Bayes ball as follows. First consider a chain structure $X \rightarrow Y \rightarrow Z$, which encodes

$$p(x, y, z) = p(x)p(y|x)p(z|y) \quad (4.19)$$

When we condition on y , are x and z independent? We have

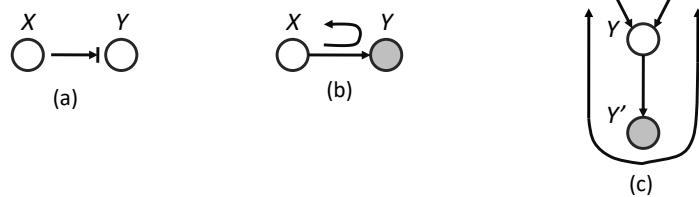
$$p(x, z|y) = \frac{p(x)p(y|x)p(z|y)}{p(y)} = \frac{p(x, y)p(z|y)}{p(y)} = p(x|y)p(z|y) \quad (4.20)$$

1
2
3
4
5
6
7
8
9
10
11
12
13



14 *Figure 4.4: Bayes ball rules. A shaded node is one we condition on. If there is an arrow hitting a bar, it*
15 *means the ball cannot pass through; otherwise the ball can pass through.*

16
17
18
19
20
21
22
23
24
25
26



27 *Figure 4.5: (a-b) Bayes ball boundary conditions. (c) Example of why we need boundary conditions. y' is an*
28 *observed child of y , rendering y “effectively observed”, so the ball bounces back up on its way from x to z .*

29
30

31 and therefore $X \perp Z | Y$. So observing the middle node of chain breaks it in two (as in a Markov
32 chain).

33 Now consider the tent structure $X \leftarrow Y \rightarrow Z$. The joint is

$$35 \quad p(x, y, z) = p(y)p(x|y)p(z|y) \quad (4.21)$$

37 When we condition on y , are x and z independent? We have

$$39 \quad p(x, z|y) = \frac{p(x, y, z)}{p(y)} = \frac{p(y)p(x|y)p(z|y)}{p(y)} = p(x|y)p(z|y) \quad (4.22)$$

41 and therefore $X \perp Z | Y$. So observing a root node separates its children (as in a naive Bayes
42 classifier: see Section 4.2.7.2).

44 Finally consider a v-structure $X \rightarrow Y \leftarrow Z$. The joint is

$$45 \quad p(x, y, z) = p(x)p(z)p(y|x, z) \quad (4.23)$$

	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.

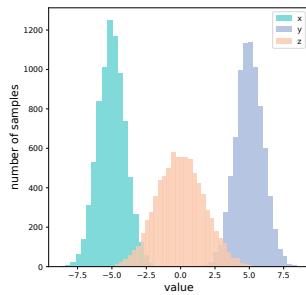
As an example of the CI statements encoded by a DAG, Table 4.1 shows some properties that follow from the student network in Figure 4.2.

4.2.3.2 Explaining away (Berkson's paradox)

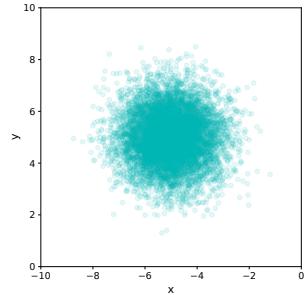
In this section, we give some examples of the **explaining away** phenomenon, also called **Berkson's paradox**.

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

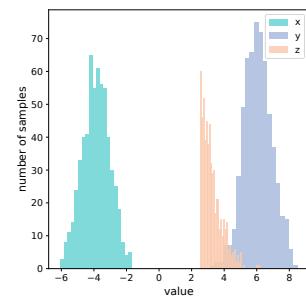
1
2
3
4
5
6
7
8
9
10
11
12
13



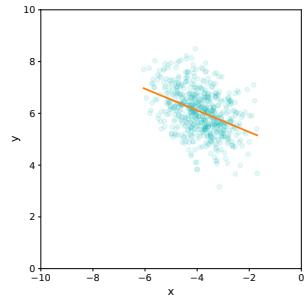
(a)



(b)



(c)



(d)

24

25
26 *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`.*

27

28

29 to record about 75 entries. You will see that every time coin 1 is recorded as heads, coin 2 will be
30 recorded as tails. If we ignore the way in which the data was collected, we might infer from the fact
31 that that coins 1 and 2 are correlated that there is a hidden common cause. However, the correct
32 explanation is that the correlation is due to conditioning on a hidden common effect (namely the
33 decision of whether to record the outcome or not, so we can censor tail-tail events). This is called
34 **selection bias**.

35

36 As another example of this, consider a Gaussian DGM of the form

37

$$38 \quad p(x, y, z) = \mathcal{N}(x| -5, 1)\mathcal{N}(y|5, 1)\mathcal{N}(z|x + y, 1) \quad (4.26)$$

39

40 The graph structure is $X \rightarrow Z \leftarrow Y$, where Z is the child node. Some samples from the unconditional
41 joint distribution $p(x, y, z)$ are shown in Figure 4.6(a); we see that X and Y are uncorrelated. Now
42 suppose we only select samples where $z > 2.5$. Some samples from the conditional joint distribution
43 $p(x, y|z > 2.5)$ are shown in Figure 4.6(b); we see that now X and Y are correlated. This could cause
44 us to erroneously conclude that there is a causal relationship, but in fact the dependency is caused
45 by selection bias.

46

4.2.3.3 Other Markov properties

From the d-separation criterion, one can conclude that

$$t \perp \text{nd}(t) \setminus \text{pa}(t) | \text{pa}(t) \quad (4.27)$$

where the **non-descendants** of a node $nd(t)$ are all the nodes except for its descendants, $nd(t) = \{1, \dots, D\} \setminus \{t \cup \text{desc}(t)\}$. Equation (4.27) is called the (directed) **local Markov property**. For example, in Figure 4.20(a), we have $nd(3) = \{1, 2, 4\}$, and $\text{pa}(3) = 1$, so $3 \perp 2, 4 | 1$.

A special case of this property is when we only look at predecessors of a node according to some topological ordering. We have

$$t \perp \text{pred}(t) \setminus \text{pa}(t) | \text{pa}(t) \quad (4.28)$$

which follows since $\text{pred}(t) \subseteq \text{nd}(t)$. This is called the **ordered Markov property**, which justifies Equation (4.2). For example, in Figure 4.20(a), if we use the ordering $1, 2, \dots, 7$, we find $\text{pred}(3) = \{1, 2\}$ and $\text{pa}(3) = 1$, so $3 \perp 2 | 1$.

We have now described three Markov properties for DAGs: the directed global Markov property G in Equation (4.18), the directed local Markov property L in Equation (4.27), and the ordered Markov property O in Equation (4.28). It is obvious that $G \implies L \implies O$. What is less obvious, but nevertheless true, is that $O \implies L \implies G$ (see e.g., [KF09a] for the proof). Hence all these properties are equivalent.

Furthermore, any distribution p that is Markov wrt G can be factorized as in Equation (4.2); this is called the **factorization property** F. It is obvious that $O \implies F$, but one can show that the converse also holds (see e.g., [KF09a] for the proof).

4.2.3.4 Markov blankets and full conditionals

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 $\text{mb}(t)$. One can show that the Markov blanket of a node in a DGM is equal to the parents, the children, and the **co-parents**, i.e., other nodes who are also parents of its children:

$$\text{mb}(t) \triangleq \text{ch}(t) \cup \text{pa}(t) \cup \text{copa}(t) \quad (4.29)$$

For example, in Figure 4.20(a), we have

$$\text{mb}(5) = \{6, 7\} \cup \{2, 3\} \cup \{4\} = \{2, 3, 4, 6, 7\} \quad (4.30)$$

where 4 is a co-parent of 5 because they share a common child, namely 7.

To see why the co-parents are in the Markov blanket, note that when we derive $p(x_t|x_{-t}) = p(x_t, \mathbf{x}_{-t})/p(\mathbf{x}_{-t})$, all the terms that do not involve x_t will cancel out between numerator and denominator, so we are left with a product of CPDs which contain x_t in their **scope**. Hence

$$p(x_t | \mathbf{x}_{-t}) \propto p(x_t | \mathbf{x}_{\text{pa}(t)}) \prod_{s \in \text{ch}(t)} p(x_s | \mathbf{x}_{\text{pa}(s)}) \quad (4.31)$$

The resulting expression is called *t*'s **full conditional**. For example, in Figure 4.20(a) we have

$$p(x_5|x_{-5}) \propto p(x_5|x_2, x_3)p(x_5|x_2, x_5)p(x_7|x_4, x_5, x_6) \quad (4.32)$$

1 **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 .

16 **4.2.4 Generation (sampling)**

18 It is easy to generate prior samples from a PGM-D: we simply visit the nodes in **topological order**,
19 parents before children, and then sample a value for each node given the value of its parents. This
20 will generate independent samples from the joint, $(x_1, \dots, x_V) \sim p(\mathbf{x}|\boldsymbol{\theta})$. This is called **ancestral**
21 **sampling**.

22 **4.2.5 Inference**

25 In the context of PGMs, the term “**inference**” refers to the task of computing the posterior over a
26 set of **query nodes** Q given the observed values for a set of **visible nodes** V , while marginalizing
27 over the irrelevant **nuisance variables**, $R = \{1, \dots, V\} \setminus \{Q, V\}$:

$$\frac{29}{30} \quad p_{\boldsymbol{\theta}}(Q|V) = \frac{p_{\boldsymbol{\theta}}(Q, V)}{p_{\boldsymbol{\theta}}(V)} = \frac{\sum_R p_{\boldsymbol{\theta}}(Q, V, R)}{p_{\boldsymbol{\theta}}(V)} \quad (4.33)$$

32 (If the variables are continuous, we should replace sums with integrals.) If Q is a single node, then
33 $p_{\boldsymbol{\theta}}(Q|V)$ is called the **posterior marginal** for node Q .

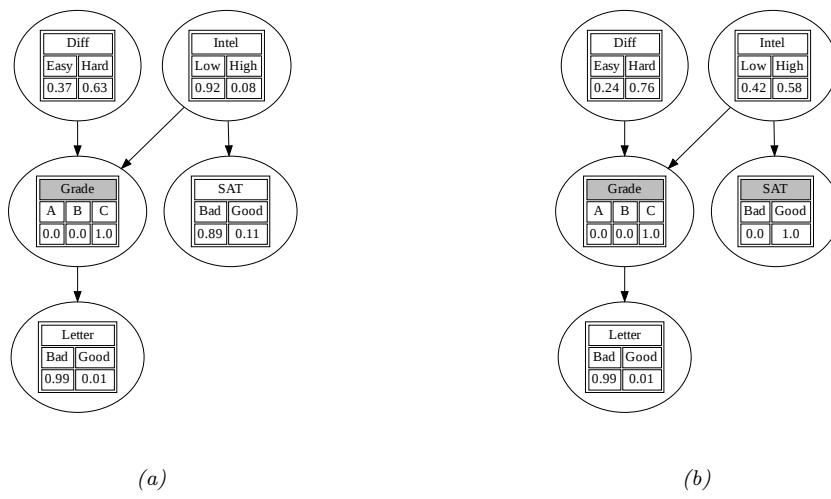
34 As an example, suppose $V = \mathbf{x}$ is a sequence of observed sound waves, $Q = \mathbf{z}$ is the corresponding
35 set of unknown spoken words, and $R = \mathbf{r}$ are random “non-semantic” factors associated with the
36 signal, such as prosody or background noise. Our goal is to compute the posterior over the words
37 given the sounds, while being invariant to the irrelevant factors:

$$\frac{39}{40} \quad p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = \sum_{\mathbf{r}} p_{\boldsymbol{\theta}}(\mathbf{z}, \mathbf{r}|\mathbf{x}) = \sum_{\mathbf{r}} \frac{p_{\boldsymbol{\theta}}(\mathbf{z}, \mathbf{r}, \mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{x})} = \sum_{\mathbf{r}} \frac{p_{\boldsymbol{\theta}}(\mathbf{z}, \mathbf{r}, \mathbf{x})}{\sum_{\mathbf{z}', \mathbf{r}'} p_{\boldsymbol{\theta}}(\mathbf{z}', \mathbf{r}', \mathbf{x})} \quad (4.34)$$

42 As a simplification, we can “lump” the random factors R into the query set Q to define the complete
43 set of **hidden variables** $H = Q \cup R$. In this case, the tasks simplifies to

$$\frac{45}{46} \quad p_{\boldsymbol{\theta}}(\mathbf{h}|\mathbf{x}) = p_{\boldsymbol{\theta}}(\mathbf{h}|\mathbf{x}) = \frac{p_{\boldsymbol{\theta}}(\mathbf{h}, \mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{x})} = \frac{p_{\boldsymbol{\theta}}(\mathbf{h}, \mathbf{x})}{\sum_{\mathbf{h}'} p_{\boldsymbol{\theta}}(\mathbf{h}', \mathbf{x})} \quad (4.35)$$

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16



19 *Figure 4.7: Illustration of belief updating in the “Student” PGM. The histograms show the marginal distribution*
20 *of each node. Nodes with shaded titles are clamped to an observed value. (a) Posterior after conditioning on*
21 *Grade=C. (b) Posterior after also conditioning on SAT=Good. Generated by [student_pgmpy.ipynb](#).*

22
23
24
25
26

27 The computational complexity of the inference task depends on the CI properties of the graph, as
28 we discuss in Chapter 9. In general it is NP-hard (see Section 9.4.3), but for certain graph structures
29 (such as chains, trees and other sparse graphs), it can be solved efficiently (in polynomial) time
30 using dynamic programming (see Chapter 9). For cases where it is intractable, we can use standard
31 methods for approximate Bayesian inference, which we review in Chapter 7.

32
33

34 4.2.5.1 Example: inference in the Student network

35
36
37
38
39
40
41
42
43
44
45
46
47

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 $p(D = \text{Hard}|G = C) = 0.63$), but is more likely explained by the student having low intelligence (since $p(I = \text{High}|G = C) = 0.08$).

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 $p(I = \text{High}|G = C, \text{SAT} = \text{Good}) = 0.58$, 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 $p(D = \text{Hard}|G = C) = 0.76$. (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.)

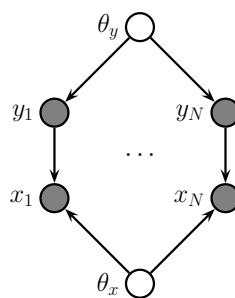


Figure 4.8: A PGM-D representing the joint distribution $p(\mathbf{y}_{1:N}, \mathbf{x}_{1:N}, \boldsymbol{\theta}_y, \boldsymbol{\theta}_x)$. Here $\boldsymbol{\theta}_x$ and $\boldsymbol{\theta}_y$ are global parameter nodes that are shared across the examples, whereas \mathbf{x}_n and y_n are local variables.

4.2.6 Learning

So far, we have assumed that the structure G and parameters $\boldsymbol{\theta}$ 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 $p(\boldsymbol{\theta}|\mathcal{D}, G)$. (Henceforth we will drop the conditioning on G , since we assume the graph structure is fixed.)

We can compute the parameter posterior $p(\boldsymbol{\theta}|\mathcal{D})$ by treating $\boldsymbol{\theta}$ 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, $\hat{\boldsymbol{\theta}} = \text{argmax } p(\boldsymbol{\theta}|\mathcal{D})$. 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 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 y_n to \mathbf{x}_n ; if we are fitting a discriminative classifier, the edge would be from \mathbf{x}_n to y_n , and there would be no $\boldsymbol{\theta}_y$ 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:

$$p(\boldsymbol{\theta}, \mathcal{D}) = p(\boldsymbol{\theta}_x)p(\boldsymbol{\theta}_y) \left[\prod_{n=1}^N p(y_n|\boldsymbol{\theta}_y)p(\mathbf{x}_n|y_n, \boldsymbol{\theta}_x) \right] \quad (4.36)$$

$$= \left[p(\boldsymbol{\theta}_y) \prod_{n=1}^N p(y_n|\boldsymbol{\theta}_y) \right] \left[p(\boldsymbol{\theta}_x) \prod_{n=1}^N p(\mathbf{x}_n|y_n, \boldsymbol{\theta}_x) \right] \quad (4.37)$$

$$= [p(\boldsymbol{\theta}_y)p(\mathcal{D}_y|\boldsymbol{\theta}_y)] [p(\boldsymbol{\theta}_x)p(\mathcal{D}_x|\boldsymbol{\theta}_x)] \quad (4.38)$$

where $\mathcal{D}_y = \{y_n\}_{n=1}^N$ is the data that is sufficient for estimating $\boldsymbol{\theta}_y$ and $\mathcal{D}_x = \{\mathbf{x}_n, y_n\}_{n=1}^N$ is the

1 data that is sufficient for $\boldsymbol{\theta}_x$.

3 From Equation (4.38), we see that the prior, likelihood and posterior all **decompose** or factorize
4 according to the graph structure. Thus we can compute the posterior for each parameter independently.
5 In general, we have

$$\underline{6} \quad p(\boldsymbol{\theta}, \mathcal{D}) = \prod_{i=1}^V p(\boldsymbol{\theta}_i) p(\mathcal{D}_i | \boldsymbol{\theta}_i) \quad (4.39)$$

10 Hence the likelihood and prior factorizes, and thus so does the posterior. If we just want to compute
11 the MLE, we can compute

$$\underline{12} \quad \hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{i=1}^V p(\mathcal{D}_i | \boldsymbol{\theta}_i) \quad (4.40)$$

16 We can solve this for each node independently, as we illustrate in Section 4.2.6.2.

18 4.2.6.2 Example: computing the MLE for CPTs

19 In this section, we illustrate how to compute the MLE for tabular CPDs. The likelihood is given by
20 the following:

$$\underline{22} \quad p(\mathcal{D} | \boldsymbol{\theta}) = \prod_{n=1}^N \prod_{i=1}^V p(x_{ni} | \mathbf{x}_{n,\text{pa}(i)}, \boldsymbol{\theta}_i) \quad (4.41)$$

$$\underline{25} \quad = \prod_{n=1}^N \prod_{i=1}^V \prod_{j=1}^{J_i} \prod_{k=1}^{K_i} \theta_{ijk}^{\mathbb{I}(x_{ni}=k, \mathbf{x}_{n,\text{pa}(i)}=j)} \quad (4.42)$$

28 where

$$\underline{30} \quad \theta_{ijk} \triangleq p(x_i = k | \mathbf{x}_{\text{pa}(i)} = j) \quad (4.43)$$

32 Let us define the sufficient statistics for node i to be N_{ijk} , which is the number of times that node i
33 is in state k while its parents are in joint state j :

$$\underline{36} \quad N_{ijk} \triangleq \sum_{n=1}^N \mathbb{I}(x_{n,i} = k, x_{n,\text{pa}(i)} = j) \quad (4.44)$$

38 The MLE for a multinomial is given by the normalized empirical frequencies:

$$\underline{40} \quad \hat{\theta}_{ijk} = \frac{N_{ijk}}{\sum_{k'} N_{ijk'}} \quad (4.45)$$

43 For example, consider the student network from Section 4.2.2.2. In Table 4.2, we show some sample
44 training data. For example, the last line in the tabel encodes a student who is smart ($I = 1$), who
45 takes a hard class ($D = 1$), gets a C ($G = 2$), but who does well on the SAT ($S = 1$) and gets a good
46 letter of recommendation ($L = 1$).

	I	D	G	S	L
0	0	2	0	0	
1	0	1	2	0	0
2	0	0	1	1	1
3	1	1	1	1	0
4	1	0	0	1	1
5	0	0	0	0	1
6	1	1	2	1	1
7					
8					
9					

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{2}{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]	[\frac{0}{5}, \frac{1}{5}, \frac{1}{5}]	[\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 network, where we use a uniform Dirichlet prior, $\alpha_{ijk} = 1$.

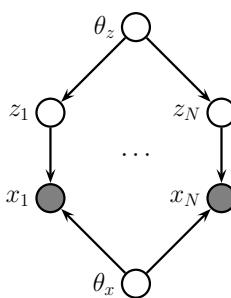


Figure 4.9: A PGM-D representing the joint distribution $p(\mathbf{z}_{1:N}, \mathbf{x}_{1:N}, \boldsymbol{\theta}_z, \boldsymbol{\theta}_x)$. The local variables \mathbf{z}_n are hidden, whereas \mathbf{x}_n are observed. This is typical for learning unsupervised latent variable models.

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 the sections 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.)

but the basic idea is to alternate between inferring the latent variables \mathbf{z}_n (the E or expectation step), and estimating the parameters given this completed dataset (the M or maximization step). 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 sufficient statistics (ESS), which takes much less space. In the M step, we maximize the expected value of the log likelihood of the fully observed data using these ESS.

As example, suppose all the CPDs are tabular, as in the example in Section 4.2.6.2. The log-likelihood of the complete data is given by

$$\log p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{i=1}^V \sum_{j=1}^{J_i} \sum_{k=1}^{K_i} N_{ijk} \log \theta_{ijk} \quad (4.50)$$

and hence the expected complete data log-likelihood has the form

$$\mathbb{E} [\log p(\mathcal{D}|\boldsymbol{\theta})] = \sum_i \sum_j \sum_k \bar{N}_{ijk} \log \theta_{ijk} \quad (4.51)$$

where

$$\bar{N}_{ijk} = \sum_{n=1}^N \mathbb{E} [\mathbb{I}(x_{ni} = k, \mathbf{x}_{n,\text{pa}(i)} = j)] = \sum_{n=1}^N p(x_{ni} = k, \mathbf{x}_{n,\text{pa}(i)} = j | \mathcal{D}_n, \boldsymbol{\theta}^{old}) \quad (4.52)$$

where \mathcal{D}_n are all the visible variables in case n , and $\boldsymbol{\theta}^{old}$ are the parameters from the previous iteration. The quantity $p(x_{ni}, \mathbf{x}_{n,\text{pa}(i)} | \mathcal{D}_n, \boldsymbol{\theta}^{old})$ is known as a **family marginal**, and can be computed using any GM inference algorithm. The \bar{N}_{ijk} are the **expected sufficient statistics** (ESS), and constitute the output of the E step.

Given these ESS, the M step has the simple form

$$\hat{\theta}_{ijk} = \frac{\bar{N}_{ijk}}{\sum_{k'} \bar{N}_{ijk'}} \quad (4.53)$$

We can modify this to perform MAP estimation with a Dirichlet prior by simply adding pseudo counts to the expected counts.

The famous Baum-Welch algorithm (Section 30.4.1) is a special case of the above equations which arises when the PGM-D is an HMM.

34

4.2.6.6 Using SGD to fit CPTs in the incomplete data case

The EM algorithm is a batch algorithm. To scale up to large datasets, it is more common to use stochastic gradient descent or SGD (see e.g., [BC94; Bin+97]). To apply this, we need to compute the marginal likelihood of the observed data for each example:

$$p(\mathbf{x}_n|\boldsymbol{\theta}) = \sum_{\mathbf{z}_n} p(\mathbf{z}_n|\boldsymbol{\theta}_z) p(\mathbf{x}_n|\mathbf{z}_n, \boldsymbol{\theta}_x) \quad (4.54)$$

(We say that we have “collapsed” the model by marginalizing out \mathbf{z}_n .) We can then compute the log likelihood using

$$LL(\boldsymbol{\theta}) = \sum_n \log p(\mathbf{x}_n|\boldsymbol{\theta}) \quad (4.55)$$

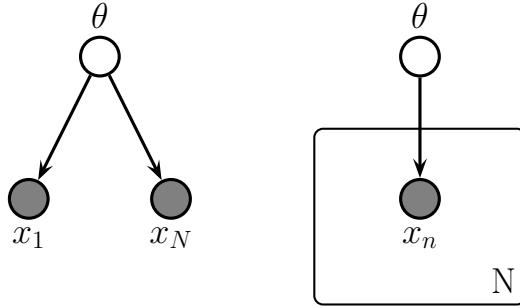


Figure 4.10: Left: data points \mathbf{x}_n are conditionally independent given θ . Right: Same model, using plate notation. This represents the same model as the one on the left, except the repeated \mathbf{x}_n nodes are inside a box, known as a plate; the number in the lower right hand corner, N , 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

$$p(\mathcal{D}, \theta) = p(\theta)p(\mathcal{D} | \theta) \quad (4.61)$$

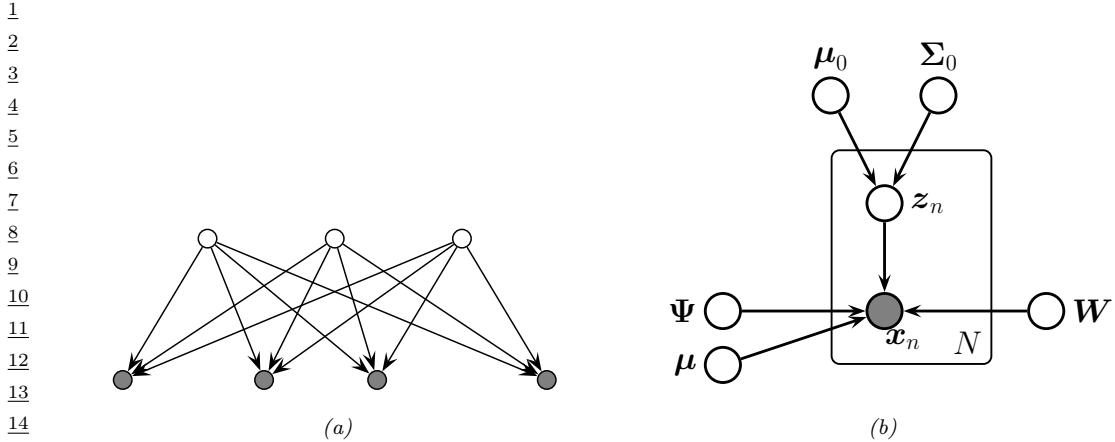


Figure 4.11: (a) Factor analysis model illustrated as a PGM-D. We show the components of z (top row) and x (bottom row) as individual scalar nodes. (b) Equivalent model, where z and x are collapsed to vector-valued nodes, and parameters are added, using plate notation.

where $p(\theta)$ is the prior distribution for the parameters, and $p(\mathcal{D}|\theta)$ is the likelihood. By virtue of the iid assumption, the likelihood can be rewritten as follows:

$$p(\mathcal{D}|\theta) = \prod_{n=1}^N p(x_n|\theta) \quad (4.62)$$

Notice that the order of the data vectors is not important for defining this model, i.e., we can permute the leaves of the PGM-D. When this property holds, we say that the data is **exchangeable**.

In Figure 4.10(a), we see that the x nodes are repeated N times. (The **shaded nodes** represent observed values, whereas the unshaded (hollow) nodes represent latent variables or parameters.) To avoid visual clutter, it is common to use a form of **syntactic sugar** called **plates**. This is a notational convention in which we draw a little box around the repeated variables, with the understanding that nodes within the box will get repeated when the model is **unrolled**. We often write the number of copies or repetitions in the bottom right corner of the box. This is illustrated in Figure 4.10(b).

34

4.2.7.1 Example: factor analysis

36 In Section 29.3.1, we introduced the factor analysis model, which has the form

$$p(z) = \mathcal{N}(z|\mu_0, \Sigma_0) \quad (4.63)$$

$$p(x|z) = \mathcal{N}(x|Wz + \mu, \Psi) \quad (4.64)$$

41 where W is a $D \times L$ matrix, known as the factor loading matrix, and Ψ is a diagonal $D \times D$ covariance matrix.

43 Note that z and x are both vectors. We can explicitly represent their components as scalar nodes 44 as in Figure 4.11a. Here the directed edges correspond to non-zero entries in the W matrix.

45 We can also explicitly show the parameters of the model, using plate notation, as shown in 46 Figure 4.11b.

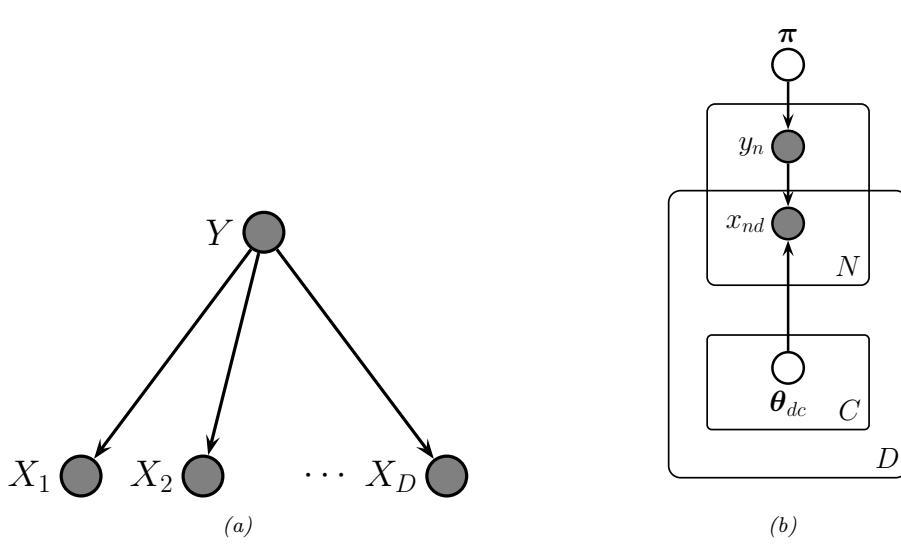


Figure 4.12: (a) Naive Bayes classifier as a PGM-D. (b) Model augmented with plate notation.

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:

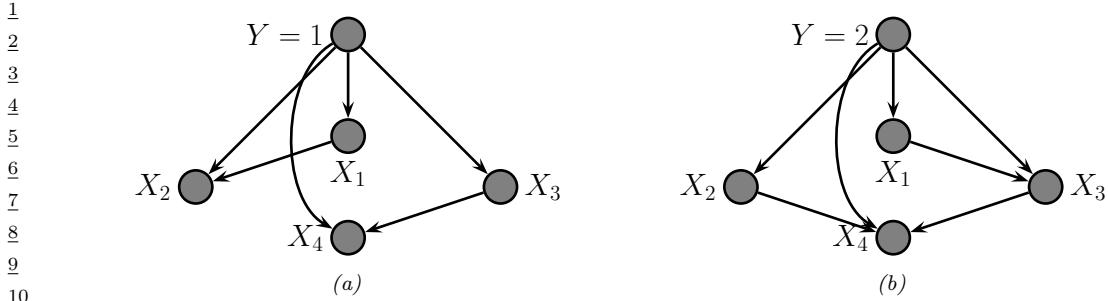
$$p(\mathbf{x}, y|\boldsymbol{\theta}) = p(y|\boldsymbol{\pi}) \prod_{d=1}^D p(x_d|y, \boldsymbol{\theta}_d) \quad (4.65)$$

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 \mathbf{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 y , as illustrated.*

13

14

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.
 17

18

19 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.
 26

27 An alternative is to use an **undirected graphical model (UGM)**, also called a **Markov random**
 28 **field (MRF)** or **Markov network**. These do not require us to specify edge orientations, and are
 29 much more natural for some problems such as image analysis and spatial statistics. For example, an
 30 undirected 2d lattice is shown in Figure 4.14(b); now the Markov blanket of each node is just its
 31 nearest neighbors, as we show in Section 4.3.3.

32 Roughly speaking, the main advantages of PGM-U's over PGM-D's are: (1) they are symmetric and
 33 therefore more "natural" for certain domains, such as spatial or relational data; and (2) discriminative
 34 PGM-U's (aka conditional random fields, or CRFs), which define conditional densities of the form
 35 $p(\mathbf{y}|\mathbf{x})$, work better than discriminative DGMs, for reasons we explain in Section 19.2.1.1. The main
 36 disadvantages of PGM-U's compared to PGM-D's are: (1) the parameters are less interpretable and
 37 less modular, for reasons we explain in Section 4.3.1; and (2) it is more computationally expensive to
 38 estimate the parameters, for reasons we explain in Section 4.3.6.1.

39

40 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.
 47

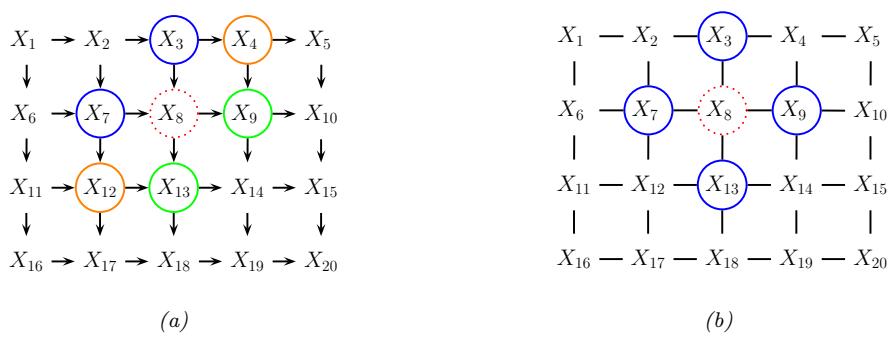


Figure 4.14: (a) A 2d lattice represented as a DAG. The dotted red node X_8 is independent of all other nodes (black) given its Markov blanket, which include its parents (blue), children (green) and co-parents (orange). (b) The same model represented as a PGM-U. The red node X_8 is independent of the 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 **4.3.1.2 Gibbs distribution**

2 The distribution in Equation (4.66) can be rewritten as follows:

3

$$\underline{p}(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp(-\mathcal{E}(\mathbf{x}; \boldsymbol{\theta})) \quad (4.68)$$

4

5 where $\mathcal{E}(\mathbf{x}) > 0$ is the **energy** of state \mathbf{x} , defined by

6

7

$$\underline{\mathcal{E}}(\mathbf{x}; \boldsymbol{\theta}) = \sum_c \mathcal{E}(\mathbf{x}_c; \boldsymbol{\theta}_c) \quad (4.69)$$

8

9 where \mathbf{x}_c are the variables in clique c . We can see the equivalence by defining the clique potentials as

10

11

$$\psi_c(\mathbf{x}_c; \boldsymbol{\theta}_c) = \exp(-\mathcal{E}(\mathbf{x}_c; \boldsymbol{\theta}_c)) \quad (4.70)$$

12

13 We see that low energy is associated with high probability states.

14 Equation (4.68) is known as the **Gibbs distribution**. This kind of probability model is also called
15 an **energy based model**. These are commonly used in physics and biochemistry. They are also be
16 used in ML to define generative models, as we discuss in Chapter 25. (See also Section 19.2, where
17 we discuss **conditional random fields (CRFs)**, which are models of the form $p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})$, where the
18 potential functions are conditioned on input features \mathbf{x} .)

19

20 **4.3.2 Examples**

21 In this section, we give various examples of models that are conveniently represented as MRFs.

22

23 **4.3.2.1 Ising models**

24 Consider the 2d lattice in Figure 4.14(b). We can represent the joint distribution as follows:

25

26

$$\underline{p}(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{i \sim j} \psi_{ij}(x_i, x_j; \boldsymbol{\theta}) \quad (4.71)$$

27

28 where $i \sim j$ means i and j are neighbors in the graph. This is called a **2d lattice model**.

29 An **Ising model** is a special case of the above, where the variables x_i s are binary. Such models
30 are often used to represent magnetic materials. In particular, each node represents an atom, which
31 can have a magnetic dipole, or **spin**, which is in one of two states, +1 and -1. In some magnetic
32 systems, neighboring spins like to be similar; in other systems, they like to be dissimilar. We can
33 capture this interaction by defining the clique potentials as follows:

34

35

$$\underline{\psi}_{ij}(x_i, x_j; \boldsymbol{\theta}) = \begin{cases} e^{J_{ij}} & \text{if } x_i = x_j \\ e^{-J_{ij}} & \text{if } x_i \neq x_j \end{cases} \quad (4.72)$$

36

37 where J_{ij} is the coupling strength between nodes i and j . This is known as the **Ising model**. If
38 two nodes are not connected in the graph, we set $J_{ij} = 0$. We assume that the weight matrix is
39

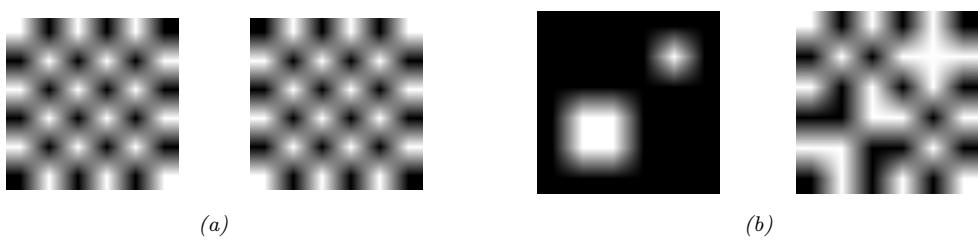


Figure 4.15: (a) The two ground states for a small ferromagnetic Ising model where $J = 1$. (b) Two different states for a small Ising model which have the same energy. Left: $J = 1$, so neighboring pixels have similar values. Right: $J = -1$, so neighboring pixels have different values. From Figures 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

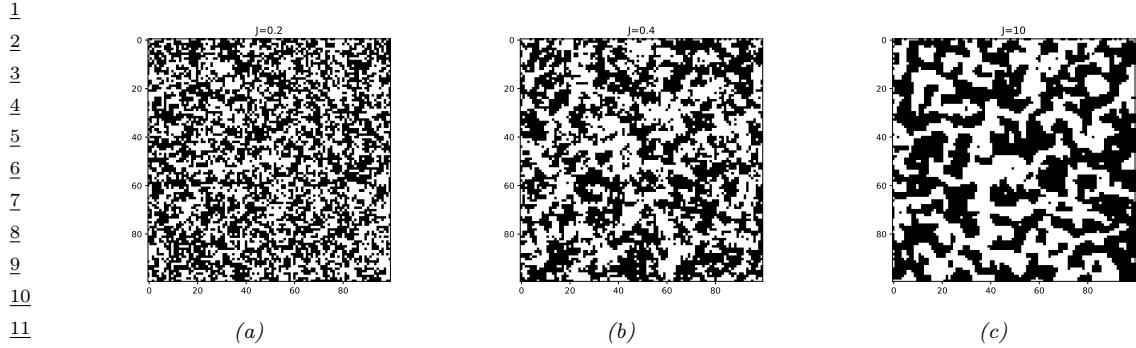


Figure 4.16: Samples from an associative Ising model with varying $J > 0$. Generated by `gibbs_demo_ising.py`.

clusters occur, and above which many small clusters occur. In the case of an isotropic square lattice model, one can show [Geo88] that

$$J_c = \frac{1}{2} \log(1 + \sqrt{2}) \approx 0.44 \quad (4.76)$$

This rapid change in global behavior as we vary a parameter of the system is called a **phase transition**. This can be used to explain how natural systems, such as water, can suddenly go from solid to liquid, or from liquid to gas, when the temperature changes slightly. See e.g., [Mac03, 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:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_i \psi_i(x_i; \boldsymbol{\theta}) \prod_{i \sim j} \psi_{ij}(x_i, x_j; \boldsymbol{\theta}) \quad (4.77)$$

The ψ_i terms can be thought of as a local bias term, that are independent of the contributions of the neighboring nodes. For binary nodes, we can define this as follows:

$$\psi_i(x_i) = \begin{cases} e^\alpha & \text{if } x_i = +1 \\ e^{-\alpha} & \text{if } x_i = -1 \end{cases} \quad (4.78)$$

If we write this as an energy based model, we have

$$\mathcal{E}(\mathbf{x}|\boldsymbol{\theta}) = -\alpha \sum_i x_i - J \sum_{i \sim j} x_i x_j \quad (4.79)$$

4.3.2.2 Potts models

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

$$\psi_{ij}(x_i = k, x_j = k') = e^{J_{ij}(k, k')} \quad (4.80)$$

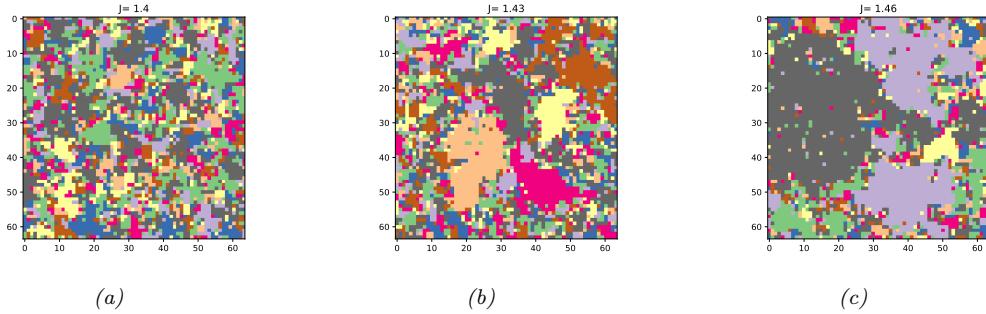


Figure 4.17: Visualizing a sample from a 10-state Potts model of size 128×128 . The critical value is $J_c = \log(1 + \sqrt{10}) = 1.426$. for different association strengths: (a) $J = 1.40$, (b) $J = 1.43$, (c) $J = 1.46$. Generated by [gibbs_demo_potts_jax.ipynb](#).

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).

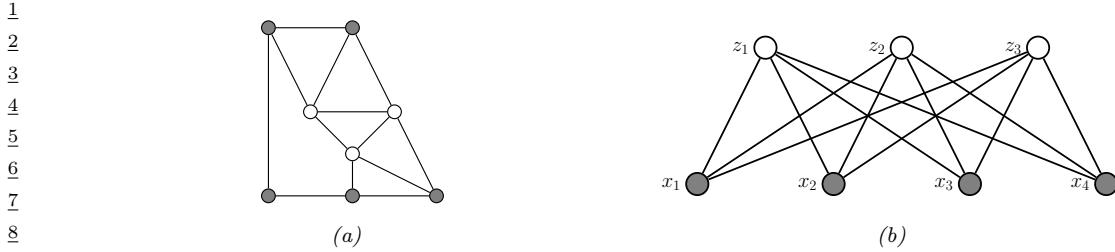


Figure 4.18: (a) A general Boltzmann machine, with an arbitrary graph structure. The shaded (visible) nodes are partitioned into input and output, although the model is actually symmetric and defines a joint distribution on all the nodes. (b) A restricted Boltzmann machine with a bipartite structure. 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.

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.

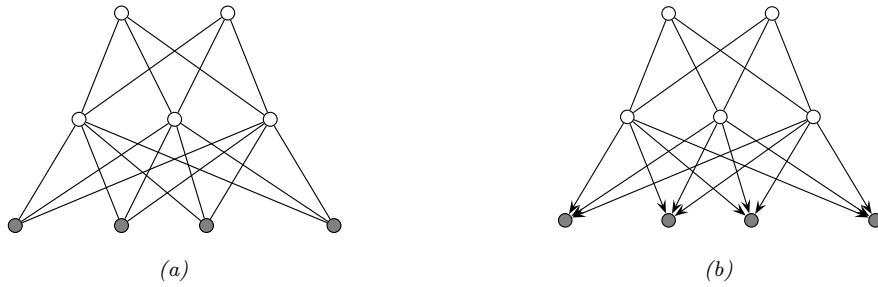


Figure 4.19: (a) Deep Boltzmann machine. (b) Deep belief network. The top two layers define the prior in terms on an RBM. The remaining layers are a directed graphical model that “decodes” the prior into observable data.

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

$$p(\mathbf{x}, \mathbf{z}_1, \mathbf{z}_2 | \boldsymbol{\theta}) = \frac{1}{Z(\mathbf{W}_1, \mathbf{W}_2)} \exp(\mathbf{x}^\top \mathbf{W}_1 \mathbf{z}_1 + \mathbf{z}_1^\top \mathbf{W}_2 \mathbf{z}_2) \quad (4.84)$$

where 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

$$p(\mathbf{x}, z_1, z_2 | \boldsymbol{\theta}) = p(\mathbf{x} | z_1, \mathbf{W}_1) \frac{1}{Z(\mathbf{W}_2)} \exp(z_1^\top \mathbf{W}_2 z_2) \quad (4.85)$$

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

⁸ 4.3.2.7 Maximum entropy and log-linear models

¹⁰ It is common to assume that the potential functions have the following log-linear form:
¹¹

$$\psi_c(\mathbf{x}_c; \boldsymbol{\theta}_c) = \exp(\boldsymbol{\theta}_c^\top \phi(\mathbf{x}_c)) \quad (4.86)$$

¹⁴ where $\phi(\mathbf{x}_c)$ is a feature vector derived from the variables in clique c . The overall model is then
¹⁵ given by
¹⁶

$$p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp \left(\sum_c \boldsymbol{\theta}_c^\top \phi(\mathbf{x}_c) \right) = \frac{1}{Z(\boldsymbol{\theta})} \exp (\boldsymbol{\theta}^\top \phi(\mathbf{x})) \quad (4.87)$$

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,
²³ subject to the constraints that the expected value of the features (sufficient statistics) $\phi(\mathbf{x})$ match
²⁴ the empirical expectations. Consequently, the model in Equation (4.87) is often called a **maximum**
²⁵ **entropy** or **maxent** model.

²⁶ For example, in a Gaussian model, we have

$$\phi([x_i, x_j]) = [x_i, x_j, x_i x_j] \quad (4.88)$$

³⁰ for $x_i \in \mathbb{R}$. And in an Ising model, we have

$$\phi([x_i, x_j]) = [x_i, x_j, x_i x_j] \quad (4.89)$$

³⁴ for $x_i \in \{-1, +1\}$. Thus both of these are maxent models.

³⁵ If the features ϕ 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

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

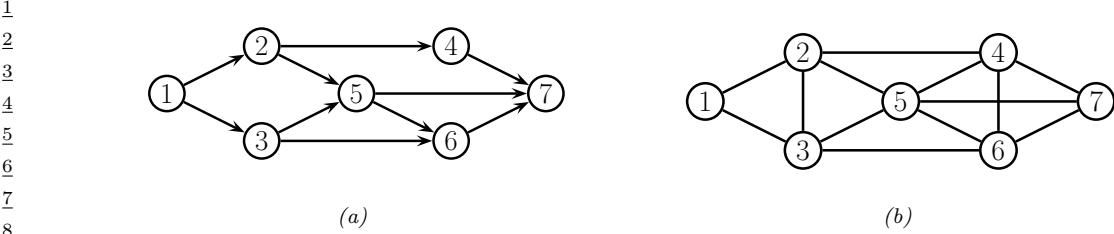


Figure 4.20: (a) A PGM-D. (b) Its moralized version, represented as a PGM-U.

the following form:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{i \sim j} \psi_{ij}(x_i, x_j) \prod_i \psi_i(x_i) \quad (4.90)$$

$$\psi_{ij}(x_i, x_j) = \exp\left(-\frac{1}{2}x_i \Lambda_{ij} x_j\right) \quad (4.91)$$

$$\psi_i(x_i) = \exp\left(-\frac{1}{2}\Lambda_{ii}x_i^2 + \eta_i x_i\right) \quad (4.92)$$

$$Z(\boldsymbol{\theta}) = (2\pi)^{D/2} |\Lambda|^{-\frac{1}{2}} \quad (4.93)$$

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 $\{X_1, X_2\} \perp \{X_6, X_7\} | \{X_3, X_4, X_5\}$.

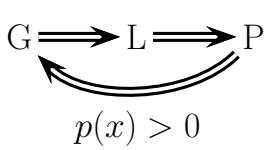


Figure 4.21: Relationship between Markov properties of PGM-U's.

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 $I(p) \neq I(G)$.

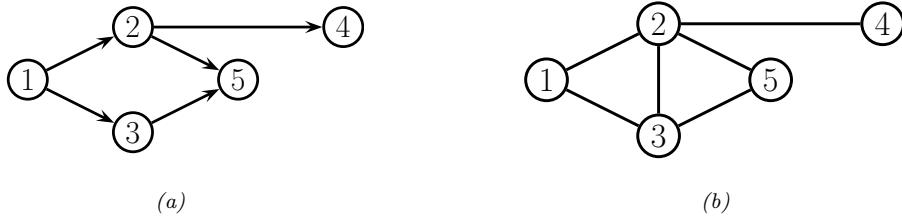


Figure 4.22: (a) The ancestral graph induced by the DAG in Figure 4.20(a) wrt $U = \{X_2, X_4, X_5\}$. (b) The moralized version of (a).

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).

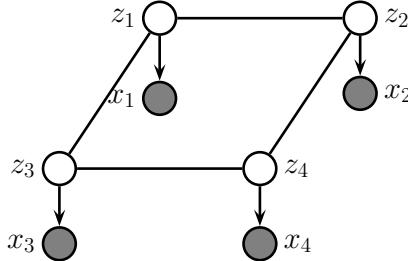
In the special case of PGM-U’s with low treewidth and discrete or Gaussian potentials, it is possible to use the junction tree algorithm to draw samples using dynamic programming (see Section 9.5.4).

4.3.5 Inference

We discuss inference in graphical models in detail in Chapter 9. In this section, we just give an example.

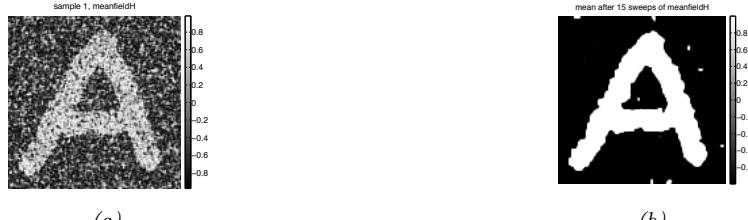
Suppose we have an image composed of binary pixels, z_i , but we only observe noisy versions of the

1
2
3
4
5
6
7
8
9
10



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
14
15
16
17
18
19
20



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

$$29 \quad p(\mathbf{x}, \mathbf{z}) = p(\mathbf{z})p(\mathbf{x}|\mathbf{z}) = \left[\frac{1}{Z} \sum_{i \sim j} \psi_{ij}(z_i, z_j) \right] \prod_i p(x_i|z_i) \quad (4.97)$$

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
45
46
47

In this section, we discuss how to estimate the parameters for an MRF. As we will see, computing the MLE can be computationally expensive, even in the fully observed case, because of the need to deal with the partition function $Z(\boldsymbol{\theta})$. And computing the posterior over the parameters, $p(\boldsymbol{\theta}|\mathcal{D})$,

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:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp \left(\sum_c \boldsymbol{\theta}_c^\top \boldsymbol{\phi}_c(\mathbf{x}) \right) \quad (4.98)$$

where c indexes the cliques. The log-likelihood becomes

$$\ell(\boldsymbol{\theta}) \triangleq \frac{1}{N} \sum_n \log p(\mathbf{x}_n|\boldsymbol{\theta}) = \frac{1}{N} \sum_n \left[\sum_c \boldsymbol{\theta}_c^\top \boldsymbol{\phi}_c(\mathbf{x}_n) - \log Z(\boldsymbol{\theta}) \right] \quad (4.99)$$

Its gradient is given by

$$\frac{\partial \ell}{\partial \boldsymbol{\theta}_c} = \frac{1}{N} \sum_n \left[\boldsymbol{\phi}_c(\mathbf{x}_n) - \frac{\partial}{\partial \boldsymbol{\theta}_c} \log Z(\boldsymbol{\theta}) \right] \quad (4.100)$$

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.,

$$\frac{\partial \log Z(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_c} = \mathbb{E} [\boldsymbol{\phi}_c(\mathbf{x})|\boldsymbol{\theta}] = \sum_{\mathbf{x}} p(\mathbf{x}|\boldsymbol{\theta}) \boldsymbol{\phi}_c(\mathbf{x}) \quad (4.101)$$

Hence the gradient of the log likelihood is

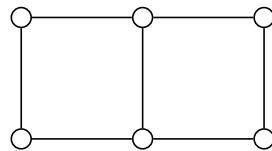
$$\frac{\partial \ell}{\partial \boldsymbol{\theta}_c} = \frac{1}{N} \sum_n [\boldsymbol{\phi}_c(\mathbf{x}_n)] - \mathbb{E} [\boldsymbol{\phi}_c(\mathbf{x})] \quad (4.102)$$

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:

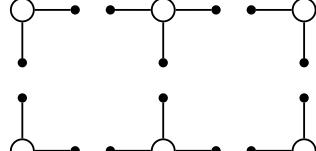
$$\mathbb{E}_{p_{\mathcal{D}}} [\boldsymbol{\phi}_c(\mathbf{x})] = \mathbb{E}_{p(\mathbf{x}|\boldsymbol{\theta})} [\boldsymbol{\phi}_c(\mathbf{x})] \quad (4.103)$$

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.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17



(a)



(b)

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

2 4.3.6.3 Maximum pseudo-likelihood estimation

3 When fitting fully visible MRFs (or CRFs), a simple alternative to maximizing the likelihood is to
4 maximize the **pseudo likelihood** [Bes75], defined as follows:
5

6

$$\ell_{PL}(\boldsymbol{\theta}) \triangleq \frac{1}{N} \sum_{n=1}^N \sum_{d=1}^D \log p(x_{nd} | \mathbf{x}_{n,-d}, \boldsymbol{\theta}) \quad (4.107)$$

7

8 That is, we optimize the product of the full conditionals, also known as the **composite likelihood**
9 [Lin88a; DL10; VRF11]. Compare this to the objective for maximum likelihood:
10

11

$$\ell_{ML}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N \log p(\mathbf{x}_n | \boldsymbol{\theta}) \quad (4.108)$$

12

13 In the case of Gaussian MRFs, PL is equivalent to ML [Bes75], although this is not true in general.
14 Nevertheless, it is a consistent estimator in the large sample limit [LJ08].

15 The PL approach is illustrated in Figure 4.25 for a 2d grid. We learn to predict each node, given all
16 of its neighbors. This objective is generally fast to compute since each full conditional $p(x_d | \mathbf{x}_{-d}, \boldsymbol{\theta})$
17 only requires summing over the states of a single node, x_d , in order to compute the local normalization
18 constant. The PL approach is similar to fitting each full conditional separately, except that, in PL,
19 the parameters are tied between adjacent nodes.

20 Experiments in [PW05; HT09] suggest that PL works as well as exact ML for fully observed Ising
21 models, but is much faster. In [Eke+13], they use PL to fit Potts models to (aligned) protein
22 sequence data. However, when fitting RBMs, [Mar+10] found that PL is worse than some of the
23 stochastic ML methods we discuss in Section 25.2.

24 Another more subtle problem is that each node assumes that its neighbors have known values
25 during training. If node $j \in \text{nbr}(i)$ is a perfect predictor for node i , then j will learn to rely completely
26 on node i , even at the expense of ignoring other potentially useful information, such as its local
27 evidence, say y_i . At test time, the neighboring nodes will not be observed, and performance will
28 suffer.⁷

29

30 4.3.6.4 Learning from incomplete data

31 In this section, we consider parameter estimation for MRFs (and CRFs) with hidden variables. Such
32 **incomplete data** can arise for several reasons. For example, we may want to learn a model of
33 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
34 we only observe \mathbf{x} , the model is called a **hidden Gibbs random field**. See Section 10.2.2 for an
35 example. As another example, we may have a CRF in which the hidden variables are used to encode
36

37 6. In the case of decomposable graphs, IPF converges in a single iteration. Intuitively, this is because a decomposable
38 graph can be converted to a DAG without any loss of information, as explained in Section 4.4, and we know that we
39 can compute the MLE for tabular CPDs in closed form, just by normalizing the counts.

40 7. Geoff Hinton has an analogy for this problem. Suppose we want to learn to denoise images of symmetric shapes,
41 such as Greek vases. Each hidden pixel x_i depends on its spatial neighbors, as well as the noisy observation y_i . Since its
42 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
43 not be available at test time.

1
2 an unknown alignment between the inputs and outputs [Qua+07], or to model missing parts of the
3 input [SRS10].

4 We now discuss how to compute the MLE in such cases. For notational simplicity, we focus on
5 unconditional models (MRFs, not CRFs), and we assume all the potentials are log-linear. In this
6 case, the model has the following form:

$$\frac{7}{8} p(\mathbf{x}, \mathbf{z} | \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{\theta}^\top \phi(\mathbf{x}, \mathbf{z}))}{Z(\boldsymbol{\theta})} = \frac{\tilde{p}(\mathbf{x}, \mathbf{z} | \boldsymbol{\theta})}{Z(\boldsymbol{\theta})} \quad (4.109)$$

$$\frac{10}{11} Z(\boldsymbol{\theta}) = \sum_{\mathbf{x}, \mathbf{z}} \exp(\boldsymbol{\theta}^\top \phi(\mathbf{x}, \mathbf{z})) \quad (4.110)$$

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

14 The log likelihood is now given by

$$\frac{15}{16} \ell(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N \log \left(\sum_{\mathbf{z}_n} p(\mathbf{x}_n, \mathbf{z}_n | \boldsymbol{\theta}) \right) \quad (4.111)$$

$$\frac{18}{19} = \frac{1}{N} \sum_{n=1}^N \log \left(\frac{1}{Z(\boldsymbol{\theta})} \sum_{\mathbf{z}_n} \tilde{p}(\mathbf{x}_n, \mathbf{z}_n | \boldsymbol{\theta}) \right) \quad (4.112)$$

$$\frac{21}{22} = \frac{1}{N} \sum_{n=1}^N \left[\log \sum_{\mathbf{z}_n} \tilde{p}(\mathbf{x}_n, \mathbf{z}_n | \boldsymbol{\theta}) \right] - \log Z(\boldsymbol{\theta}) \quad (4.113)$$

24 Note that

$$\frac{26}{27} \log \sum_{\mathbf{z}_n} \tilde{p}(\mathbf{x}_n, \mathbf{z}_n | \boldsymbol{\theta}) = \log \sum_{\mathbf{z}_n} \exp(\boldsymbol{\theta}^\top \phi(\mathbf{x}_n, \mathbf{z}_n)) \triangleq \log Z(\boldsymbol{\theta}, \mathbf{x}_n) \quad (4.114)$$

29 where $Z(\boldsymbol{\theta}, \mathbf{x}_n)$ 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

$$\frac{34}{35} \frac{\partial \ell}{\partial \boldsymbol{\theta}} = \frac{1}{N} \sum_n [\mathbb{E}_{\mathbf{z} \sim p(\mathbf{z} | \mathbf{x}_n, \boldsymbol{\theta})} [\phi(\mathbf{x}_n, \mathbf{z})]] - \mathbb{E}_{(\mathbf{z}, \mathbf{x}) \sim p(\mathbf{z}, \mathbf{x} | \boldsymbol{\theta})} [\phi(\mathbf{x}, \mathbf{z})] \quad (4.115)$$

38 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.

42

43 4.4.1 CI properties

44 Which model has more “expressive power”, a PGM-D or a PGM-U? To formalize the question,
45 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
46

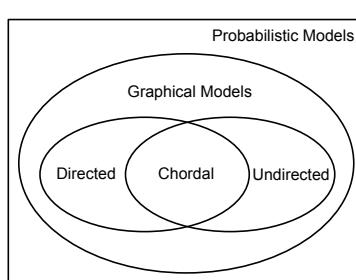


Figure 4.26: PGM-D's and PGM-U's can perfectly represent different sets of distributions. Some distributions can be perfectly represented by either PGM-D's or PGM-U's; the corresponding graph must be chordal.

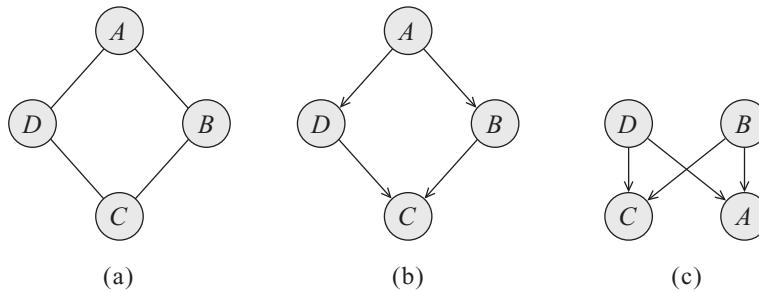


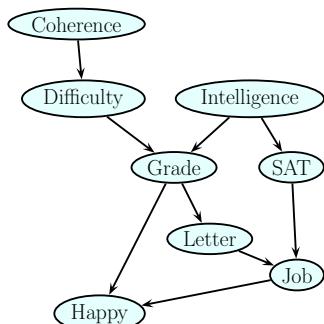
Figure 4.27: A PGM-U and two failed attempts to represent it as a PGM-U. From Figure 3.10 of [KF09a]. Used with kind permission of Daphne Koller.

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.

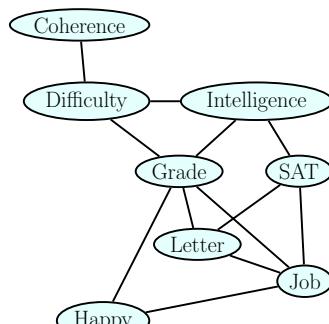
As an example of some CI relationships that can be perfectly modeled by a PGM-D but not a PGM-U, consider a v-structure $A \rightarrow C \leftarrow B$. This asserts that $A \perp B$, and $A \not\perp B|C$. If we drop the arrows, we get $A - C - B$, which asserts $A \perp B|C$ and $A \not\perp B$, which 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



(a)



(b)

14 Figure 4.28: Left: The full student PGM-D. Right: the equivalent PGM-U. We add moralization arcs D-I,
15 G-J and L-S. Adapted from Figure 9.8 of [KF09a].
16
17

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 4.4.2.1 Converting a PGM-D to a PGM-U 35

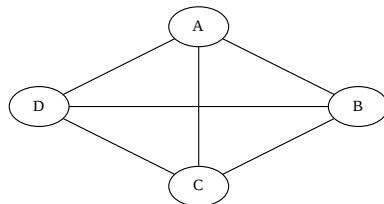
36 We can easily convert a PGM-D to a PGM-U as follows. First, any “unmarried” parents that share a
37 child must get “married”, by adding an edge between them; this process is known as **moralization**.
38 Then we can drop the arrows, resulting in an undirected graph. The reason we need to do this is 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

42 Let us consider an example from [KF09a]. We will use the (full version of the student network
43 shown in Figure 4.28(a). The corresponding joint has the following form:
44

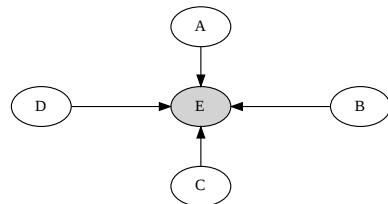
$$P(C, D, I, G, S, L, J, H) \quad (4.116)$$

$$= P(C)P(D|C)P(I)P(G|I, D)P(S|I)P(L|G)P(J|L, S)P(H|G, J) \quad (4.117)$$

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17



(a)



(b)

Figure 4.29: (a) An undirected graphical model. (b) A directed equivalent, obtained by adding a dummy observed child node.

18
19
20
21
22
23
24
25
26
27
28
29

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 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.

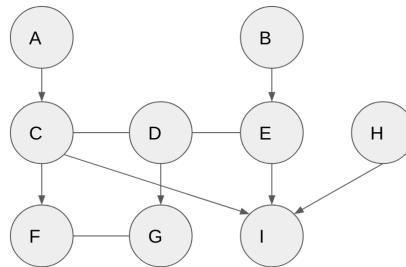
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

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.

1
2
3
4
5
6
7
8
9
10



11 *Figure 4.30: A partially directed acyclic graph (PDAG). The chain components are {A}, {B}, {C, D, E},
12 {F, G}, {H} and {I}. Adapted from Figure 4.15 of [KF09a].*
13

14

15

16 4.4.3.1 Chain graphs

17 A **chain graph** is a PGM which may have both directed and undirected edges, but without any
18 directed cycles. A simple example is shown in Figure 10.2, which defines the following joint model:
19

20

$$21 p(\mathbf{x}_{1:D}, \mathbf{y}_{1:D}) = p(\mathbf{x}_{1:D})p(\mathbf{y}_{1:D}|\mathbf{x}_{1:D}) = \left[\frac{1}{Z} \prod_{(ij)} \psi_{ij}(x_i, x_j) \right] \left[\prod_{i=1}^D p(y_i|x_i) \right] \quad (4.120)$$

22

23 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
24 fully factorized PGM-D.

25 More generally, a chain graph can be defined in terms of a **partially directed acyclic graph**
26 (**PDAG**). This is a graph which can be decomposed into a directed graph of **chain components**,
27 where the nodes within each chain component are connected with each other only with undirected
28 edges. See Figure 4.30 for an example.

29 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
30 component, and each CPD is a conditional random field. For example, referring to Figure 4.30, we
31 have

32

$$33 p(A, B, \dots, I) = p(A)p(B)p(C, D, E|A, B)p(F, G|C, D)p(H)p(I|C, E, H) \quad (4.121)$$

34

$$35 p(C, D, E|A, B) = \frac{1}{Z(A, B)} \phi(A, C)\phi(B, E)\phi(C, D)\phi(D, E) \quad (4.122)$$

36

$$37 p(F, G|C, D) = \frac{1}{Z(C, D)} \phi(F, C)\phi(G, D)\phi(F, G) \quad (4.123)$$

38 For more details, see e.g., [KF09a, Sec 4.6.2].

39

40 4.4.3.2 Acyclic directed mixed graphs

41 One can show [Pea09b, p51] that every latent variable PGM-D can be rewritten in a way such that
42 every latent variable is a root node with exactly two observed children. This is called the **projection**
43 of the latent variable PGM, and is observationally indistinguishable from the original model.

44

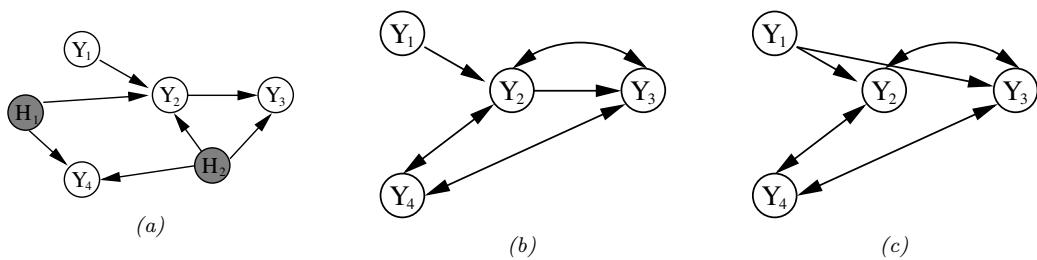


Figure 4.31: (a) A DAG with two hidden variables (shaded). (b) The corresponding ADMG. The bidirected edges reflect correlation due to the hidden variable. (c) A Markov equivalent ADMG. From Figure 3 of [SG09]. Used with kind permission of Ricardo Silva.

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

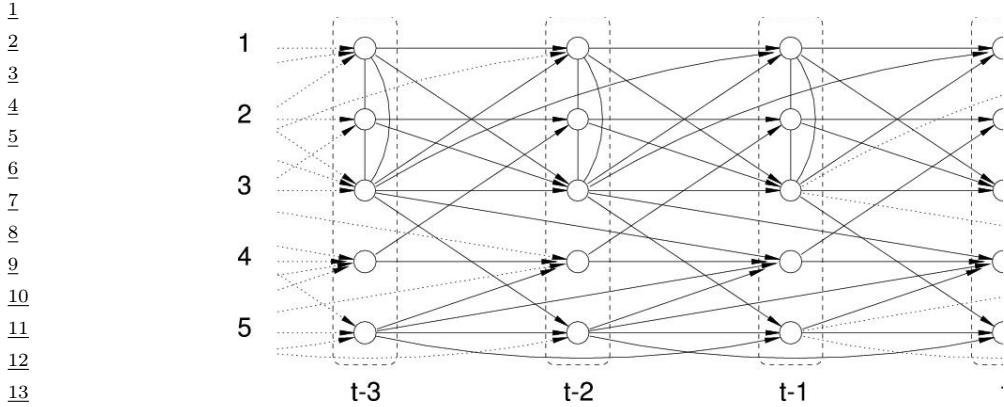


Figure 4.32: A VAR(2) process represented as a dynamic chain graph. From [DE00]. Used with kind permission of Rainer Dahlhaus.

PGM-U. For example, suppose we have

$$\mathbf{A}_1 = \begin{pmatrix} \frac{3}{5} & 0 & \frac{1}{5} & 0 & 0 \\ 0 & \frac{3}{5} & 0 & -\frac{1}{5} & 0 \\ \frac{2}{5} & \frac{1}{5} & \frac{3}{5} & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{2} & \frac{1}{5} \\ 0 & 0 & \frac{1}{5} & 0 & \frac{2}{5} \end{pmatrix}, \quad \mathbf{A}_2 = \begin{pmatrix} 0 & 0 & -\frac{1}{5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{5} & 0 & \frac{1}{3} \\ 0 & 0 & 0 & 0 & -\frac{1}{5} \end{pmatrix} \quad (4.125)$$

and

$$\boldsymbol{\Sigma} = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & 0 & 0 \\ \frac{1}{2} & 1 & -\frac{1}{3} & 0 & 0 \\ \frac{1}{3} & -\frac{1}{3} & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad \boldsymbol{\Sigma}^{-1} = \begin{pmatrix} 2.13 & -1.47 & -1.2 & 0 & 0 \\ -1.47 & 2.13 & 1.2 & 0 & 0 \\ -1.2 & 1.2 & 1.8 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.126)$$

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 $\boldsymbol{\Sigma}^{-1}$ 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 $\Lambda = \boldsymbol{\Sigma}^{-1}$.



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

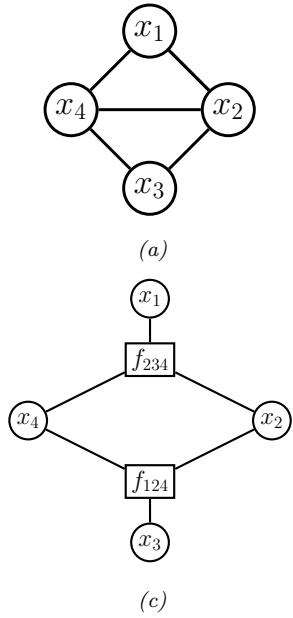
$$f(x_1, x_2, x_3, x_4) = f_{14}(x_1, x_4)f_{12}(x_1, x_2)f_{34}(x_3, x_4)f_{23}(x_2, x_3)f_{24}(x_2, x_4) \quad (4.128)$$

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



23 Figure 4.34: (a) A simple PGM-U. (b) A factor graph representation assuming one potential per maximal
24 clique. (c) Same as (b), but graph is visualized differently. (d) A factor graph representation assuming one
25 potential per edge.

26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

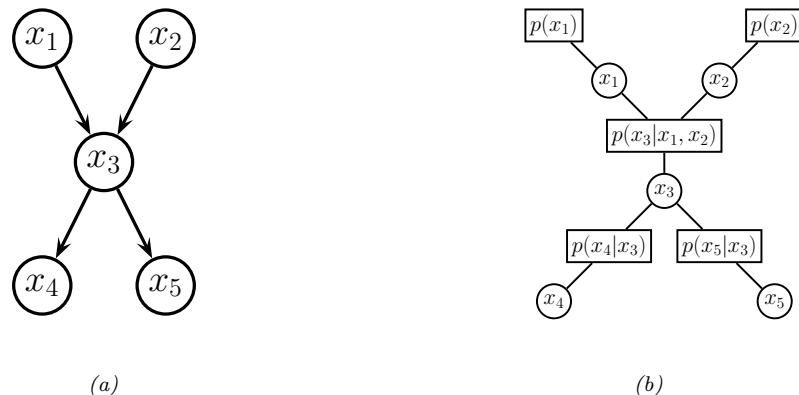


Figure 4.35: (a) A simple PGM-D. (b) Its corresponding factor graph.

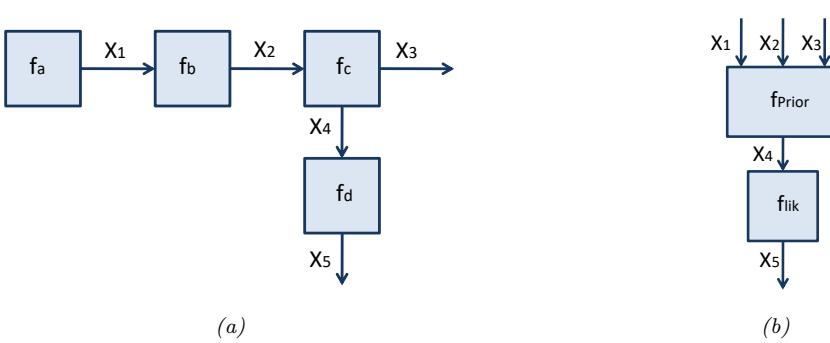


Figure 4.36: A Forney factor graph. (a) Directed version. (b) Hierarchical version.

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

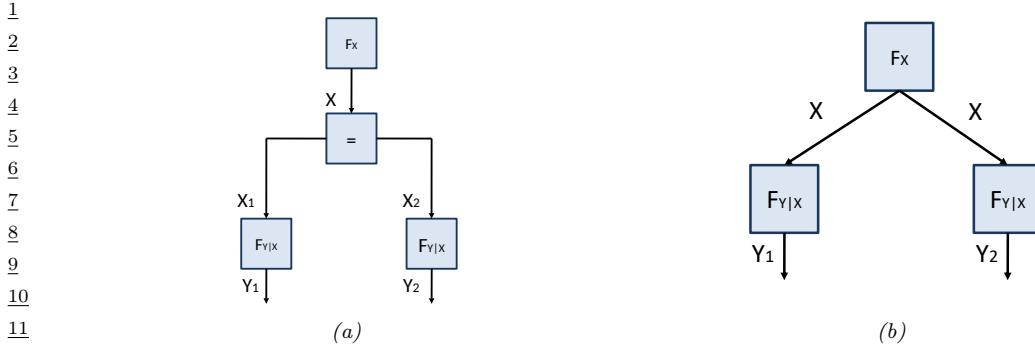


Figure 4.37: An FFG with an equality constraint node.

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).

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.

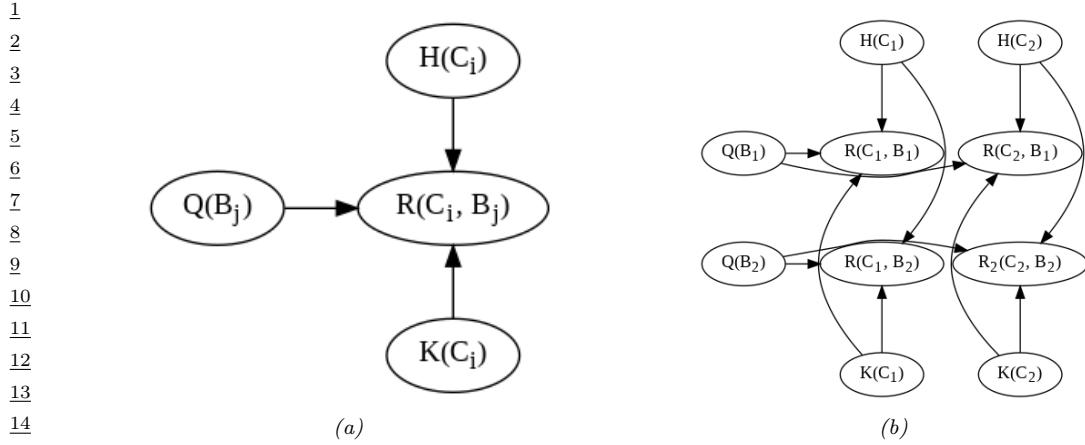


Figure 4.38: RPM for the book review domain. (a) Template for a generic customer C_i and book B_j pair. R is rating, Q is quality, H is honesty, and K is kindness. (b) Unrolled model for 2 books and 2 customers.

about what kind of ratings a book receives based on the quality of the book, but also properties of the reviewer, such as their honest and kindness. (More sophisticated models of human raters in the context of crowd-sourced data collection can be found in e.g., [LRC19].)

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 $2^{C5^{C+B+BC}}$, 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.

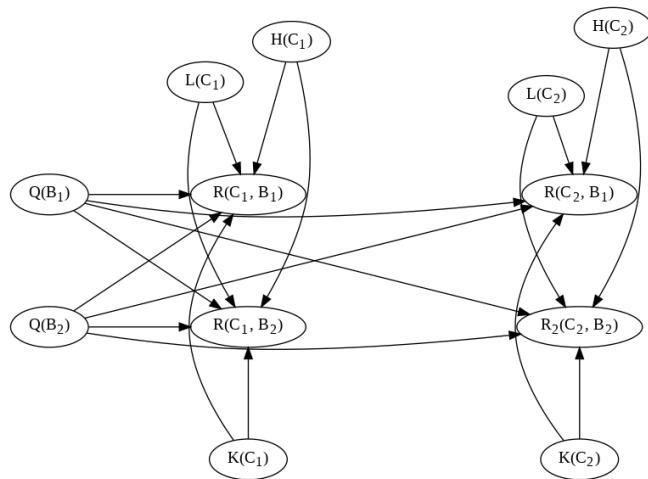
Things get more interesting when we don’t know which objects are being referred to. For example, customer C_1 might write a review of a book called “Probabilistic Machine Learning”, but do they mean edition 1 (B_1) or edition 2 (B_2)? To handle this kind of **relational uncertainty**, we can add all 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

$$R(c, b) \sim \text{RecCPT}(H(c), K(c), Q(b')) \text{ where } b' = L(c) \quad (4.142)$$

This CPD acts like a **multiplexer**, where the $L(c)$ node specifies which of the parents $Q(1 : B)$ to actually use.

47

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16



17
18
19
20

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, $L(c)$.

21
22

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).

29

30
31

4.5.3 Open-universe probability models

32
33
34

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.

35

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).

43
44
45
46
47

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

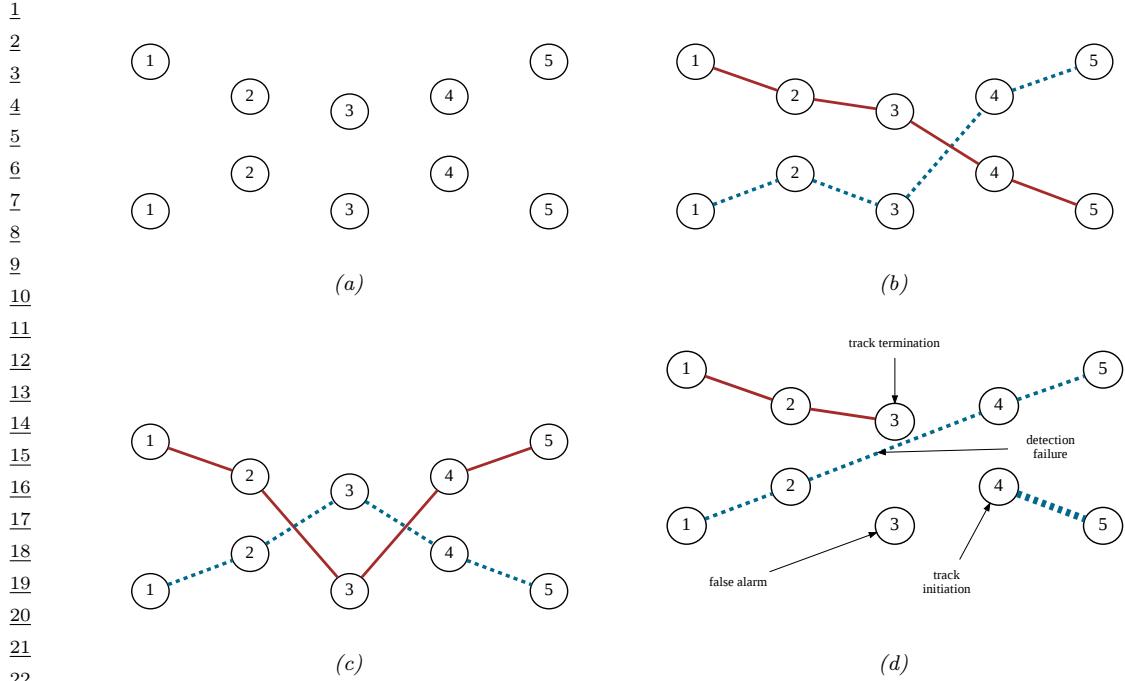


Figure 4.40: Illustration of multi-target tracking in 2d over 5 time steps. (a) We observe 2 measurements per time step. (b-c) Possible hypotheses about the underlying object tracks. (d) A more complex hypothesis in which the red track stops at step 3, the dashed red track starts at step 4 the dotted 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

2

3

4 4.5.4 Programs as probability models

5

6 OUPMs, discussed in Section 4.5.3, let us define probability models over complex dynamic state
7 spaces of unbounded and variable size. The set of possible worlds correspond to objects and their
8 attributes and relationships. Another approach is to use a **probabilistic programming language**
9 or **PPL**, in which we define the set of possible words as the set of **execution traces** generated by
10 the program when it is endowed with a random choice mechanism. (This is a **procedural approach**
11 to the problem, whereas OUPMs are a **declarative approach**.)

12 The difference between a probabilistic programming language and a standard one was described in
13 [Gor+14] as follows: “Probabilistic programs are usual functional or imperative programs with two
14 added constructs: (1) the ability to draw values at random from distributions, and (2) the ability to
15 condition values of variables in a program via observation”. The former is a way to define $p(\mathbf{z}, \mathbf{y})$,
16 and the latter is the same as standard Bayesian conditioning $p(\mathbf{z}|\mathbf{y})$.

17 Some recent examples of PPLs include **Gen** [CT+19], **Pyro** [Bin+19] and **Turing** [GXG18].
18 Inference in such models is often based on SMC, which we discuss in Chapter 13. For more details
19 on PPLs, see e.g. [Mee+18].

20 4.6 Structural causal models

21

22 While probabilities encode our beliefs about a static world, causality tells us whether and how
23 probabilities change when the world changes, be it by intervention or by act of imagination. —
24 Judea Pearl, [PM18b].

25 In this section, we discuss how we can use directed graphical model notation to represent **causal**
26 **models**. We discuss causality in greater detail in Chapter 38, but we introduce some basic ideas and
27 notation here, since it is foundational material that we will need in other parts of the book.

28 The core idea behind causal models is to create a mechanistic model of the world in which we
29 can reason about the effects of local changes. The canonical example is an electronic circuit: we
30 can predict the effects of any action, such as “knocking out” a particular transistor, or changing the
31 resistance level of a wire, by modifying the circuit locally, and then “re-running” it from the same
32 initial conditions.

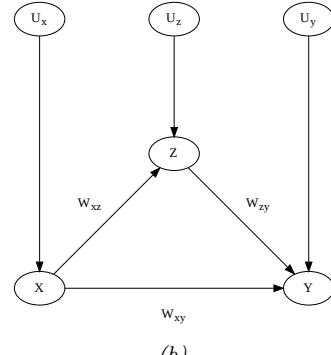
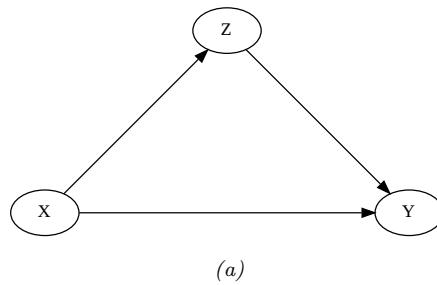
33 We can generalize this idea to create a **structural causal models** or **SCM** [PGJ16], also called
34 **functional causal model** [Sch19]. An SCM is a triple $\mathcal{M} = (\mathcal{U}, \mathcal{V}, \mathcal{F})$, where $\mathcal{U} = \{U_i : i = 1 : N\}$
35 is a set of unexplained or **exogenous** “noise” variables, which are passed as input to the model,
36 $\mathcal{V} = \{V_i : i = 1 : N\}$ is a set of **endogeneous** variables that are part of the model itself, and
37 $\mathcal{F} = \{f_i : i = 1 : N\}$ is a set of deterministic functions of the form $V_i = f_i(V_{\text{pa}_i}, U_i)$, where pa_i are the
38 parents of variable i , and $U_i \in \mathcal{U}$ are the external inputs. We assume the equations can be structured
39 in a **recursive** way, so the dependency graph of nodes given their parents is a DAG. Finally, we
40 assume our model is **causally sufficient**, which means that \mathcal{V} and \mathcal{U} are all of the causally relevant
41 factors (although they may not all be observed). This is called the “**causal Markov assumption**”.

42 Of course, a model typically cannot represent all the variables that might influence observations or
43 decisions. After all, models are *abstractions* of reality. The variables that we choose not to model

44 11. See <https://tinyurl.com/2svy5tmh>.

45

1
2
3
4
5
6
7
8
9
10
11
12
13



14 *Figure 4.41: (a) PGM for modeling relationship between salary, education and work experience. (b) Corresponding SCM.*

15
16
17

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, $p(V)$, 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 .

27

28 4.6.1 Example: causal impact of education on wealth

29 We now give a simple example of an SCM, based on [PM18b, p276]. Suppose we are interested in
30 the causal effect of education on wealth. Let X represent the level of education of a person (on
31 some numeric scale, say 0 = high school, 1 = college, 2 = graduate school), and Y represent their
32 wealth (at some moment in time). In some cases we might expect that increasing X would increase Y
33 (although it of course depends on the nature of the degree, the nature of the job, etc). Thus we add
34 an edge from X to Y . However, getting more education can cost a lot of money (in certain countries),
35 which is a potentially confounding factor on wealth. Let Z be the debt incurred by a person based
36 on their education. We add an edge from X to Z to reflect the fact that larger X means larger Z (in
37 general), and we add an edge from Z to Y to reflect that larger Z means lower Y (in general).

38 We can represent our structural assumptions graphically as shown in Section 4.6.1(a). The
39 corresponding SCM has the form:

40
$$X = f_X(U_x) \tag{4.143}$$

41
$$Z = f_Z(X, U_z) \tag{4.144}$$

42
$$Y = f_Y(X, Z, U_y) \tag{4.145}$$

43 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

1 represent the exogeneous noise terms as shown in Section 4.6.1(b); this makes clear our assumption
2 that the noise terms are a-priori independent. (We return to this point later.)
3

4 4.6.2 Structural equation models

5 A **structural equation model** [Bol89; BP13], also known as a **path diagram**, is a special case of
6 a structural causal model in which all the functional relationships are linear, and the prior on the
7 noise terms is Gaussian. SEMs are widely used in economics and social science, due to the fact that
8 they have a causal interpretation, yet they are computationally tractable.
9

10 For example, let us make an SEM version of our education example. We have
11

$$\underline{X} = U_x \tag{4.146}$$

$$\underline{Z} = c_z + w_{xz}X + U_z \tag{4.147}$$

$$\underline{Y} = c_y + w_{xy}X + w_{zy}Z + U_y \tag{4.148}$$

12 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
13 can be converted to the following Gaussian DGM:
14

$$p(X) = \mathcal{N}(X|\mu_x, \sigma_x^2) \tag{4.149}$$

$$p(Z|X) = \mathcal{N}(Z|c_z + w_{xz}X, \sigma_z^2) \tag{4.150}$$

$$p(Y|X, Z) = \mathcal{N}(Y|c_y + w_{xy}X + w_{zy}Z, \sigma_y^2) \tag{4.151}$$

15 We can relax the linearity assumption, to allow arbitrarily flexible functions, and relax the Gaussian
16 assumption, to allow any noise distribution. The resulting “nonparametric SEMs” are equivalent to
17 structural causal models. (For a more detailed comparison between SEMs and SCMs, see [Pea12;
18 BP13; Shi00b].)
19

20 4.6.3 Do operator and augmented DAGs

21 One of the main advantages of SCMs is that they let us predict the effect of **interventions**, which
22 are actions that change one or more local mechanisms. A simple intervention is to force a variable to
23 have a given value, e.g., we can force a gene to be “on” or “off”. This is called a **perfect intervention**
24 and is written as $\text{do}(X_i = x_i)$, where we have introduced new notation for the “**do**” operator (as
25 in the verb “to do”). This notation means we actively clamp variable X_i to value x_i (as opposed to
26 just observing that it has this value). Since the value of X_i is now independent of its usual parents,
27 we should “cut” the incoming edges to node X_i in the graph. This is called the “**graph surgery**”
28 operation.

29 In Figure 4.42a we illustrate this for our education SCM, where we force Z to have a given value.
30 For example, we may set $Z = 0$, by paying off everyone’s student debt. Note that $p(X|\text{do}(Z = z)) \neq$
31 $p(X|Z = z)$, since the intervention changes the model. For example, if we see someone with a debt of
32 0, we may infer that they probably did not get higher education, i.e., $p(X \geq 1|Z = 0)$ is small; but if
33 we pay off everyone’s college loans, then observing someone with no debt in this modified world should
34 not change our beliefs about whether they got higher education, i.e., $p(X \geq 1|\text{do}(Z = 0)) = p(X \geq 1)$.
35

36 In more realistic scenarios, we may not be able to set a variable to a specific value, but we may
37 be able to change it from its current value in some way. For example, we may be able to reduce
38

1
2
3
4
5
6
7
8
9
10
11
12
13



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

42

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 .

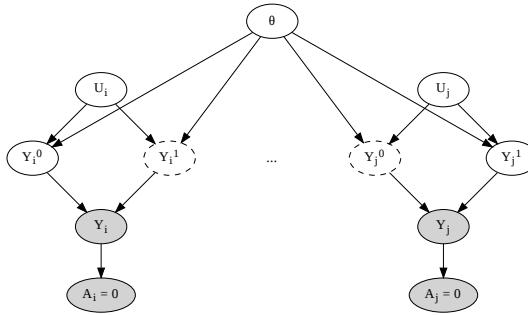
4.6.5 Counterfactuals

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

Level	Activity	Questions	Examples
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?
2: Intervention. $p(Y \text{do}(a))$	Doing	What if I do A ?	If I take aspirin, will my headache be cured?
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?

Table 4.5: Pearl’s causal hierarchy. Adapted from Table 1 of [Pea19].

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 $Y_2 = Y_2^1 = y_2$.

anyway?”. This is an example where we want to reason about the **causes of effects**, rather than just the **effects of causes**. This is crucial for legal reasoning (see e.g., [DMM17]), as well as for tasks like explainability and fairness.

Counterfactual reasoning requires strictly more assumptions than reasoning about interventions, as we discuss below. Indeed, Judea Pearl has proposed what he calls the **causal hierarchy** [Pea09b; PGJ16; PM18b], which has three levels of analysis, each more powerful than the last, but each making stronger assumptions. See Table 4.5 for a summary.

In counterfactual reasoning, we want to answer questions of the type $p(Y^{a'}|\text{do}(a), y)$, which is read as: “what is the probability distribution over outcomes Y if I were to do a' , given that I have already done a and observed outcome y ”. (We can also condition on any other evidencee that was observed, such as covariates \mathbf{x} .) The quantity $Y^{a'}$ is often called a **potential outcome** [Rub74], since it is the outcome that would occur in a hypothetical world in which you did a' instead of a . (Note that $p(Y^{a'} = y)$ is equivalent to $p(Y = y|\text{do}(a'))$, and is an interventional prediction, not a counterfactual one.)

The assumptions behind the potential outcomes framework can be clearly expressed using a structural causal model. We illustrate this in Figure 4.43 for a simple case where there are two possible actions. We see that we have a set of “**units**”, such as individual patients, indexed by

¹ subscripts. Each unit is associated with a hidden exogeneous random noise source, U_i , that captures
² everything that is unique about that unit. This noise gets deterministically mapped to two potential
³ outcomes, Y_i^0 and Y_i^1 , depending on which action is taken. For any given unit, we only get to observe
⁴ one of the outcomes, namely the one corresponding to the action that was actually chosen. In the
⁵ 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
⁶ action $A_2 = 1$, so we get to see $Y_2^1 = y_2$. The fact that we cannot simultaneously see both outcomes
⁷ for the same unit is called the “**fundamental problem of causal inference**” [Hol86].
⁸

⁹ We will assume the noise sources are independent, which is known as the “stable unit treatment
¹⁰ value assumption” or **SUTVA**. (This would not be true if the treatment on person j could somehow
¹¹ affect the outcome of person i , e.g., due to spreading disease or information between i and j .) We
¹² also assume that the deterministic mechanisms that map noise to outcomes are the same across
¹³ all units (represented by the shared parameter vector θ in Figure 4.43). We need to make one final
¹⁴ assumption, namely that the exogeneous noise is not affected by our actions. (This is a formalization
¹⁵ of the assumption known as “all else being equal”, or (in legal terms) “**ceteris paribus**”).

¹⁶ With the above assumptions, we can predict what the outcome *for an individual unit* would have
¹⁷ been in the alternative universe where we picked the other action. The procedure is as follows: first
¹⁸ we perform **abduction** using SCM G , to infer $p(U_i|A_i = a, Y_i = y_i)$, which is the posterior over
¹⁹ the latent factors for unit i given the observed evidence in the actual world; second we perform
²⁰ **intervention**, in which we modify the causal mechanisms of G by replacing $A_i = a$ with $A_i = a'$ to
²¹ get $G_{a'}$; third we perform **prediction**, in which we propagate the distribution of the latent factors,
²² $p(U_i|A_i = a, Y_i = y_i)$, through the modified SCM $G_{a'}$ to get $p(Y_i^{a'}|A_i = a, Y_i = y_i)$.

²³ In Figure 4.43, we see that we have two copies of every possible outcome variable, to represent
²⁴ the set of possible worlds. Of course, we only get to see one such world, based on the actions that
²⁵ we actually took. More generally, a model in which we “clone” all the deterministic variables, with
²⁶ the noise being held constant between the two branches of the graph for the same unit, is called a
²⁷ **twin network** [Pea09b]. We will see a more practical example in Section 19.3.5, where we discuss
²⁸ assessing the counterfactual causal impact of an intervention in a time series. (See also [RR11; RR13],
²⁹ who propose a related formalism known as **single world intervention graph** or **SWIG**.)

³⁰ We see from the above that the potential outcomes framework is mathematically equivalent to
³¹ structural causal models, but does not use graphical model notation. This has led to heated debate
³² between the founders of the two schools of thought.¹² The SCM approach is more popular in
³³ computer science (see e.g., [PJS17; Sch19; Sch+21b]), and the PO approach is more popular in
³⁴ economics (see e.g. [AP09; Imb19]). Modern textbooks on causality usually use both formalisms (see
³⁵ e.g., [HR20a; Nea20]).

³⁶³⁷³⁸³⁹⁴⁰⁴¹⁴²⁴³

⁴⁴ 12. The potential outcomes framework is based on the work of Donald Rubin, and others, and is therefore sometimes
⁴⁵ called the **Rubin Causal Model** (see e.g., https://en.wikipedia.org/wiki/Rubin_causal_model). The structural
⁴⁶ 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.

⁴⁷

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: $I[p\|q] +$
15 $\epsilon, \frac{1}{4}, \frac{1}{4}, \frac{1}{4} - \epsilon$ should be close to $I[p\|q]$ for small ϵ , where $q = [\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}]$.
- 16 2. *non-negative*: $I[p\|q] \geq 0$ 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: $I[a, b, c, d\|e, f, g, h] = I[a, d, c, b\|e, h, g, f]$.
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

$$\text{38 } p(S) = \left[\frac{3}{8}, \frac{2}{8}, \frac{2}{8}, \frac{1}{8} \right] \quad (5.1)$$

39 becomes

$$\text{40 } p(C) = \left[\frac{5}{8}, \frac{3}{8} \right] \quad p(\{\clubsuit, \spadesuit\}|C = \blacksquare) = \left[\frac{3}{5}, \frac{2}{5} \right] \quad p(\{\heartsuit, \diamondsuit\}|C = \square) = \left[\frac{2}{3}, \frac{1}{3} \right]. \quad (5.2)$$

41 In the same way we could decompose our uniform distribution q . Obviously, for our measure of
42 information to be of use the magnitude of the update needs to be the same regardless of how we
43

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

$$\begin{aligned} \frac{12}{13} \quad f\left(\sum_{i=1}^n \lambda_i \mathbf{x}_i\right) &\leq \sum_{i=1}^n \lambda_i f(\mathbf{x}_i) \\ \frac{14}{15} \end{aligned} \tag{5.12}$$

¹⁶ where $\lambda_i \geq 0$ and $\sum_{i=1}^n \lambda_i = 1$. This can be proved by induction, where the base case with $n = 2$
¹⁷ follows by definition of convexity.

¹⁸ **Theorem 5.1.1.** (*Information inequality*) $D_{\text{KL}}(p\|q) \geq 0$ with equality iff $p = q$.

¹⁹ *Proof.* We now prove the theorem, following [CT06, p28]. As we noted in the previous section, the
²⁰ KL divergence requires special consideration when $p(x)$ or $q(x) = 0$, the same is true here. Let
²¹ $A = \{x : p(x) > 0\}$ be the support of $p(x)$. Using the convexity of the log function and Jensen's
²² inequality, we have that
²³

$$\begin{aligned} \frac{25}{26} \quad -D_{\text{KL}}(p\|q) &= -\sum_{x \in A} p(x) \log \frac{p(x)}{q(x)} = \sum_{x \in A} p(x) \log \frac{q(x)}{p(x)} \\ \frac{27}{28} \end{aligned} \tag{5.13}$$

$$\begin{aligned} \frac{29}{30} \quad &\leq \log \sum_{x \in A} p(x) \frac{q(x)}{p(x)} = \log \sum_{x \in A} q(x) \\ \frac{31}{32} \end{aligned} \tag{5.14}$$

$$\begin{aligned} \frac{33}{34} \quad &\leq \log \sum_{x \in \mathcal{X}} q(x) = \log 1 = 0 \\ \frac{35}{36} \end{aligned} \tag{5.15}$$

³⁷ Since $\log(x)$ is a strictly concave function ($-\log(x)$ is convex), we have equality in Equation (5.14) iff
³⁸ $p(x) = cq(x)$ for some c that tracks the fraction of the whole space \mathcal{X} contained in A . We have equality
³⁹ 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
⁴⁰ $p(x) = q(x)$ for all x . \square

⁴¹ The non-negativity of KL divergence often feels as though its one of the most useful results in
⁴² Information Theory. It is a good result to keep in your back pocket. Anytime you can rearrange an
⁴³ expression in terms of KL divergence terms, since those are guaranteed to be non-negative, dropping
⁴⁴ them immediately generates a bound.

⁴⁵

⁴⁶ 5.1.2.3 KL divergence is invariant to reparameterizations

⁴⁷ We wanted our measure of information to be invariant to permutations of the labels. The discrete
⁴⁸ form is manifestly permutation invariant as summations are. The KL divergence actually satisfies a
⁴⁹

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:

$$\begin{aligned} \text{D}_{\text{KL}}(p(x)\|q(x)) &= \int dx p(x) \log \frac{p(x)}{q(x)} = \int dy p(y) \log \left(\frac{p(y)}{q(y)} \left| \frac{dy}{dx} \right| \right) = \text{D}_{\text{KL}}(p(y)\|q(y)). \end{aligned} \quad (5.16)$$

6 Because of this reparameterization invariance we can rest assured that when we measure the KL
7 divergence between two distributions we are measuring something about the distributions and not the
8 way we choose to represent the space in which they are defined. We are therefore free to transform
9 our data into a convenient basis of our choosing, such as a Fourier bases for images, without affecting
10 the result.

11 5.1.2.4 Montonicity for uniform distributions

12 Consider updating a probability distribution from a uniform distribution on N elements to a uniform
13 distribution on N' elements. The KL divergence is:

$$\text{D}_{\text{KL}}(p\|q) = \sum_k \frac{1}{N'} \log \frac{\frac{1}{N'}}{\frac{1}{N}} = \log \frac{N}{N'}, \quad (5.17)$$

14 or the log of the ratio of the elements before and after the update. This satisfies our monotonocity
15 requirement.

16 We can interpret this result as follows: Consider finding an element of a sorted array by means of
17 bisection. A well designed yes/no question can cut the search space in half. Measured in bits, the
18 KL divergence tells us how many well designed yes/no questions are required on average to move
19 from q to p .

20 5.1.2.5 Chain rule for KL divergence

21 Here we show that the KL divergence satisfies a natural chain rule:

$$\text{D}_{\text{KL}}(p(x,y)\|q(x,y)) = \int dx dy p(x,y) \log \frac{p(x,y)}{q(x,y)} \quad (5.18)$$

$$= \int dx dy p(x,y) \left[\log \frac{p(x)}{q(x)} + \log \frac{p(y|x)}{q(y|x)} \right] \quad (5.19)$$

$$= \text{D}_{\text{KL}}(p(x)\|q(x)) + \mathbb{E}_{p(x)} [\text{D}_{\text{KL}}(p(y|x)\|q(y|x))]. \quad (5.20)$$

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:

$$\text{D}_{\text{KL}}(p(y|x)\|q(y|x)) \triangleq \int dx p(x) \int dy p(y|x) \log \frac{p(y|x)}{q(y|x)}. \quad (5.21)$$

46 This allows us to drop many expectation symbols.

¹ **5.1.3 Thinking about KL**

³ In this section, we discuss some qualitative properties of the KL divergence.

⁵ **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

$$\text{16} \quad 1 \text{ bit} = \log 2 \text{ nats} \sim 0.693 \text{ nats} \quad (5.22)$$

$$\text{17} \quad 1 \text{ nat} = \frac{1}{\log 2} \text{ bits} \sim 1.44 \text{ bits.} \quad (5.23)$$

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:

$$\text{31} \quad \text{KL} = 0.975 \log \frac{0.975}{0.443} + 0.025 \log \frac{0.025}{0.557} = 0.692 \text{ nats} \sim 1.0 \text{ bits.} \quad (5.24)$$

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?

$$\text{36} \quad \text{KL} = 0.443 \log \frac{0.443}{0.975} + 0.557 \log \frac{0.557}{0.025} = 1.38 \text{ nats} \sim 2.0 \text{ bits,} \quad (5.25)$$

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

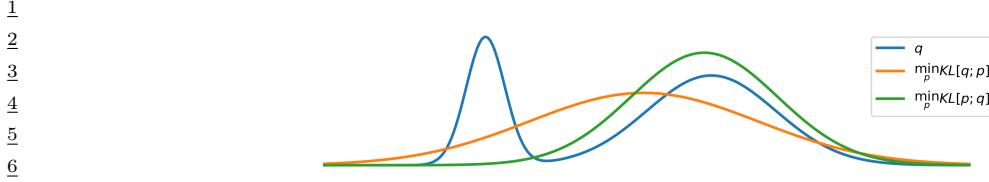


Figure 5.1: Demonstration of the mode-covering or mode-seeking behavior of KL divergence. The original distribution q is bimodal. When we minimize $D_{\text{KL}}(q||p)$, then p covers the modes of q (orange). When we minimize $D_{\text{KL}}(p||q)$, then p ignores some of the modes of q (green).

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.

6 **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:

19
$$g(x) = \frac{q(x)}{\mathcal{Z}} e^{\phi(x)}. \quad (5.31)$$

22 where the *partition function* is given by:

24
$$\mathcal{Z} = \int dx q(x) e^{\phi(x)}. \quad (5.32)$$

26 Taking the KL divergence between $p(x)$ and $g(x)$ and rearranging gives the bound:

29
$$D_{\text{KL}}(P\|G) = D_{\text{KL}}(P\|Q) - \mathbb{E}_P [\phi(x)] + \log(\mathcal{Z}) \geq 0. \quad (5.33)$$

31 \square

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:

36
$$D_{\text{KL}}(P\|Q) = \sup_{\phi} \mathbb{E}_P [\phi(x)] - \log \mathbb{E}_Q [e^{\phi(x)}]. \quad (5.34)$$

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

5.1.4.2 Data processing inequality for KL

We now show that any processing we do on samples from two different distributions makes their samples approach one another. This is called the **data processing inequality**, since it shows that we cannot increase the information gain from q to p by processing our data and then measuring it.

Theorem 5.1.3. Consider two different distributions $p(x)$ and $q(x)$ combined with a probabilistic channel $t(y|x)$. If $p(y)$ is the distribution that results from sending samples from $p(x)$ through the channel $t(y|x)$ and similarly for $q(y)$ we have that:

$$D_{\text{KL}}(p(x)\|q(x)) \geq D_{\text{KL}}(p(y)\|q(y)) \quad (5.35)$$

Proof. The proof uses Jensen's inequality from Section 5.1.2.2 again. Call $p(x, y) = p(x)t(y|x)$ and $q(x, y) = q(x)t(y|x)$.

$$D_{\text{KL}}(p(x)\|q(x)) = \int dx p(x) \log \frac{p(x)}{q(x)} \quad (5.36)$$

$$= \int dx \int dy p(x)t(y|x) \log \frac{p(x)t(y|x)}{q(x)t(y|x)} \quad (5.37)$$

$$= \int dx \int dy p(x, y) \log \frac{p(x, y)}{q(x, y)} \quad (5.38)$$

$$= - \int dy p(y) \int dx p(x|y) \log \frac{q(x, y)}{p(x, y)} \quad (5.39)$$

$$\geq - \int dy p(y) \log \left(\int dx p(x|y) \frac{q(x, y)}{p(x, y)} \right) \quad (5.40)$$

$$= - \int dy p(y) \log \left(\frac{q(y)}{p(y)} \int dx q(x|y) \right) \quad (5.41)$$

$$= \int dy p(y) \log \frac{p(y)}{q(y)} = D_{\text{KL}}(p(y)\|q(y)) \quad (5.42)$$

□

One way to interpret this result is that any processing done to random samples makes it harder to tell two distributions apart.

As a special form of processing, we can simply marginalize out a subset of random variables.

Corollary 5.1.1. (*Monotonicity of KL divergence*)

$$D_{\text{KL}}(p(x, y)\|q(x, y)) \geq D_{\text{KL}}(p(x)\|q(x)) \quad (5.43)$$

1 2 *Proof.* The proof is essentially the same as the one above.

3 4 $D_{\text{KL}}(p(x,y)\|q(x,y)) = \int dx \int dy p(x,y) \log \frac{p(x,y)}{q(x,y)}$ (5.44)

5 6 $= - \int dy p(y) \int dx p(x|y) \log \left(\frac{q(y)}{p(y)} \frac{q(x|y)}{p(x|y)} \right)$ (5.45)

7 8 $\geq - \int dy p(y) \log \left(\frac{q(y)}{p(y)} \int dx q(x|y) \right)$ (5.46)

9 10 $= \int dy p(y) \log \frac{p(y)}{q(y)} = D_{\text{KL}}(p(y)\|q(y))$ (5.47)

11 12 (5.48)

13 14 \square

15 One intuitive interpretation of this result is that if you only partially observe random variables, it
16 is harder to distinguish between two candidate distributions than if you observed all of them.
17

18 19 5.1.5 KL divergence and MLE

20 Suppose we want to find the distribution q that is as close as possible to p , as measured by KL
21 divergence:

22 $q^* = \arg \min_q D_{\text{KL}}(p\|q) = \arg \min_q \int p(x) \log p(x) dx - \int p(x) \log q(x) dx$ (5.49)

23 Now suppose p is the empirical distribution, which puts a probability atom on the observed training
24 data and zero mass everywhere else:

25 $p_{\mathcal{D}}(x) = \frac{1}{N} \sum_{n=1}^N \delta(x - x_n)$ (5.50)

26 Using the sifting property of delta functions we get

27 $D_{\text{KL}}(p_{\mathcal{D}}\|q) = - \int p_{\mathcal{D}}(x) \log q(x) dx + C$ (5.51)

28 $= - \int \left[\frac{1}{N} \sum_n \delta(x - x_n) \right] \log q(x) dx + C$ (5.52)

29 $= - \frac{1}{N} \sum_n \log q(x_n) + C$ (5.53)

30 where $C = \int p_{\mathcal{D}}(x) \log p_{\mathcal{D}}(x)$ is a constant independent of q .

31 We can rewrite the above as follows

32 $D_{\text{KL}}(p_{\mathcal{D}}\|q) = \mathbb{H}(p_{\mathcal{D}}, q) - \mathbb{H}(p_{\mathcal{D}})$ (5.54)

1 where

2

$$\mathbb{H}(p, q) \triangleq - \sum_k p_k \log q_k \quad (5.55)$$

3

4 is known as the **cross entropy**. The quantity $\mathbb{H}(p_D, q)$ is the average negative log likelihood of q
5 evaluated on the training set. Thus we see that minimizing KL divergence to the empirical distribution
6 is equivalent to maximizing likelihood.

7 This perspective points out the flaw with likelihood-based training, namely that it puts too
8 much weight on the training set. In most applications, we do not really believe that the empirical
9 distribution is a good representation of the true distribution, since it just puts “spikes” on a finite
10 set of points, and zero density everywhere else. Even if the dataset is large (say 1M images), the
11 universe from which the data is sampled is usually even larger (e.g., the set of “all natural images”)
12 is much larger than 1M). Thus we need to somehow smooth the empirical distribution by sharing
13 probability mass between “similar” inputs.

17 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

$$p(\theta, D) = \min D_{\text{KL}}(p(\theta, D) \| q(\theta, D)) \text{ such that } p(D) = \delta(D - D_0). \quad (5.56)$$

28 where $\delta(D - D_0)$ is a degenerate distribution that puts all its mass on the dataset D that is identically
29 equal to D_0 . Writing the KL out in its chain rule form:

31 $D_{\text{KL}}(p(\theta, D) \| q(\theta, D)) = D_{\text{KL}}(p(D) \| q(D)) + D_{\text{KL}}(p(\theta|D) \| q(\theta|D)), \quad (5.57)$

33 makes clear that the solution is given by the joint distribution:

35 $p(\theta, D) = p(D)p(\theta|D) = \delta(D - D_0)q(\theta|D). \quad (5.58)$

37 Our updated beliefs have a marginal over the θ

39 $p(\theta) = \int dD p(\theta, D) = \int dD \delta(D - D_0)q(\theta|D) = q(\theta|D = D_0), \quad (5.59)$

41 which is just the usual Bayesian posterior from our prior beliefs evaluated at the data we observed.

42 By contrast, the usual statement of Bayes’ rule is just a trivial observation about the chain rule of
43 probabilities:

45 $q(\theta, D) = q(D)q(\theta|D) = q(\theta)q(D|\theta) \implies q(\theta|D) = \frac{q(D|\theta)}{q(D)}q(\theta). \quad (5.60)$

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:

$$\frac{15}{16} p(\theta) = \int dD p(D)q(\theta|D). \quad (5.61)$$

¹⁷ This generalization of Bayes' rule is sometimes called **Jeffrey's conditionalization rule** [Cat08].

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 $\mathcal{T}(\mathbf{x})$:

$$\frac{25}{26} p(\mathbf{x}) = h(\mathbf{x}) \exp[\boldsymbol{\eta}^\top \mathcal{T}(\mathbf{x}) - A(\boldsymbol{\eta})] \quad (5.62)$$

²⁷ where

$$\frac{29}{30} A(\boldsymbol{\eta}) = \log \int h(\mathbf{x}) \exp(\boldsymbol{\eta}^\top \mathcal{T}(\mathbf{x})) d\mathbf{x} \quad (5.63)$$

³¹ is the *log partition function*, a convex function of $\boldsymbol{\eta}$.

³² The KL divergence between two exponential family distributions from the same family is as follows:

$$\frac{34}{35} D_{\text{KL}}(p(\mathbf{x}|\boldsymbol{\eta}_1) \| p(\mathbf{x}|\boldsymbol{\eta}_2)) = \mathbb{E}_{\boldsymbol{\eta}_1}[(\boldsymbol{\eta}_1 - \boldsymbol{\eta}_2)^\top \mathcal{T}(\mathbf{x}) - A(\boldsymbol{\eta}_1) + A(\boldsymbol{\eta}_2)] \quad (5.64)$$

$$= (\boldsymbol{\eta}_1 - \boldsymbol{\eta}_2)^\top \boldsymbol{\mu}_1 - A(\boldsymbol{\eta}_1) + A(\boldsymbol{\eta}_2) \quad (5.65)$$

³⁷ 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

$$\begin{aligned} \frac{43}{44} D_{\text{KL}}(\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) \| \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)) \\ = \frac{1}{2} \left[\text{tr}(\boldsymbol{\Sigma}_2^{-1} \boldsymbol{\Sigma}_1) + (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^\top \boldsymbol{\Sigma}_2^{-1} (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) - D + \log \left(\frac{\det(\boldsymbol{\Sigma}_2)}{\det(\boldsymbol{\Sigma}_1)} \right) \right] \end{aligned} \quad (5.66)$$

In the scalar case, this becomes

$$D_{\text{KL}}(\mathcal{N}(x|\mu_1, \sigma_1) \| \mathcal{N}(x|\mu_2, \sigma_2)) = \log \frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2} \quad (5.67)$$

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:

$$B_f(\boldsymbol{\eta}_q || \boldsymbol{\eta}_p) = A(\boldsymbol{\eta}_q) - A(\boldsymbol{\eta}_p) - (\boldsymbol{\eta}_q - \boldsymbol{\eta}_p)^\top \nabla_{\boldsymbol{\eta}_p} A(\boldsymbol{\eta}_p) \quad (5.68)$$

$$= A(\boldsymbol{\eta}_q) - A(\boldsymbol{\eta}_p) - (\boldsymbol{\eta}_q - \boldsymbol{\eta}_p)^\top \mathbb{E}_p [\mathcal{T}(\mathbf{x})] \quad (5.69)$$

$$= D_{\text{KL}}(p || q) \quad (5.70)$$

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:

$$\mathbb{H}(X) = \log K - D_{\text{KL}}(p(X) \| u(X)) \quad (5.72)$$

$$D_{\text{KL}}(p(X) \| u(X)) = \sum_{k=1}^K p(X=k) \log \frac{p(X=k)}{\frac{1}{K}} \quad (5.73)$$

$$= \log K + \sum_{k=1}^K p(X=k) \log p(X=k) \quad (5.74)$$

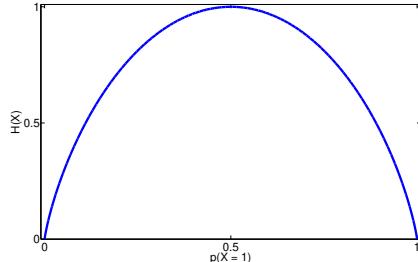
If p is uniform, the KL is zero, and we see that the entropy achieves its maximal value of $\log K$.

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

$$\mathbb{H}(X) = -[p(X=1) \log_2 p(X=1) + p(X=0) \log_2 p(X=0)] \quad (5.75)$$

$$= -[\theta \log_2 \theta + (1 - \theta) \log_2 (1 - \theta)] \quad (5.76)$$

1
2
3
4
5
6
7
8
9
10



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 $H(\theta)$. 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 **5.2.2 Differential entropy for continuous random variables**

20 If X is a continuous random variable with pdf $p(x)$, we define the **differential entropy** as

21

$$22 \quad h(X) \triangleq - \int_{\mathcal{X}} dx p(x) \log p(x) \quad (5.77)$$

23 assuming this integral exists.

24 For example, one can show that the entropy of a d -dimensional Gaussian is

25

$$26 \quad h(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})) = \frac{1}{2} \ln |2\pi e \boldsymbol{\Sigma}| = \frac{1}{2} \ln [(2\pi e)^d |\boldsymbol{\Sigma}|] = \frac{d}{2} + \frac{d}{2} \ln(2\pi) + \frac{1}{2} \ln |\boldsymbol{\Sigma}| \quad (5.78)$$

27 In the 1d case, this becomes

28

$$29 \quad h(\mathcal{N}(\mu, \sigma^2)) = \frac{1}{2} \ln [2\pi e \sigma^2] \quad (5.79)$$

30

31 Note that, unlike the discrete case, *differential entropy can be negative*. This is because pdf's can
 32 be bigger than 1. For example, suppose $X \sim U(0, a)$. Then

33

$$34 \quad h(X) = - \int_0^a dx \frac{1}{a} \log \frac{1}{a} = \log a \quad (5.80)$$

35

36 If we set $a = 1/8$, we have $h(X) = \log_2(1/8) = -3$.

37 One way to understand differential entropy is to realize that all real-valued quantities can only be
 38 represented to finite precision. It can be shown [CT91, p228] that the entropy of an n -bit quantization
 39 of a continuous random variable X is approximately $h(X) + n$. For example, suppose $X \sim U(0, \frac{1}{8})$.
 40 Then in a binary representation of X , the first 3 bits to the right of the binary point must be 0 (since
 41 the number is $\leq 1/8$). So to describe X to n bits of accuracy only requires $n - 3$ bits, which agrees
 42 with $h(X) = -3$ calculated above.

43

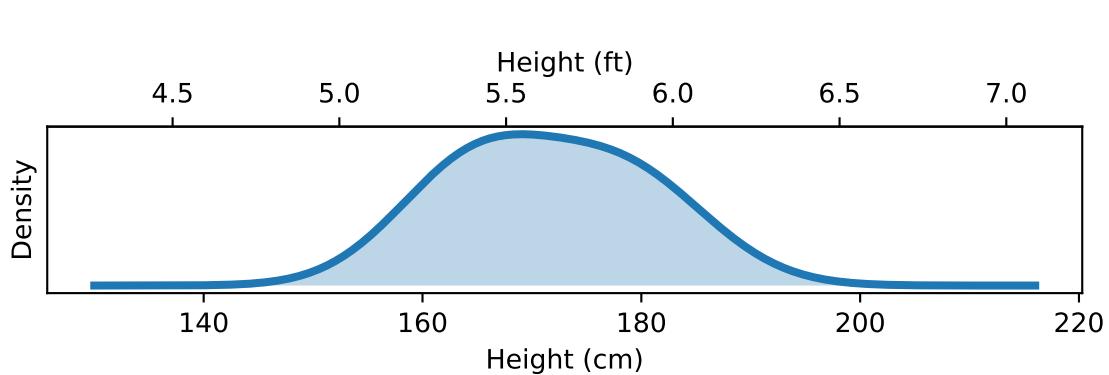


Figure 5.3: Distribution of adult heights. The continuous entropy of the distribution depends on its units of measurement. If heights are measured in feet, this distribution has a continuous entropy of 0.43 bits. If measured in centimeters it's 5.4 bits. If measured in meters it's -1.3 bits. Data taken 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

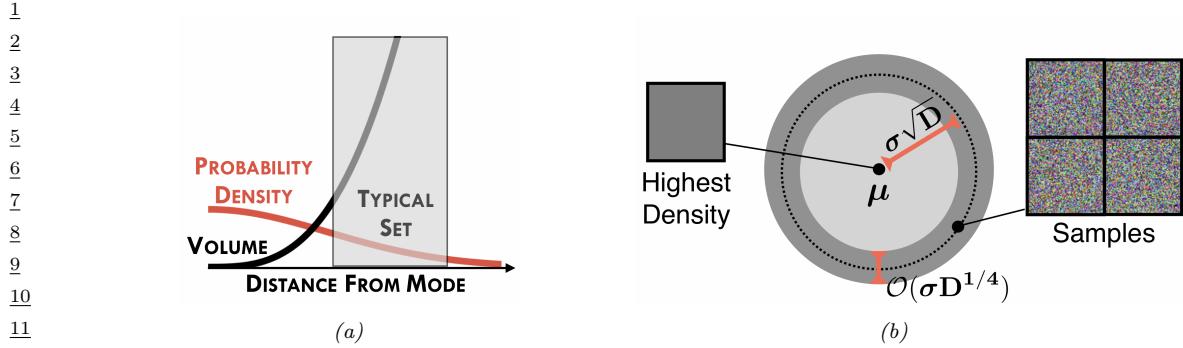


Figure 5.4: (a) Cartoon illustration of why the typical set of a Gaussian is not centered at the mode of the distribution. (b) Illustration of the typical set of a Gaussian, which is concentrated in a thin annulus of thickness $\sigma D^{1/4}$ and distance $\sigma D^{1/2}$ from the origin. We also show 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:
4

$$\lim_{D \rightarrow \infty} \frac{\text{std}[d^2]}{\mathbb{E}[d^2]} = \lim_{D \rightarrow \infty} \frac{\sqrt{D}}{D} = 0 \quad (5.84)$$

5 Thus the expected square distance concentrates around D , so the expected distance concentrates
6 around $\mathbb{E}[d(\mathbf{x})] = \sqrt{D}$. See [Ver18] for a more rigorous proof.
7

8 5.2.4 Cross entropy and perplexity

9 A standard way to measure how close a model q is to a true distribution p is in terms of the KL
10 divergence (Section 5.1), given by
11

$$D_{\text{KL}}(p||q) = \sum_x p(x) \log \frac{p(x)}{q(x)} = \mathbb{H}(p, q) - \mathbb{H}(p) \quad (5.85)$$

12 where $\mathbb{H}(p, q)$ is the **cross entropy**
13

$$\mathbb{H}(p, q) = - \sum_x p(x) \log q(x) \quad (5.86)$$

14 and $\mathbb{H}(p) = \mathbb{H}(p, p)$ is the entropy, which is a constant independent of the model.
15

16 In language modeling, it is common to report an alternative performance measure known as the
17 **perplexity**. This is defined as
18

$$\text{perplexity}(p, q) \triangleq 2^{\mathbb{H}(p, q)} \quad (5.87)$$

19 We can compute an empirical approximation to the cross entropy as follows. Suppose we approxi-
20 mate the true distribution with an empirical distribution based on data sampled from p :
21

$$p_{\mathcal{D}}(x|\mathcal{D}) = \frac{1}{N} \sum_{n=1}^N \mathbb{I}(x = x_n) \quad (5.88)$$

22 In this case, the cross entropy is given by
23

$$H = - \frac{1}{N} \sum_{n=1}^N \log p(x_n) = - \frac{1}{N} \log \prod_{n=1}^N p(x_n) \quad (5.89)$$

24 The corresponding perplexity is given by
25

$$\text{perplexity}(p_{\mathcal{D}}, p) = 2^{-\frac{1}{N} \log(\prod_{n=1}^N p(x_n))} = 2^{\log(\prod_{n=1}^N p(x_n)) - \frac{1}{N}} \quad (5.90)$$

$$= (\prod_{n=1}^N p(x_n))^{-1/N} = \sqrt[N]{\prod_{n=1}^N \frac{1}{p(x_n)}} \quad (5.91)$$

1 In the case of language models, we usually condition on previous words when predicting the next
2 word. For example, in a bigram model, we use a second order Markov model of the form $p(x_n|x_{n-1})$.
3 We define the **branching factor** of a language model as the number of possible words that can
4 follow any given word. For example, suppose the model predicts that each word is equally likely,
5 regardless of context, so $p(x_n|x_{n-1}) = 1/K$, where K is the number of words in the vocabulary. Then
6 the perplexity is $((1/K)^N)^{-1/N} = K$. If some symbols are more likely than others, and the model
7 correctly reflects this, its perplexity will be lower than K . However, we have $\mathbb{H}(p^*) \leq \mathbb{H}(p^*, p)$, so
8 we can never reduce the perplexity below $2^{-\mathbb{H}(p^*)}$.
9

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
$$\mathbb{I}(X; Y) \triangleq D_{\text{KL}}(p(x, y) \| p(x)p(y)) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \quad (5.92)$$

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
$$\mathbb{I}(X; Y) = D_{\text{KL}}(p(x, y) \| p(x)p(y)) \geq 0 \quad (5.93)$$

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
$$\mathbb{I}(X; Y) = \mathbb{H}(X) - \mathbb{H}(X|Y) = \mathbb{H}(Y) - \mathbb{H}(Y|X) \quad (5.94)$$

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

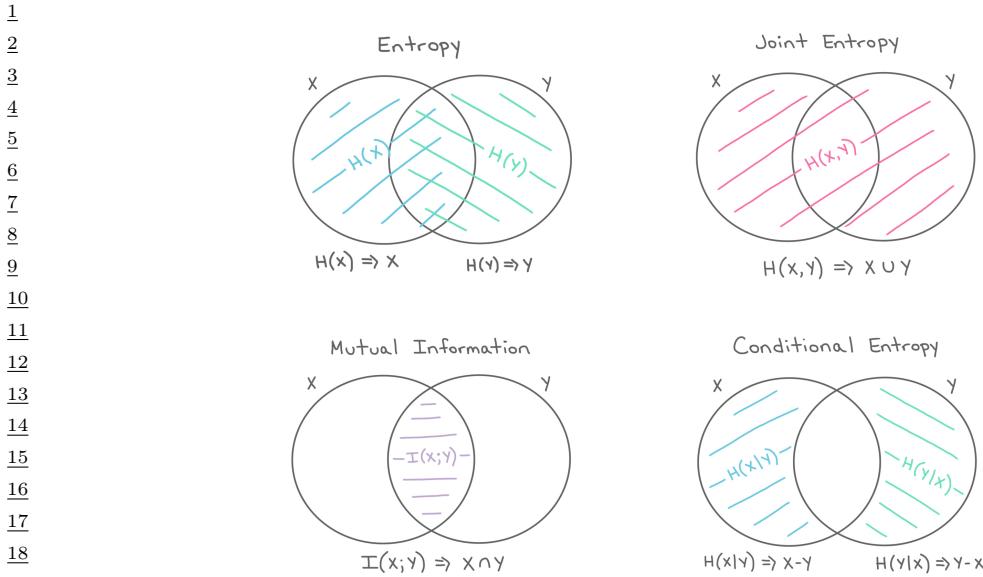


Figure 5.5: The marginal entropy, joint entropy, conditional entropy and mutual information represented as information diagrams. Used with kind permission of Katie Everett.

We can also obtain a different interpretation. One can show that

$$I(X;Y) = H(X,Y) - H(X|Y) - H(Y|X) \quad (5.95)$$

Finally, one can show that

$$I(X;Y) = H(X) + H(Y) - H(X,Y) \quad (5.96)$$

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 $I(X;Y) \geq I(X;Z)$.

Proof. By the chain rule for mutual information we can expand the mutual information in two different ways:

$$I(X;Y,Z) = I(X;Z) + I(X;Y|Z) \quad (5.97)$$

$$= I(X;Y) + I(X;Z|Y) \quad (5.98)$$

1 Since $X \perp Z|Y$, we have $\mathbb{I}(X; Z|Y) = 0$, so

$$\mathbb{I}(X; Z) + \mathbb{I}(X; Y|Z) = \mathbb{I}(X; Y) \quad (5.99)$$

6 Since $\mathbb{I}(X; Y|Z) \geq 0$, we have $\mathbb{I}(X; Y) \geq \mathbb{I}(X; Z)$. Similarly one can prove that $\mathbb{I}(Y; Z) \geq \mathbb{I}(X; Z)$. \square

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

$$\mathbb{I}(\theta; s(X)) \leq \mathbb{I}(\theta; X) \quad (5.100)$$

16 If this holds with equality, then we say that $s(X)$ is a **sufficient statistic** of the data X for the
17 purposes of inferring θ . In this case, we can equivalently write $\theta \rightarrow s(X) \rightarrow X$, since we can
18 reconstruct the data from knowing $s(X)$ just as accurately as from knowing θ .

19 An example of a sufficient statistic is the data itself, $s(X) = X$, but this is not very useful, since it
20 doesn't summarize the data at all. Hence we define a **minimal sufficient statistic** $s(X)$ as one
21 which is sufficient, and which contains no extra information about θ ; thus $s(X)$ maximally compresses
22 the data X without losing information which is relevant to predicting θ . More formally, we say s is a
23 minimal sufficient statistic for X if $s(X) = f(s'(X))$ for some function f and all sufficient statistics
24 $s'(X)$. We can summarize the situation as follows:

$$\theta \rightarrow s(X) \rightarrow s'(X) \rightarrow X \quad (5.101)$$

26 Here $s'(X)$ takes $s(X)$ and adds redundant information to it, thus creating a one-to-many mapping.
27 For example, a minimal sufficient statistic for a set of N Bernoulli trials is simply N and $N_1 =$
28 $\sum_n \mathbb{I}(X_n = 1)$, i.e., the number of successes. In other words, we don't need to keep track of the
29 entire sequence of heads and tails and their ordering, we only need to keep track of the total number
30 of heads and tails. Similarlt, for inferring the mean of a Gaussian distribution with known variance
31 we only need to know the empirical mean and number of samples.

32 Earlier in Section 5.1.7 we motivated the exponential family of distributions as being the ones that
33 are minimal in the sense that they contain no other information than constraints on some statistics of
34 the data. It makes sense then that the statistics used to generate exponential family distributions are
35 sufficient. It also hints at the more remarkable fact of the **Pitman-Koopman-Darmois theorem**,
36 which says that for any distribution whose domain is fixed, it is only the exponential family that
37 admits sufficient statistics with bounded dimensionality as the number of samples increases Diaconis
38 [Dia88].

39

40 5.3.5 Multivariate mutual information

41 There are several ways to generalize the idea of mutual information to a set of random variables as
42 we discuss below.

43

1
2
3
4
5
6
7
8
9
10
11
12
13
14

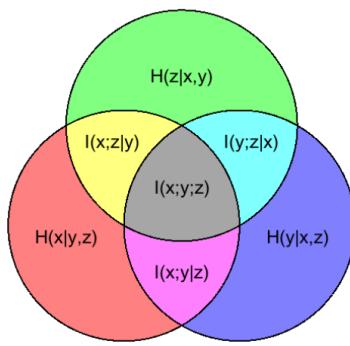


Figure 5.6: Illustration of multivariate mutual information between three random variables. From https://en.wikipedia.org/wiki/Mutual_information. Used with kind permission of Wikipedia author PAR.

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].

1 For 3 variables, the MMI is given by
2

$$\underline{3} \quad \underline{4} \quad \mathbb{I}(X; Y; Z) = \mathbb{I}(X; Y) - \mathbb{I}(X; Y|Z) \quad (5.106)$$

$$\underline{5} \quad = \mathbb{I}(X; Z) - \mathbb{I}(X; Z|Y) \quad (5.107)$$

$$\underline{6} \quad = \mathbb{I}(Y; Z) - \mathbb{I}(Y; Z|X) \quad (5.108)$$

7 This can be interpreted as the change in mutual information between two pairs of variables when
8 conditioning on the third. Note that this quantity is symmetric in its arguments.

9 By the definition of conditional mutual information, we have

$$\underline{10} \quad \underline{11} \quad \mathbb{I}(X; Z|Y) = \mathbb{I}(Z; X, Y) - \mathbb{I}(Y; Z) \quad (5.109)$$

12 Hence we can rewrite Equation (5.107) as follows:

$$\underline{13} \quad \underline{14} \quad \mathbb{I}(X; Y; Z) = \mathbb{I}(X; Z) + \mathbb{I}(Y; Z) - \mathbb{I}(X, Y; Z) \quad (5.110)$$

15 This tells us that the MMI is the difference between how much we learn about Z given X and Y
16 individually vs jointly (see also Section 5.3.5.3).

17 The 3-way MMI is illustrated in the information diagram in Figure 5.6. The way to interpret such
18 diagrams when we have multiple variables is as follows: the area of a shaded area that includes circles
19 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
20 is just $\mathbb{H}(A|F, G, H, \dots)$; if $F = G = H = \emptyset$, this is just $\mathbb{I}(A; B; C, \dots)$.

21 5.3.5.3 Synergy and redundancy

22 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
23 negative. If some of the information about Z that is provided by X is also provided by Y , then
24 there is some **redundancy** between X and Y (wrt Z). In this case, $\mathbb{I}(X; Z) + \mathbb{I}(Y; Z) > \mathbb{I}(X, Y; Z)$,
25 so (from Equation (5.110)) we see that the MMI will be positive. If, by contrast, we learn more
26 about Z when we see X and Y together, we say there is some **synergy** between them. In this case,
27 $\mathbb{I}(X; Z) + \mathbb{I}(Y; Z) < \mathbb{I}(X, Y; Z)$, so the MMI will be negative.

28 5.3.5.4 MMI and causality

29 The sign of the MMI can be used to distinguish between different kinds of directed graphical models,
30 which can sometimes be interpreted causally (see Chapter 38 for a general discussion of causality).
31 For example, consider a model of the form $X \leftarrow Z \rightarrow Y$, where Z is a “cause” of X and Y . For
32 example, suppose X represents the event it is raining, Y represents the event that the sky is dark,
33 and Z represents the event that the sky is cloudy. Conditioning on the common cause Z renders
34 the children X and Y independent, since if I know it is cloudy, noticing that the sky is dark does
35 not change my beliefs about whether it will rain or not. Consequently $\mathbb{I}(X; Y|Z) \leq \mathbb{I}(X; Y)$, so
36 $\mathbb{I}(\{X, Y, Z\}) \geq 0$.

37 Now consider the case where Z is a common effect, $X \rightarrow Z \leftarrow Y$. In this case, conditioning on
38 Z makes X and Y dependent, due to the explaining away phenomenon (see Section 4.2.3.2). For
39 example, if X and Y are independent random bits, and Z is the XOR of X and Y , then observing
40 $Z = 1$ means that $p(X \neq Y|Z = 1) = 1$, so X and Y are now dependent (information-theoretically,
41

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 **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:

7 $\mathbb{I}(\mathbf{x}; \mathbf{y}) = \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} \left[\log \frac{p(\mathbf{y}|\mathbf{x})q(\mathbf{y})}{p(\mathbf{y})q(\mathbf{y})} \right] \quad (5.120)$

9 $= \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} \left[\log \frac{p(\mathbf{y}|\mathbf{x})}{q(\mathbf{y})} \right] - D_{\text{KL}}(p(\mathbf{y})\|q(\mathbf{y})) \quad (5.121)$

10 $\leq \mathbb{E}_{p(\mathbf{x})} \left[\mathbb{E}_{p(\mathbf{y}|\mathbf{x})} \left[\log \frac{p(\mathbf{y}|\mathbf{x})}{q(\mathbf{y})} \right] \right] \quad (5.122)$

11 $= \mathbb{E}_{p(\mathbf{x})} [D_{\text{KL}}(p(\mathbf{y}|\mathbf{x})\|q(\mathbf{y}))] \quad (5.123)$

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.

22 **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:

28 $\mathbb{I}(\mathbf{x}; \mathbf{y}) = \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} \left[\log \frac{p(\mathbf{x}|\mathbf{y})}{p(\mathbf{x})} \right] \quad (5.124)$

30 $= \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} \left[\log \frac{q(\mathbf{x}|\mathbf{y})}{p(\mathbf{x})} \right] + \mathbb{E}_{p(\mathbf{y})} [D_{\text{KL}}(p(\mathbf{x}|\mathbf{y})\|q(\mathbf{x}|\mathbf{y}))] \quad (5.125)$

32 $\geq \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} \left[\log \frac{q(\mathbf{x}|\mathbf{y})}{p(\mathbf{x})} \right] = \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} [\log q(\mathbf{x}|\mathbf{y})] + h(\mathbf{x}) \quad (5.126)$

36 where $h(\mathbf{x})$ is the differential entropy of \mathbf{x} . This is called the **BA lower bound**, after the authors
37 Barber and Agakov [BA03].

38

39 **5.3.6.3 NWJ lower bound**

40

41 The BA lower bound requires a tractable normalized distribution $q(\mathbf{x}|\mathbf{y})$ that we can evaluate
42 pointwise. If we reparameterize this distribution in a clever way, we can generate a lower bound that
43 does not require a normalized distribution. Let's write:

44

45 $q(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{x})e^{f(\mathbf{x}, \mathbf{y})}}{Z(\mathbf{y})} \quad (5.127)$

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:
3

$$\mathbb{E}_{p(\mathbf{x}, \mathbf{y})} \left[\log \frac{p(\mathbf{x}) e^{f(\mathbf{x}, \mathbf{y})}}{p(\mathbf{x}) Z(\mathbf{y})} \right] = \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} [f(\mathbf{x}, \mathbf{y})] - \mathbb{E}_{p(\mathbf{y})} [Z(\mathbf{y})] \quad (5.128)$$

$$= \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} [f(\mathbf{x}, \mathbf{y})] - \mathbb{E}_{p(\mathbf{y})} \left[\log \mathbb{E}_{p(\mathbf{x})} [e^{f(\mathbf{x}, \mathbf{y})}] \right] \quad (5.129)$$

$$\triangleq I_{DV}(X; Y). \quad (5.130)$$

10 This is the **Donsker Varadhan lower bound** [DV75].
11

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
18

$$\mathbb{I}(X; Y) \geq \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} [f(\mathbf{x}, \mathbf{y})] - e^{-1} \mathbb{E}_{p(\mathbf{y})} Z(\mathbf{y}) \triangleq I_{NWJ}(X; Y) \quad (5.132)$$

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].
23

24 5.3.6.4 InfoNCE lower bound 25

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):
28

$$\mathbb{I}_{\text{NCE}} = \mathbb{E} \left[\frac{1}{K} \sum_{i=1}^K \log \frac{e^{f(\mathbf{x}_i, \mathbf{y}_i)}}{\frac{1}{K} \sum_{j=1}^K e^{f(\mathbf{x}_i, \mathbf{y}_j)}} \right] \quad (5.133)$$

$$= \log K - \mathbb{E} \left[\frac{1}{K} \sum_{i=1}^K \log \left(1 + \sum_{j \neq i}^K e^{f(\mathbf{x}_i, \mathbf{y}_j) - f(\mathbf{x}_i, \mathbf{y}_i)} \right) \right] \quad (5.134)$$

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.)
47

¹ **5.4 Data compression (source coding)**

² **Data compression**, also known as **source coding**, is at the heart of information theory. It is also
³ related to probabilistic machine learning. The reason for this is as follows: if we can model the
⁴ probability of different kinds of data samples, then we can assign short **code words** to the most
⁵ frequently occurring ones, reserving longer encodings for the less frequent ones. This is similar to
⁶ the situation in natural language, where common words (such as “a”, “the”, “and”) are generally
⁷ much shorter than rare words. Thus the ability to compress data requires an ability to discover
⁸ the underlying patterns, and their relative frequencies, in the data. This has led Marcus Hutter
⁹ to propose that compression be used as an objective way to measure performance towards general
¹⁰ purpose AI. More precisely, he is offering 50,000 Euros to anyone who can compress the first 100MB
¹¹ of (English) Wikipedia better than some baseline. This is known as the **Hutter prize**.²

¹² In this section, we give a brief summary of some of the key ideas in data compression. For details,
¹³ see e.g., [Mac03; CT06; YMT22].

¹⁴ **5.4.1 Lossless compression**

¹⁵ Discrete data, such as natural language, can always be compressed in such a way that we can uniquely
¹⁶ recover the original data. This is called **lossless compression**.

¹⁷ Claude Shannon proved that the expected number of bits needed to losslessly encode some data
¹⁸ coming from distribution p is at least $\mathbb{H}(p)$. This is known as the **source coding theorem**. Achieving
¹⁹ this lower bound requires coming up with good probability models, as well as good ways to design
²⁰ codes based on those models. Because of the non-negativity of the KL divergence, $\mathbb{H}(p, q) \geq \mathbb{H}(p)$, so
²¹ if we use any model q other than the true model p to compress the data, it will take some excess bits.
²² The number of excess bits is exactly $D_{\text{KL}}(p||q)$.

²³ Common techniques for realizing lossless codes include Huffman coding, arithmetic coding and
²⁴ asymmetric numeral systems [Dud13]. The input to these algorithms is a probability distribution
²⁵ over strings (which is where ML comes in). This distribution is often represented using a latent
²⁶ variable model (see e.g., [TBB19; KAH19]).

²⁷ **5.4.2 Lossy compression and the rate-distortion tradeoff**

²⁸ To encode real-valued signals, such as images and sound, as a digital signal, we first have to quantize
²⁹ the signal into a sequence of symbols. A simple way to do this is to use vector quantization. We can
³⁰ then compress this discrete sequence of symbols using lossless coding methods. However, when we
³¹ uncompress, we lose some information. Hence this approach is called **lossy compression**.

³² In this section, we quantify this tradeoff between the size of the representation (number of symbols
³³ we use), and the resulting error. We will use the terminology of the variational information bottleneck
³⁴ discussed in Section 5.6.2 (except here we are in the unsupervised setting). In particular, we assume
³⁵ we have a stochastic encoder $p(\mathbf{z}|\mathbf{x})$, a stochastic decoder $d(\mathbf{x}|\mathbf{z})$ and a prior marginal $m(\mathbf{z})$.

³⁶ 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>.

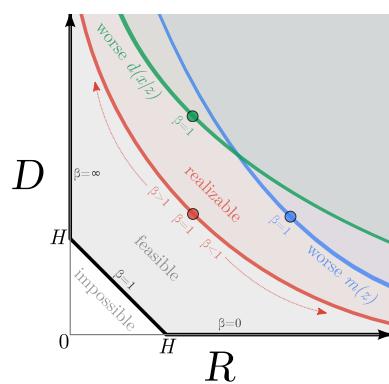


Figure 5.7: Illustration of the rate-distortion tradeoff. See text for details. From Figure 1 of [Ale+18]. Used with kind permission of Alex Alemi.

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 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
30 the VAE objective in Section 22.2. To see this, note that the ELBO from Section 10.1.2 can be
31 written as

32
33
$$\mathcal{L} = -(D + R) = \mathbb{E}_{p(\mathbf{x})} \left[\mathbb{E}_{e(\mathbf{z}|\mathbf{x})} [\log d(\mathbf{x}|\mathbf{z})] - \mathbb{E}_{e(\mathbf{z}|\mathbf{x})} \left[\log \frac{e(\mathbf{z}|\mathbf{x})}{m(\mathbf{z})} \right] \right] \quad (5.143)$$

34

35 which we recognize as the expected reconstruction error minus the KL term $D_{\text{KL}}(e(\mathbf{z}|\mathbf{x}) \| m(\mathbf{z}))$.

36 If we allow $\beta \neq 1$, we recover the β -VAE objective discussed in Section 22.3.2. Note, however, that
37 the β -VAE model cannot distinguish between different solutions on the diagonal line, all of which have
38 $\beta = 1$. This is because all such models have the same marginal likelihood (and hence same ELBO),
39 although they differ radically in terms of whether they learn an interesting latent representation or
40 not. Thus likelihood is not a sufficient metric for comparing the quality of unsupervised representation
41 learning methods, as discussed in Section 22.3.2.

42 For further discussion on the inherent conflict between rate, distortion and *perception*, see Blau
43 and Michaeli [BM19]. For techniques for evaluating rate distortion curves for models see Huang, Cao,
44 and Grosse [HCG20].

45

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

$$R(\mathbf{x}) \triangleq \int d\mathbf{z} p(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{z}|\mathbf{x})}{m(\mathbf{z})} = \mathbb{H}(p(\mathbf{z}|\mathbf{x}), m(\mathbf{z})) - \mathbb{H}(p(\mathbf{z}|\mathbf{x})). \quad (5.144)$$

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.

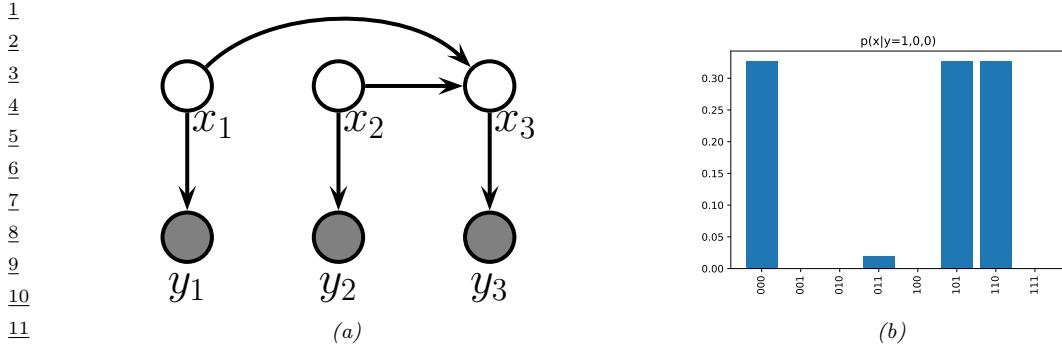


Figure 5.8: (a) A simple error-correcting code PGM-D. x_i are the sent bits, y_i are the received bits. x_3 is an even parity check bit computed from x_1 and x_2 . (b) Posterior over codewords given that $\mathbf{y} = (1, 0, 0)$; the probability of a bit flip is 0.2. Generated by [error_correcting_code_demo.py](#).

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

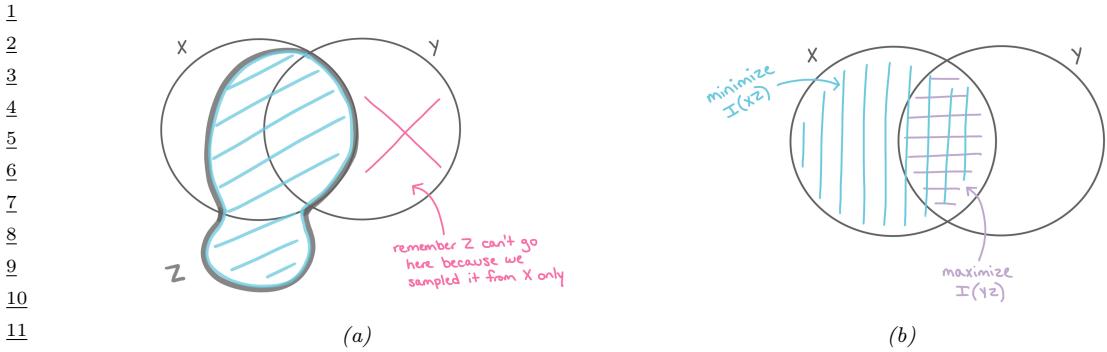


Figure 5.9: Information diagrams for information bottleneck. (a) Z can contain any amount of information about X (whether it useful for predicting Y or not), but it cannot contain information about Y that is not shared with X . (b) The optimal representation for Z maximizes $\mathbb{I}(Z, Y)$ and minimizes $\mathbb{I}(Z, X)$. 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 If all the random variables are discrete, and $\mathbf{z} = e(\mathbf{x})$ is a deterministic function of \mathbf{x} , then the
⁶ algorithm of [TPB99] can be used to minimize the IB objective in Section 5.6. The objective can
⁷ also be solved analytically if all variables are jointly Gaussian [Che+05] (the resulting method can be
⁸ viewed as a form of supervised PCA). But in general, it is intractable to solve this problem exactly.
⁹ We discuss a tractable approximation in Section 5.6.2. (More details can be found in e.g., [SZ22].)

¹⁰

¹¹ 5.6.2 Variational IB

¹² In this section, we derive a variational upper bound on Equation (5.147), leveraging ideas from
¹³ Section 5.3.6. This is called the **variational IB** or **VIB** method [Ale+16]. The key trick will be to
¹⁴ use the non-negativity of the KL divergence to write

$$\int d\mathbf{x} p(\mathbf{x}) \log p(\mathbf{x}) \leq \int d\mathbf{x} p(\mathbf{x}) \log q(\mathbf{x}) \quad (5.149)$$

¹⁵ for any distribution q . (Note that both p and q may be conditioned on other variables.)

¹⁶ To explain the method in more detail, let us define the following notation. Let $e(\mathbf{z}|\mathbf{x}) = p(\mathbf{z}|\mathbf{x})$
¹⁷ 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
¹⁸ the classifier (decoder), and $m(\mathbf{z}) \approx p(\mathbf{z})$ represent the marginal. (Note that we get to choose
¹⁹ $p(\mathbf{z}|\mathbf{x})$, but the other distributions are derived by approximations of the corresponding marginals
²⁰ and conditionals of the exact joint $p(\mathbf{x}, \mathbf{y}, \mathbf{z})$.) Also, let $\langle \cdot \rangle$ represent expectations wrt the relevant
²¹ terms from the $p(\mathbf{x}, \mathbf{y}, \mathbf{z})$ joint.

²² With this notation, we can derive a lower bound on $\mathbb{I}(\mathbf{z}; \mathbf{y})$ as follows:

$$\mathbb{I}(\mathbf{z}; \mathbf{y}) = \int d\mathbf{y} d\mathbf{z} p(\mathbf{y}, \mathbf{z}) \log \frac{p(\mathbf{y}, \mathbf{z})}{p(\mathbf{y})p(\mathbf{z})} \quad (5.150)$$

$$= \int d\mathbf{y} d\mathbf{z} p(\mathbf{y}, \mathbf{z}) \log p(\mathbf{y}|\mathbf{z}) - \int d\mathbf{y} d\mathbf{z} p(\mathbf{y}, \mathbf{z}) \log p(\mathbf{y}) \quad (5.151)$$

$$= \int d\mathbf{y} d\mathbf{z} p(\mathbf{z})p(\mathbf{y}|\mathbf{z}) \log p(\mathbf{y}|\mathbf{z}) - \text{const} \quad (5.152)$$

$$\geq \int d\mathbf{y} d\mathbf{z} p(\mathbf{y}, \mathbf{z}) \log d(\mathbf{y}|\mathbf{z}) \quad (5.153)$$

$$= \langle \log d(\mathbf{y}|\mathbf{z}) \rangle \quad (5.154)$$

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

$$p(\mathbf{y}, \mathbf{z}) = \int d\mathbf{x} p(\mathbf{x})p(\mathbf{y}|\mathbf{x})p(\mathbf{z}|\mathbf{x}) = \int d\mathbf{x} p(\mathbf{x}, \mathbf{y})e(\mathbf{z}|\mathbf{x}) \quad (5.155)$$

⁴³

⁴⁴ This is just the empirical distribution “pushed through” the encoder.

⁴⁵

1 Similarly, we can derive an upper bound on $\mathbb{I}(\mathbf{z}; \mathbf{x})$ as follows:

$$\mathbb{I}(\mathbf{z}; \mathbf{x}) = \int d\mathbf{z} d\mathbf{x} p(\mathbf{x}, \mathbf{z}) \log \frac{p(\mathbf{z}, \mathbf{x})}{p(\mathbf{x})p(\mathbf{z})} \quad (5.156)$$

$$= \int d\mathbf{z} d\mathbf{x} p(\mathbf{x}, \mathbf{z}) \log p(\mathbf{z}|\mathbf{x}) - \int d\mathbf{z} p(\mathbf{z}) \log p(\mathbf{z}) \quad (5.157)$$

$$\leq \int d\mathbf{z} d\mathbf{x} p(\mathbf{x}, \mathbf{z}) \log p(\mathbf{z}|\mathbf{x}) - \int d\mathbf{z} p(\mathbf{z}) \log m(\mathbf{z}) \quad (5.158)$$

$$= \int d\mathbf{z} d\mathbf{x} p(\mathbf{x}, \mathbf{z}) \log \frac{e(\mathbf{z}|\mathbf{x})}{m(\mathbf{z})} \quad (5.159)$$

$$= \langle \log e(\mathbf{z}|\mathbf{x}) \rangle - \langle m(\mathbf{z}) \rangle \quad (5.160)$$

14 Note that we can approximate the expectations by sampling from $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x})p(\mathbf{z}|\mathbf{x})$.

15 Putting it altogether, we get the following upper bound on the IB objective:

$$\beta \mathbb{I}(\mathbf{x}; \mathbf{z}) - \mathbb{I}(\mathbf{z}; \mathbf{y}) \leq \beta (\langle \log e(\mathbf{z}|\mathbf{x}) \rangle - \langle \log m(\mathbf{z}) \rangle) - \langle \log d(\mathbf{y}|\mathbf{z}) \rangle \quad (5.161)$$

18 Thus the VIB objective is

$$\mathcal{L}_{\text{VIB}} = \beta (\mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})e(\mathbf{z}|\mathbf{x})} [\log e(\mathbf{z}|\mathbf{x}) - \log m(\mathbf{z})]) - \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})e(\mathbf{z}|\mathbf{x})d(\mathbf{y}|\mathbf{z})} [\log d(\mathbf{y}|\mathbf{z})] \quad (5.162)$$

$$= -\mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})e(\mathbf{z}|\mathbf{x})d(\mathbf{y}|\mathbf{z})} [\log d(\mathbf{y}|\mathbf{z})] + \beta \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [D_{\text{KL}}(e(\mathbf{z}|\mathbf{x}) \| m(\mathbf{z}))] \quad (5.163)$$

22 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).

30 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 31 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 33 image gets mapped to a Gaussian distribution. The classes are still well separated, but individual 34 instances of a class are no longer distinguishable, since such information is not relevant for prediction 35 purposes.

37 5.6.3 Conditional entropy bottleneck

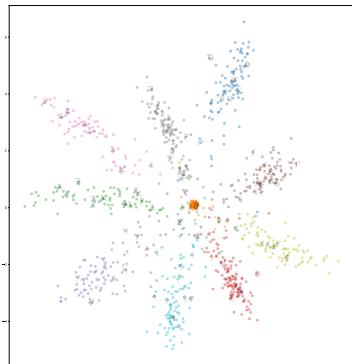
39 The IB tries to maximize $\mathbb{I}(Z; Y)$ while minimizing $\mathbb{I}(Z; X)$. We can write this objective as

$$\min \mathbb{I}(\mathbf{x}; \mathbf{z}) - \lambda \mathbb{I}(\mathbf{y}; \mathbf{z}) \quad (5.164)$$

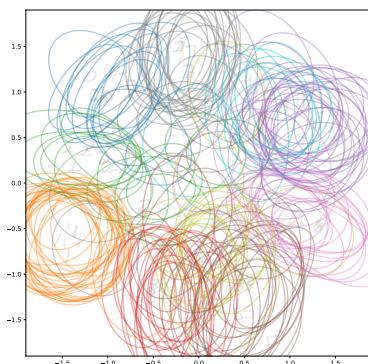
42 for $\lambda \geq 0$. However, we see from the information diagram in Figure 5.9b that $\mathbb{I}(Z; X)$ contains some 43 information that is relevant to Y . A sensible alternative objective is to minimizes the residual mutual 44 information, $\mathbb{I}(X; Z|Y)$. This gives rise to the following objective:

$$\min \mathbb{I}(\mathbf{x}; \mathbf{z}|\mathbf{y}) - \lambda' \mathbb{I}(\mathbf{y}; \mathbf{z}) \quad (5.165)$$

1
2
3
4
5
6
7
8
9
10
11
12
13



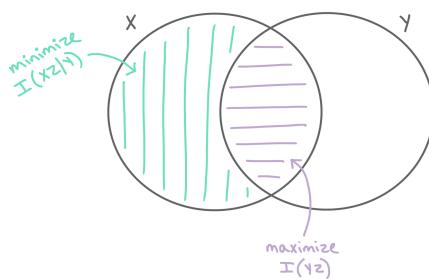
(a)



(b)

16 *Figure 5.10: 2d embeddings of MNIST digits created by an MLP classifier. (a) Deterministic model. (b)
17 Stochastic VIB model. Generated by [VIBDemo2021.ipynb](#). Used with kind permission of Alex Alemi.*

18
19
20
21
22
23
24
25
26
27
28
29



30 *Figure 5.11: Conditional entropy bottleneck (CEB) chooses a representation Z that maximizes $I(Z, Y)$ and
31 minimizes $I(X, Z|Y)$. Used with kind permission of Katie Everett.*

32
33
34
35
36
37
38
39
40
41
42

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 $I(\mathbf{x}; \mathbf{z}|\mathbf{y}) = I(\mathbf{x}; \mathbf{z}) - I(\mathbf{y}; \mathbf{z})$, we see that the CEB is equivalent to standard IB with $\lambda' = \lambda + 1$.
46 However, it is easier to upper bound $I(\mathbf{x}; \mathbf{z}|\mathbf{y})$ than $I(\mathbf{x}; \mathbf{z})$, since we are conditioning on \mathbf{y} , which
47

1 provides information about \mathbf{z} . In particular, we have
2

$$\mathbb{I}(\mathbf{x}; \mathbf{z}|\mathbf{y}) = \mathbb{I}(\mathbf{x}; \mathbf{z}) - \mathbb{I}(\mathbf{y}; \mathbf{z}) \quad (5.166)$$

$$= \mathbb{H}(\mathbf{z}) - \mathbb{H}(\mathbf{z}|\mathbf{x}) - [\mathbb{H}(\mathbf{z}) - \mathbb{H}(\mathbf{z}|\mathbf{y})] \quad (5.167)$$

$$= -\mathbb{H}(\mathbf{z}|\mathbf{x}) - \mathbb{H}(\mathbf{z}|\mathbf{y}) \quad (5.168)$$

$$= \int d\mathbf{z}d\mathbf{x} p(\mathbf{x}, \mathbf{z}) \log p(\mathbf{z}|\mathbf{x}) - \int d\mathbf{z}d\mathbf{y} p(\mathbf{z}, \mathbf{y}) \log p(\mathbf{z}|\mathbf{y}) \quad (5.169)$$

$$\leq \int d\mathbf{z}d\mathbf{x} p(\mathbf{x}, \mathbf{z}) \log e(\mathbf{z}|\mathbf{x}) - \int d\mathbf{z}d\mathbf{y} p(\mathbf{z}, \mathbf{y}) \log b(\mathbf{z}|\mathbf{y}) \quad (5.170)$$

$$= \langle \log e(\mathbf{z}|\mathbf{x}) \rangle - \langle \log b(\mathbf{z}|\mathbf{y}) \rangle \quad (5.171)$$

13 Putting it altogether, we get the final CEB objective:
14

$$\min \beta (\langle \log e(\mathbf{z}|\mathbf{x}) \rangle - \langle \log b(\mathbf{z}|\mathbf{y}) \rangle) - \langle \log d(\mathbf{y}|\mathbf{z}) \rangle \quad (5.172)$$

17 Note that it is generally easier to learn the conditional backwards encoder $b(\mathbf{z}|\mathbf{y})$ than the
18 unconditional marginal $m(\mathbf{z})$. Also, we know that the tightest upper bound occurs when $\mathbb{I}(\mathbf{x}; \mathbf{z}|\mathbf{y}) =$
19 $\mathbb{I}(\mathbf{x}; \mathbf{z}) - \mathbb{I}(\mathbf{y}; \mathbf{z}) = 0$. The corresponding value of β corresponds to an optimal representation. By
20 contrast, it is not clear how to measure distance from optimality when using IB.
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

$$\frac{\partial f}{\partial x_1} \Big|_{\mathbf{x}=\mathbf{a}} \quad (6.2)$$

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

$$\frac{\partial}{\partial a_1} f(a_1, a_2) \quad (6.3)$$

7 where now a_1 serves both as an argument name (or a symbol in an expression) and as a particular
8 evaluation point. Tracking names can become an increasingly complicated endeavor as we compose
9 many functions together, each possibly taking several arguments.
10

11 A functional notation instead defines derivatives as operators on functions. If a function has
12 multiple arguments, they are identified by position rather than by name, alleviating the need for
13 auxiliary variable definitions. Some of the following definitions draw on those in Spivak's *Calculus*
14 on Manifolds [Spi71], in Sussman and Wisdom's *Functional Differential Geometry* [SW13], and
15 generally appear more regularly in accounts of differential calculus and geometry. These texts are
16 recommended for a more formal treatment, and a more mathematically general view, of the material
17 briefly covered in this section.
18

19 Beside notation, we will rely on some basic multivariable calculus concepts. This includes the
20 notion of (partial) derivatives, the differential or Jacobian of a function at a point, its role as a linear
21 approximation local to the point, and various properties of linear maps, matrices, and transposition.
22 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 .
23

24
25 **Linear and multilinear functions.** We use $F : \mathbb{R}^n \multimap \mathbb{R}^m$ to denote a function $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$
26 that is linear, and by $F[\mathbf{x}]$ its application to $\mathbf{x} \in \mathbb{R}^n$. Recall that such a linear map corresponds
27 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.
28 Conveniently, function composition and matrix multiplication expressions look similar: to compose
29 two linear maps F and G we can write $F \circ G$ or, barely abusing notation, consider the matrix FG .
30 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
31 with transposing the corresponding matrix.
32

33 Repeatedly using the linear arrow symbol, we can denote by:
34

$$T : \underbrace{\mathbb{R}^n \multimap \cdots \multimap \mathbb{R}^n}_{k \text{ times}} \multimap \mathbb{R}^m \quad (6.4)$$

35 a multilinear, or more specifically k -linear, map:
36

$$T : \underbrace{\mathbb{R}^n \times \cdots \times \mathbb{R}^n}_{k \text{ times}} \rightarrow \mathbb{R}^m \quad (6.5)$$

37 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
38 application of such a k -linear map to vectors $\mathbf{x}_1, \dots, \mathbf{x}_k \in \mathbb{R}^n$.
39

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 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 As a toy example, consider $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ composed of three operations in sequence:

²⁵
$$f = c \circ b \circ a \tag{6.28}$$

²⁶

²⁷ By the chain rule, its derivatives are given by

²⁸
$$\partial f(\mathbf{x}) = \partial c(b(a(\mathbf{x}))) \circ \partial b(a(\mathbf{x})) \circ \partial a(\mathbf{x}) \tag{6.29}$$

²⁹

³⁰ Now consider the JVP against an input perturbation $\mathbf{v} \in \mathbb{R}^n$:

³¹
$$\partial f(\mathbf{x})[\mathbf{v}] = \partial c(b(a(\mathbf{x}))) [\partial b(a(\mathbf{x})) [\partial a(\mathbf{x})[\mathbf{v}]]] \tag{6.30}$$

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
$$\mathbf{x}, a(\mathbf{x}), b(a(\mathbf{x})) \tag{6.31}$$

³⁷

³⁸ alongside the partial JVPs, because each is then immediately used as a subsequent linearization
³⁹ point, respectively:

⁴⁰
$$\partial a(\mathbf{x}), \partial b(a(\mathbf{x})), \partial c(b(a(\mathbf{x}))) \tag{6.32}$$

⁴¹

⁴² Extending this idea to arbitrary chain compositions gives Algorithm 1.

⁴³ 2. Materializing the Jacobian as a numerical array, as is commonly required in an optimization context, is a special
⁴⁴ case of computing a JVP or VJP against the standard basis vectors in \mathbb{R}^n or \mathbb{R}^m respectively.

⁴⁵

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

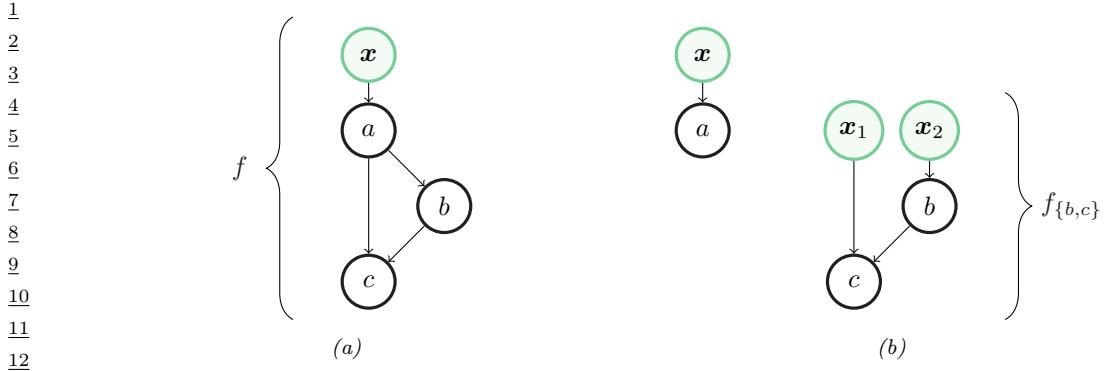
```

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

33 What made AD work in Section 6.2.2.1 is the fact that derivatives decompose along chains thanks
34 to the aptly-named chain rule. When moving from chains to directed acyclic graphs, do we need
35 some sort of “graph rule” in order to decompose our calculation along the circuit’s structure? Circuits
36 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.
37

38 What do these mean in functional terms? Fan-in happens when a primitive operation accepts
39 multiple arguments. We observed in Section 6.2.1 that multiple arguments can be treated as one, and
40 how the chain rule then applies. Fan-out requires slightly more care, specifically for reverse-mode
differentiation.
41

42 The gist of an answer can be illustrated with a small example. Consider the circuit in Figure 6.1a.
43 The operation a precedes b and c topologically, with an outgoing edge to each of both. We can cut a
44 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:
45

$$46 \quad f_{\{b,c\}}(\mathbf{x}_1, \mathbf{x}_2) = c(\mathbf{x}_1, b(\mathbf{x}_2)). \quad (6.35)$$

47

1 We can recover the complete function f from a and $f_{\{b,c\}}$ with the help of a function dup given by:

$$\text{dup}(\mathbf{x}) = (\mathbf{x}, \mathbf{x}) \equiv \begin{pmatrix} I \\ I \end{pmatrix} \mathbf{x} \quad (6.36)$$

6 so that f can be written as a chain composition:

$$\mathbf{f} = f_{\{b,c\}} \circ \text{dup} \circ a. \quad (6.37)$$

10 The circuit for $f_{\{b,c\}}$ contains no fan-out, and composition rules such as Equation (6.25) tell us its
11 derivatives in terms of b , c , and their derivatives, all via the chain rule. Meanwhile, the chain rule
12 applied to Equation (6.37) says that:

$$\partial f(\mathbf{x}) = \partial f_{\{b,c\}}(\text{dup}(a(\mathbf{x}))) \circ \partial \text{dup}(a(\mathbf{x})) \circ \partial a(\mathbf{x}) \quad (6.38)$$

$$= \partial f_{\{b,c\}}(a(\mathbf{x}), a(\mathbf{x})) \circ \begin{pmatrix} I \\ I \end{pmatrix} \circ \partial a(\mathbf{x}). \quad (6.39)$$

17 The above expression suggests calculating a JVP of f by right-to-left evaluation. It is similar to
18 the JVP calculation suggested by Equation (6.30), but with a *duplication* operation $(I \ I)^T$ in the
19 middle that arises from the Jacobian of dup.

21 Transposing the derivative of f at \mathbf{x} :

$$\partial f(\mathbf{x})^T = \partial a(\mathbf{x})^T \circ (I \ I) \circ \partial f_{\{b,c\}}(a(\mathbf{x}), a(\mathbf{x}))^T. \quad (6.40)$$

24 Considering right-to-left evaluation, this too is similar to the VJP calculation suggested by Equation
25 (6.33), but with a *summation* operation $(I \ I)$ in the middle that arises from the *transposed*
26 Jacobian of dup. The lesson of using dup in this small example is that, more generally, in order to
27 handle fan-out in reverse mode AD, we can process operations in topological order—first forward
28 and then in reverse—and then *sum* partial VJPs along multiple outgoing edges.

30 **Algorithm 3:** Foward-mode circuit differentiation (JVP)

```
32 1 input:  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  composing  $f_1, \dots, f_T$  in topological order, where  $f_1$  is identity
33 2 input: linearization point  $\mathbf{x} \in \mathbb{R}^n$  and perturbation  $\mathbf{v} \in \mathbb{R}^n$ 
34 3  $\mathbf{x}_1, \mathbf{v}_1 := \mathbf{x}, \mathbf{v}$ 
35 4 for  $t := 2, \dots, T$  do
36 5   let  $[q_1, \dots, q_r] = \text{parents}(t)$ 
37 6    $\mathbf{x}_t := f_t(\mathbf{x}_{q_1}, \dots, \mathbf{x}_{q_r})$ 
38 7    $\mathbf{v}_t := \sum_{i=1}^r \partial_i f_t(\mathbf{x}_{q_1}, \dots, \mathbf{x}_{q_r})[\mathbf{v}_{q_i}]$ 
39 8 output:  $\mathbf{x}_T$ , equal to  $f(\mathbf{x})$ 
40 9 output:  $\mathbf{v}_T$ , equal to  $\partial f(\mathbf{x})[\mathbf{v}]$ 
```

42 Algorithms 3 and 4 give a complete description of forward- and reverse-mode differentiation on
43 circuits. For brevity they assume a single argument to the entire circuit function. Nodes are indexed
44 $1, \dots, T$. The first is the input node associated and the remaining $T - 1$ are labeled by their operation
45 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
46 f_2, \dots, f_T .

```

1 Algorithm 4: Reverse-mode circuit differentiation (VJP)
2
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
6 3  $\mathbf{x}_1 := \mathbf{x}$ 
7 4 for  $t := 2, \dots, T$  do
8   5 let  $[q_1, \dots, q_r] = \text{parents}(t)$ 
9   6  $\mathbf{x}_t := f_t(\mathbf{x}_{q_1}, \dots, \mathbf{x}_{q_r})$ 
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  $\partial f(\mathbf{x})^\top \mathbf{u}$ 
17

```

20 ordered list of k indices of its parent nodes (possibly containing duplicates, due to fan-out), and let
21 $\text{children}(t)$ 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.

Reverse-mode AD on circuits has appeared under various names and formulations over the years. The algorithm is precisely the **backpropagation** algorithm in neural networks, a term introduced in the 1980s [RHW86b; RHW86a], and has separately come up in the context of control theory and sensitivity, as summarized in historical notes by Goodfellow, Bengio, and Courville [GBC16, Section 6.6].

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

In the previous sections, our approach to AD became more complex as we allowed for more 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 in its syntactic definition; its behavior may rely on side effects or global state, as allowed by the language. Here, we specifically mean a Python function that is pure and functional. JAX’s documentation details this restriction [Bra+18].

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:

```
10  f(x) =
11    let ax = a(x)
12      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.

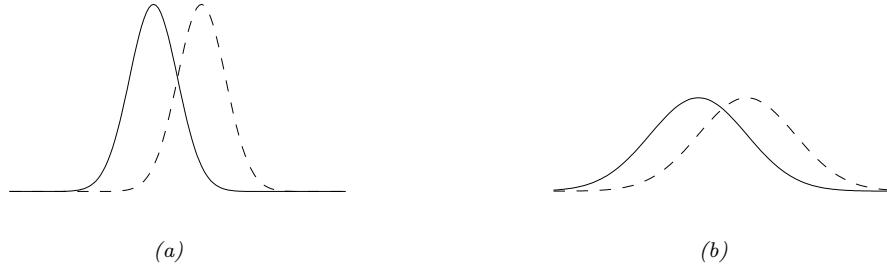
In many problems, we cannot compute the exact gradient, either because the loss is stochastic (e.g., due to random factors in the environment), or because we approximate the loss by randomly subsampling the data. In such cases, we can modify the update in Equation (6.41) to use an unbiased approximation of the gradient. For example, suppose the loss is a finite-sum objective from a supervised learning problem:

$$\mathcal{L}(\theta_t) = \frac{1}{N} \sum_{n=1}^N \ell(y_n, f(x_n; \theta_t)) = \frac{1}{N} \sum_{n=1}^N \mathcal{L}_n(\theta_t) \quad (6.42)$$

We can approximate the gradient using a minibatch \mathcal{B}_t of size $B = |\mathcal{B}_t|$ as follows:

$$\hat{\mathbf{g}}_t = \hat{\nabla} \mathcal{L}(\theta_t) = \frac{1}{B} \sum_{n \in \mathcal{B}_t} \nabla \mathcal{L}_n(\theta_t) \quad (6.43)$$

1
2
3
4
5
6
7
8
9
10
11
12



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.*

17

18

19

20 Since the minibatches are randomly sampled, this is a stochastic, but unbiased, estimate of the
21 gradient. If we insert $\hat{\mathbf{g}}_t$ into Equation (6.41), the method is called **stochastic gradient descent**
22 or **SGD**.

23

24 6.4 Natural gradient descent

25 In this section, we discuss **natural gradient descent (NGD)** [Ama98], which is a second order
26 method for optimizing the parameters of (conditional) probability distributions $p_{\theta}(\mathbf{y}|\mathbf{x})$. The key
27 idea is to compute parameter updates by measuring distances between the induced distributions,
28 rather than comparing parameter values directly.

29 For example, consider comparing two Gaussians, $p_{\theta} = p(y|\mu, \sigma)$ and $p_{\theta'} = p(y|\mu', \sigma')$. The (squared)
30 Euclidean distance between the parameter vectors decomposes as $\|\theta - \theta'\|^2 = (\mu - \mu')^2 + (\sigma - \sigma')^2$.
31 However, the predictive distribution has the form $\exp(-\frac{1}{2\sigma^2}(y - \mu)^2)$, so changes in μ need to be
32 measured relative to σ . This is illustrated in Figure 6.2(a-b), which shows two univariate Gaussian
33 distributions (dotted and solid lines) whose means differ by δ . In Figure 6.2(a), they share the same
34 small variance σ^2 , whereas in Figure 6.2(b), they share the same large variance. It is clear that
35 the value of δ matters much more (in terms of the effect on the distribution) when the variance is
36 small. Thus we see that the two parameters interact with each other, which the Euclidean distance
37 cannot capture. This problem gets much worse when we consider more complex models, such as deep
38 neural networks. By modeling such correlations, NGD can converge much faster than other gradient
39 methods.

40

41

42 6.4.1 Defining the natural gradient

43

44 The key to NGD is to measure the notion of distance between two probability distributions in terms
45 of the KL divergence. As we show in Section 2.6.4, this can be approximated in terms of the Fisher
46

47

1 information matrix (FIM). In particular, for any given input \mathbf{x} , we have
2

$$\small{3} \quad D_{\text{KL}}(p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}) \| p_{\boldsymbol{\theta}+\boldsymbol{\delta}}(\mathbf{y}|\mathbf{x})) \approx \frac{1}{2} \boldsymbol{\delta}^T \mathbf{F}_{\mathbf{x}} \boldsymbol{\delta} \quad (6.44)$$

4 where $\mathbf{F}_{\mathbf{x}}$ is the FIM
5

$$\small{6} \quad \mathbf{F}_{\mathbf{x}}(\boldsymbol{\theta}) = -\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})} [\nabla^2 \log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})] = \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})} [(\nabla \log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})) (\nabla \log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}))^T] \quad (6.45)$$

7 We can compute the average KL between the current and updated distributions using $\frac{1}{2} \boldsymbol{\delta}^T \mathbf{F} \boldsymbol{\delta}$, where
8 \mathbf{F} is the averaged FIM:
9

$$\small{10} \quad \mathbf{F}(\boldsymbol{\theta}) = \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [\mathbf{F}_{\mathbf{x}}(\boldsymbol{\theta})] \quad (6.46)$$

11 NGD uses the inverse FIM as a preconditioning matrix, i.e., we perform updates of the following
12 form:
13

$$\small{14} \quad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \mathbf{F}(\boldsymbol{\theta}_t)^{-1} \mathbf{g}_t \quad (6.47)$$

15 The term
16

$$\small{17} \quad \mathbf{F}^{-1} \mathbf{g}_t = \mathbf{F}^{-1} \nabla \mathcal{L}(\boldsymbol{\theta}_t) \triangleq \tilde{\nabla} \mathcal{L}(\boldsymbol{\theta}_t) \quad (6.48)$$

18 is called the **natural gradient**.
19

20 6.4.2 Interpretations of NGD

21 6.4.2.1 NGD as a trust region method

22 In Section 6.5.2 we show that we can interpret standard gradient descent as optimizing a linear
23 approximation to the objective subject to a penalty on the ℓ_2 norm of the change in parameters, i.e.,
24 if $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \boldsymbol{\delta}$, then we optimize
25

$$\small{26} \quad M_t(\boldsymbol{\delta}) = \mathcal{L}(\boldsymbol{\theta}_t) + \mathbf{g}_t^T \boldsymbol{\delta} + \eta \|\boldsymbol{\delta}\|_2^2 \quad (6.49)$$

27 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
28 is equivalent to squared Euclidean distance in the **whitened coordinate system** $\boldsymbol{\phi} = \mathbf{F}^{\frac{1}{2}} \boldsymbol{\theta}$, since
29

$$\small{30} \quad \|\boldsymbol{\phi}_{t+1} - \boldsymbol{\phi}_t\|_2^2 = \|\mathbf{F}^{\frac{1}{2}}(\boldsymbol{\theta}_t + \boldsymbol{\delta}) - \mathbf{F}^{\frac{1}{2}} \boldsymbol{\theta}_t\|_2^2 = \|\mathbf{F}^{\frac{1}{2}} \boldsymbol{\delta}\|_2^2 = \|\boldsymbol{\delta}\|_F^2 \quad (6.50)$$

31 The new objective becomes
32

$$\small{33} \quad M_t(\boldsymbol{\delta}) = \mathcal{L}(\boldsymbol{\theta}_t) + \mathbf{g}_t^T \boldsymbol{\delta} + \eta \boldsymbol{\delta}^T \mathbf{F} \boldsymbol{\delta} \quad (6.51)$$

34 Solving $\nabla_{\boldsymbol{\delta}} M_t(\boldsymbol{\delta}) = \mathbf{0}$ gives the update
35

$$\small{36} \quad \boldsymbol{\delta}_t = -\eta \mathbf{F}^{-1} \mathbf{g}_t \quad (6.52)$$

37 This is the same as the natural gradient direction. Thus we can view NGD as a trust region method,
38 where we use a first-order approximation to the objective, and use FIM-distance in the constraint.
39

40 In the above derivation, we assumed \mathbf{F} was a constant matrix. In most problems, it will change at
41 each point in space, since we are optimizing in a curved space known as a **Riemannian manifold**.
42 For certain models, we can compute the FIM efficiently, allowing us to capture curvature information,
43 even though we use a first-order approximation to the objective.
44

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 (GN)
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

$$\frac{23}{24} p(\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{2\pi} \exp \left[-\frac{1}{2} \left((x_1 - \left[3\theta_1 + \frac{1}{3}\theta_2 \right])^2 - \frac{1}{2} \left(x_2 - \left[\frac{1}{3}\theta_1 \right] \right)^2 \right) \right] \quad (6.53)$$

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

³¹ The gradient of this objective is given by

³²

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \left(\begin{array}{l} = \mathbb{E}_{p^*(\mathbf{x})} [3(x_1 - [3\theta_1 + \frac{1}{3}\theta_2]) + \frac{1}{3}(x_2 - [\frac{1}{3}\theta_1])] \\ \mathbb{E}_{p^*(\mathbf{x})} [\frac{1}{3}(x_1 - [3\theta_1 + \frac{1}{3}\theta_2])] \end{array} \right) \quad (6.55)$$

³³

³⁴ Suppose that $p^*(\mathbf{x}) = p(\mathbf{x}; [0, 0])$. Then the Fisher matrix is a constant matrix, given by

³⁵

$$\frac{36}{37} \mathbf{F} = \begin{pmatrix} 3^2 + \frac{1}{3^2} & 1 \\ 1 & \frac{1}{3^2} \end{pmatrix} \quad (6.56)$$

³⁸

³⁹ Figure 6.3 compares steepest descent in $\boldsymbol{\theta}$ space with the natural gradient method, which is
⁴⁰ equivalent to steepest descent in ϕ space. Both methods start at $\boldsymbol{\theta} = (1, -1)$. The global optimum is
⁴¹ at $\boldsymbol{\theta} = (0, 0)$. We see that the NG method (blue dots) converges much faster to this optimum and
⁴² takes the shortest path, whereas steepest descent takes a very circuitous route. We also see that
⁴³ the gradient field in the whitened parameter space is more “spherical”, which makes descent much
⁴⁴ simpler and faster.

⁴⁵ Finally, note that since NGD is invariant to how we parameterize the distribution, we will get the
⁴⁶ same results even for a standard parameterization of the Gaussian. This is particularly useful if our
⁴⁷ probability model is more complex, such as a DNN (see e.g., [SSE18]).

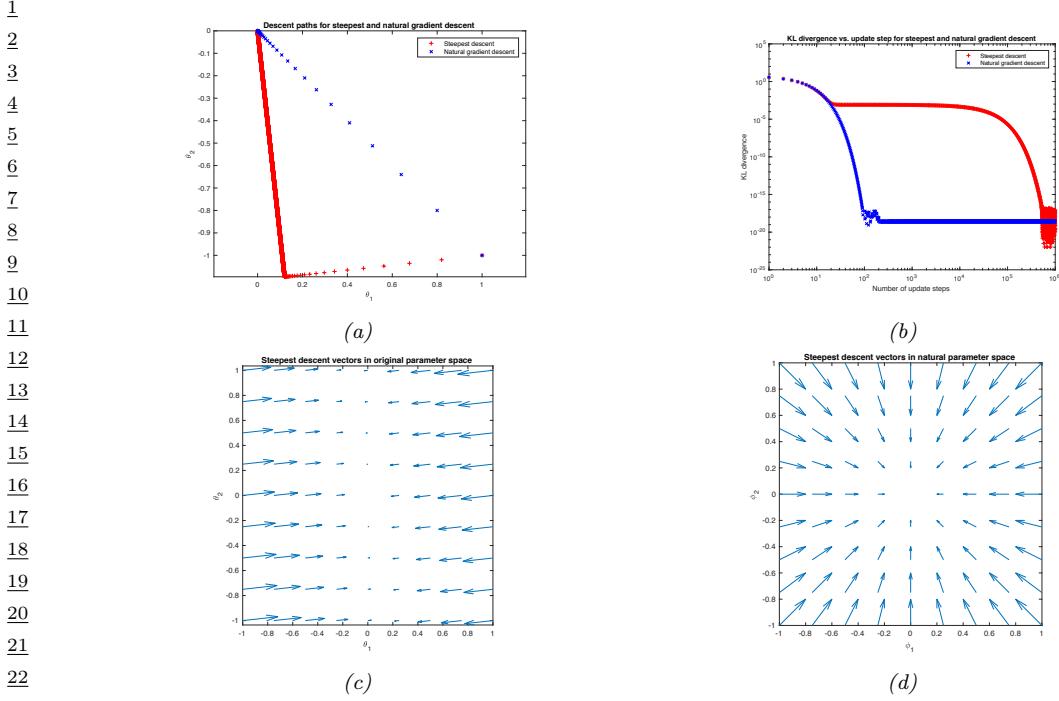


Figure 6.3: Illustration of the benefits of natural gradient vs steepest descent on a 2d problem. (a) Trajectories of the two methods in parameter space (red = steepest descent, blue = NG). They both start in the bottom right, at $(1, -1)$. (b) Objective vs number of iterations. (c) Gradient field in 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 $p_{\theta}(\mathbf{x}, \mathbf{y}) = p_{\mathcal{D}}(\mathbf{x})p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})$. Then we can compute the **empirical Fisher** Martens [Mar16] as
² follows:
³

$$\mathbf{F} = \mathbb{E}_{p_{\theta}(\mathbf{x}, \mathbf{y})} [\nabla \log p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) \nabla \log p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})^T] \quad (6.57)$$

$$\approx \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x}, \mathbf{y})} [\nabla \log p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) \nabla \log p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})^T] \quad (6.58)$$

$$= \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \nabla \log p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) \nabla \log p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})^T \quad (6.59)$$

⁴ This approximation is widely used, since it is simple to compute. In particular, we can compute a
⁵ diagonal approximation using the squared gradient vector. (This is similar to ADAGRAD, but only
⁶ uses the current gradient instead of a moving average of gradients; the latter is a better approach
⁷ when performing stochastic optimization.)

⁸ Unfortunately, the empirical Fisher does not work as well as the true Fisher [KBH19; Tho+19].
⁹ To see why, note that when we reach a flat part of parameter space where the gradient vector goes
¹⁰ to zero, the empirical Fisher will become singular, and hence the algorithm will get stuck on this
¹¹ plateau. However, the true Fisher takes expectations over the outputs, i.e., it marginalizes out \mathbf{y} .
¹² This will allow it to detect small changes in the output if we change the parameters. This is why the
¹³ natural gradient method can “escape” plateaus better than standard gradient methods.

¹⁴ An alternative strategy is to use exact computation of \mathbf{F} , but solve for $\mathbf{F}^{-1}\mathbf{g}$ approximately
¹⁵ using truncated conjugate gradient (CG) methods, where each CG step uses efficient methods for
¹⁶ Hessian-vector products [Pea94]. This is called **Hessian free optimization** [Mar10a]. However,
¹⁷ this approach can be slow, since it may take many CG iterations to compute a single parameter
¹⁸ update.

¹⁹ 6.4.5 Natural gradients for the exponential family

²⁰ In this section, we assume \mathcal{L} is an expected loss of the following form:
²¹

$$\mathcal{L}(\boldsymbol{\mu}) = \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\ell(\boldsymbol{\theta})] \quad (6.60)$$

²² where $q_{\boldsymbol{\mu}}(\boldsymbol{\theta})$ is an exponential family distribution with moment parameters $\boldsymbol{\mu}$. This is the basis of
²³ variational optimization (discussed in Section 6.8.3) and natural evolutionary strategies (discussed in
²⁴ the supplementary material).

²⁵ It turns out the gradient wrt the moment parameters is the same as the natural gradient wrt the
²⁶ natural parameters $\boldsymbol{\lambda}$. This follows from the chain rule:
²⁷

$$\frac{d}{d\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}) = \frac{d\boldsymbol{\mu}}{d\boldsymbol{\lambda}} \frac{d}{d\boldsymbol{\mu}} \mathcal{L}(\boldsymbol{\mu}) = \mathbf{F}(\boldsymbol{\lambda}) \nabla_{\boldsymbol{\mu}} \mathcal{L}(\boldsymbol{\mu}) \quad (6.61)$$

²⁸ where $\mathcal{L}(\boldsymbol{\mu}) = \mathcal{L}(\boldsymbol{\lambda}(\boldsymbol{\mu}))$, and where we used Equation (2.193) to write

$$\mathbf{F}(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\lambda}} \boldsymbol{\mu}(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\lambda}}^2 A(\boldsymbol{\lambda}) \quad (6.62)$$

²⁹ Hence

$$\tilde{\nabla}_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}) = \mathbf{F}(\boldsymbol{\lambda})^{-1} \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\mu}} \mathcal{L}(\boldsymbol{\mu}) \quad (6.63)$$

³⁰ It remains to compute the (regular) gradient wrt the moment parameters. The details on how to
³¹ do this will depend on the form of the q and the form of $\mathcal{L}(\boldsymbol{\lambda})$. We discuss some approaches to this
³² problem below.

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

$$\nabla_{\boldsymbol{\mu}} \mathcal{L}(\boldsymbol{\mu}) = \mathbf{F}(\boldsymbol{\lambda})^{-1} \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}) \quad (6.75)$$

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

$$\mathbf{F}(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\lambda}} \boldsymbol{\mu}(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})} [\mathcal{T}(\boldsymbol{\theta})] \quad (6.76)$$

$$\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})} [\ell(\boldsymbol{\theta})] \quad (6.77)$$

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

$$\tilde{\nabla}_{\boldsymbol{\lambda}} \mathbb{H}(\boldsymbol{\lambda}) = -\nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\log q(\boldsymbol{\theta})] \quad (6.78)$$

21 where, from Equation (2.123), we have
22

$$\log q(\boldsymbol{\theta}) = \log h(\boldsymbol{\theta}) + \mathcal{T}(\boldsymbol{\theta})^T \boldsymbol{\lambda} - A(\boldsymbol{\lambda}) \quad (6.79)$$

25 Since $\mathbb{E}[\mathcal{T}(\boldsymbol{\theta})] = \boldsymbol{\mu}$, we have
26

$$\nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\log q(\boldsymbol{\theta})] = \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q(\boldsymbol{\theta})} [\log h(\boldsymbol{\theta})] + \nabla_{\boldsymbol{\mu}} \boldsymbol{\mu}^T \boldsymbol{\lambda}(\boldsymbol{\mu}) - \nabla_{\boldsymbol{\mu}} A(\boldsymbol{\lambda}) \quad (6.80)$$

28 where $h(\boldsymbol{\theta})$ is the base measure. Since $\boldsymbol{\lambda}$ is a function of $\boldsymbol{\mu}$, we have
29

$$\nabla_{\boldsymbol{\mu}} \boldsymbol{\mu}^T \boldsymbol{\lambda} = \boldsymbol{\lambda} + (\nabla_{\boldsymbol{\mu}} \boldsymbol{\lambda})^T \boldsymbol{\mu} = \boldsymbol{\lambda} + (\mathbf{F}_{\boldsymbol{\lambda}}^{-1} \nabla_{\boldsymbol{\lambda}} \boldsymbol{\lambda})^T \boldsymbol{\mu} = \boldsymbol{\lambda} + \mathbf{F}_{\boldsymbol{\lambda}}^{-1} \boldsymbol{\mu} \quad (6.81)$$

31 and since $\boldsymbol{\mu} = \nabla_{\boldsymbol{\lambda}} A(\boldsymbol{\lambda})$ we have
32

$$\nabla_{\boldsymbol{\mu}} A(\boldsymbol{\lambda}) = \mathbf{F}_{\boldsymbol{\lambda}}^{-1} \nabla_{\boldsymbol{\lambda}} A(\boldsymbol{\lambda}) = \mathbf{F}_{\boldsymbol{\lambda}}^{-1} \boldsymbol{\mu} \quad (6.82)$$

34 Hence
35

$$-\nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\log q(\boldsymbol{\theta})] = -\nabla_{\boldsymbol{\mu}} \mathbb{E}_{q(\boldsymbol{\theta})} [\log h(\boldsymbol{\theta})] - \boldsymbol{\lambda} \quad (6.83)$$

38 If we assume that $h(\boldsymbol{\theta}) = \text{const}$, as is often the case, we get
39

$$\tilde{\nabla}_{\boldsymbol{\lambda}} \mathbb{H}(q) = -\boldsymbol{\lambda} \quad (6.84)$$

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

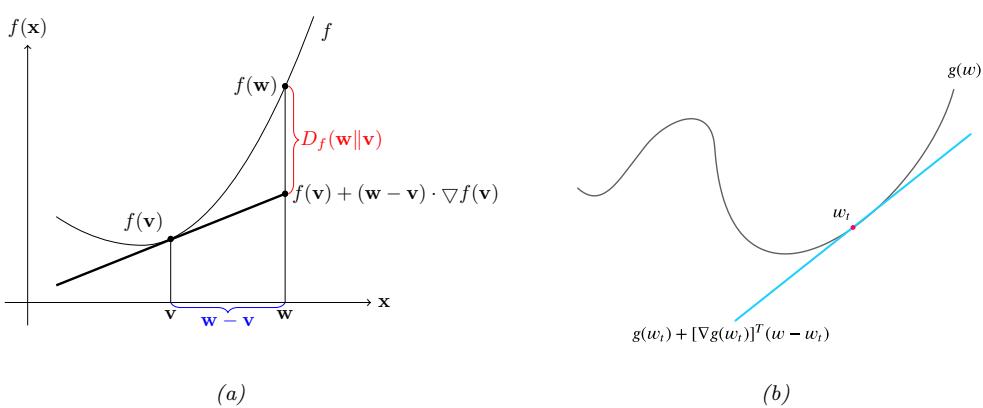


Figure 6.4: (a) Illustration of Bregman divergence. (b) A locally linear approximation to a non-convex function.

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

$$\hat{f}_\mathbf{v}(\mathbf{w}) = f(\mathbf{v}) + (\mathbf{w} - \mathbf{v})^\top \nabla f(\mathbf{v}) \quad (6.86)$$

be a first order Taylor series approximation to f centered at \mathbf{v} . Then the Bregman divergence is the difference from this linear approximation:

$$D_f(\mathbf{w}, \mathbf{v}) = f(\mathbf{w}) - \hat{f}_\mathbf{v}(\mathbf{w}) \quad (6.87)$$

See Figure 6.4a for an illustration. Since f is convex, we have $D_f(\mathbf{v}||\mathbf{w}) \geq 0$, since $\hat{f}_\mathbf{v}$ 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.

6.5.2 Proximal point method

Suppose we make a locally linear approximation to the objective at step t centered at $\boldsymbol{\theta}_t$:

$$\hat{\mathcal{L}}_t(\boldsymbol{\theta}) = \mathcal{L}_t(\boldsymbol{\theta}_t) + \mathbf{g}_t^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_t) \quad (6.88)$$

where $\mathbf{g}_t = \nabla_{\boldsymbol{\theta}} \mathcal{L}_t(\boldsymbol{\theta}_t)$. This is shown in Figure 6.4b.

If we optimize this approximation using gradient descent, we may end up at $-\infty$. To prevent this, we can use a **proximal update**, which adds a quadratic penalty to ensure we don't move too far from where the locally linear approximation is valid:

$$\boldsymbol{\theta}_{t+1} = \text{prox}_{\eta_t \hat{\mathcal{L}}_t}(\boldsymbol{\theta}_t) \triangleq \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \hat{\mathcal{L}}_t(\boldsymbol{\theta}) + \frac{1}{2\eta_t} \|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|_2^2 \quad (6.89)$$

We can solve this optimization problem by setting the gradient to 0:

$$\nabla_{\boldsymbol{\theta}} \left[\mathcal{L}_t(\boldsymbol{\theta}_t) + \mathbf{g}_t^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_t) + \frac{1}{2\eta_t} \|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|_2^2 \right] = \mathbf{g}_t + \frac{1}{\eta_t} (\boldsymbol{\theta} - \boldsymbol{\theta}_t) = \mathbf{0} \quad (6.90)$$

This yields the standard gradient descent update:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \mathbf{g}_t \quad (6.91)$$

However, by changing from Euclidean norm to another distance metric, we can derive mirror descent, as we show below.

6.5.3 PPM using Bregman divergence

Suppose we replace the Euclidean distance term $\|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|_2^2$ by a more general Bregman divergence (Equation (6.85)),

$$D_h(\mathbf{x}, \mathbf{y}) = h(\mathbf{x}) - [h(\mathbf{y}) + \nabla h(\mathbf{y})^\top (\mathbf{x} - \mathbf{y})] \quad (6.92)$$

where $h(\mathbf{x})$ is a strongly convex function. Combined with our linear approximation, this gives the following update:

$$\boldsymbol{\theta}_{t+1} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \hat{\mathcal{L}}(\boldsymbol{\theta}) + \frac{1}{\eta_t} D_h(\boldsymbol{\theta}, \boldsymbol{\theta}_t) \quad (6.93)$$

$$= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \eta_t \mathbf{g}_t^\top \boldsymbol{\theta} + D_h(\boldsymbol{\theta}, \boldsymbol{\theta}_t) \quad (6.94)$$

This is known as **mirror descent** [NY83; BT03]. This can easily be extended to the stochastic setting in the obvious way.

One can show that natural gradient descent (Section 6.4) is a form of mirror descent [RM15a]. More precisely, mirror descent in the mean parameter space is equivalent to natural gradient descent in the canonical parameter space.

6.6 Gradients of stochastic functions

In this section, we discuss how to compute the gradient of stochastic functions of the form

$$\mathcal{L}(\boldsymbol{\psi}) = \mathbb{E}_{q_{\boldsymbol{\psi}}(\mathbf{z})} [\ell(\boldsymbol{\psi}, \mathbf{z})] \quad (6.95)$$

6.6.1 Minibatch approximation to finite-sum objectives

In the simplest case, $q_\psi(\mathbf{z})$ does not depend on ψ . In this case, we can push gradients inside the expectation operator, $\nabla \mathcal{L}(\psi) = \mathbb{E}[\nabla \ell(\psi, \mathbf{z})]$ and then use Monte Carlo sampling for \mathbf{z} to approximate the gradient.

For example, consider the empirical risk minimization (ERM) problem of minimizing

$$\mathcal{L}(\psi) = \frac{1}{N} \sum_{n=1}^N \ell(\psi, \mathbf{z}_n) \quad (6.96)$$

where $\mathbf{z}_n = (\mathbf{x}_n, \mathbf{y}_n)$ and

$$\ell(\psi, \mathbf{z}_n) = \ell(h(\mathbf{x}_n; \psi), \mathbf{y}_n) \quad (6.97)$$

is the per-example loss, where h is a prediction function. This kind of objective is called a **finite sum objective**.

Now consider trying to minimize this objective. If, at each iteration, we evaluate the objective (and its gradient) using all N datapoints, the method is called **batch optimization**. However, this can be very slow if the dataset is large. Fortunately, we can reformulate it as a stochastic optimization problem, which will be faster to solve. To do this, note that Equation (6.96) can be written as an expectation wrt the empirical distribution:

$$\mathcal{L}(\psi) = \mathbb{E}_{\mathbf{z} \sim p_D} [\ell(\psi, \mathbf{z})] \quad (6.98)$$

Since the distribution is independent of the parameters, we can easily use Monte Carlo sampling to approximate the objective and its gradient. In particular, we will sample a **minibatch** of $B = |\mathcal{B}|$ datapoints from the full set \mathcal{D} at each iteration. More precisely, we have

$$\mathcal{L}(\psi) \approx \frac{1}{B} \sum_{n \in \mathcal{B}} \ell(h(\mathbf{x}_n; \psi), \mathbf{y}_n) \quad (6.99)$$

$$\nabla \mathcal{L}(\psi) \approx \frac{1}{B} \sum_{n \in \mathcal{B}} \nabla \ell(h(\mathbf{x}_n; \psi), \mathbf{y}_n) \quad (6.100)$$

These noisy gradients can then be passed to SGD, which is robust to noisy gradients (see Section 6.3).

6.6.2 Optimizing parameters of a distribution

Now suppose the stochasticity depends on the parameters we are optimizing. For example, \mathbf{z} could be an action sampled from a stochastic policy q_ψ , as in RL (Section 37.3.2). In this case, the gradient is given by

$$\nabla_\psi \mathbb{E}_{q_\psi(\mathbf{z})} [\ell(\psi, \mathbf{z})] = \nabla_\psi \int \ell(\psi, \mathbf{z}) q_\psi(\mathbf{z}) d\mathbf{z} \quad (6.101)$$

$$= \int [\nabla_\psi \ell(\psi, \mathbf{z})] q_\psi(\mathbf{z}) d\mathbf{z} + \int \ell(\psi, \mathbf{z}) [\nabla_\psi q_\psi(\mathbf{z})] d\mathbf{z} \quad (6.102)$$

1 The first term can be approximated by Monte Carlo sampling:
2

$$\int [\nabla_{\psi} \ell(\psi, z)] q_{\psi}(z) dz \approx \frac{1}{S} \sum_{s=1}^S \nabla_{\psi} \ell(\psi, z_s) \quad (6.103)$$

6 where $z_s \sim q_{\psi}$. Note that if $\ell()$ is independent of ψ , this term vanishes.
7

8 Now consider the second term, that takes the gradients of the distribution itself:
9

$$I \triangleq \int \ell(\psi, z) [\nabla_{\psi} q_{\psi}(z)] dz \quad (6.104)$$

11 We can no longer use vanilla Monte Carlo sampling to approximate this integral. However, there are
12 various other ways to approximate this (see [Moh+19a] for an extensive review). We briefly describe
13 the two main methods in Section 6.6.3 and Section 6.6.4.
14

15 6.6.3 Score function estimator (likelihood ratio trick)

17 The simplest way to approximate Equation (6.104) is to exploit the **log derivative trick**, which is
18 the following identity:
19

$$\nabla_{\psi} q_{\psi}(z) = q_{\psi}(z) \nabla_{\psi} \log q_{\psi}(z) \quad (6.105)$$

21 With this, we can rewrite Equation (6.104) as follows:
22

$$I = \int \ell(\psi, z) [q_{\psi}(z) \nabla_{\psi} \log q_{\psi}(z)] dz = \mathbb{E}_{q_{\psi}(z)} [\ell(\psi, z) \nabla_{\psi} \log q_{\psi}(z)] \quad (6.106)$$

25 This is called the **score function estimator** or **SFE** [Fu15]. (The term ‘‘score function’’ refers to
26 the gradient of a log probability distribution, as explained in Section 2.6.1.) It is also called the
27 **likelihood ratio gradient estimator**. We can now easily approximate this with Monte Carlo:
28

$$I \approx \frac{1}{S} \sum_{s=1}^S \ell(\psi, z_s) \nabla_{\psi} \log q_{\psi}(z_s) \quad (6.107)$$

32 We only require that the sampling distribution is differentiable, not the objective $\ell(\psi, z)$ itself. This
33 allows the method to be used for blackbox stochastic optimization problems, such as variational
34 optimization (Section 6.8.3), black-box variational inference (Section 10.3.1), reinforcement learning
35 (Section 37.3.2), etc.
36

37 6.6.3.1 Control variates

39 The score function estimate can have high variance. One way to reduce this is to use **control**
40 **variates**, in which we replace $\ell(\psi, z)$ with
41

$$\hat{\ell}(\psi, z) = \ell(\psi, z) - c(b(\psi, z) - \mathbb{E}[b(\psi, z)]) \quad (6.108)$$

43 where $b(\psi, z)$ is a **baseline function** that is correlated with $\ell(\psi, z)$, and $c > 0$ is a coefficient. Since
44 $\mathbb{E}[\hat{\ell}(\psi, z)] = \mathbb{E}[\ell(\psi, z)]$, we can use $\hat{\ell}$ to compute unbiased gradient estimates of ℓ . The advantage
45 is that this new estimate can result in lower variance, as we show in Section 11.6.2.
46

47

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 However, in the special case of Gaussian distributions, we can also compute the gradient vector
4 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 Similarly, Price's theorem states that

$$\frac{\partial}{\partial v} \mathbb{E} [\ell(z)] = 0.5 \mathbb{E} \left[\frac{\partial^2}{\partial z^2} \ell(z) \right] \quad (6.115)$$

15 In `gradient_expected_value_gaussian.py` we show that these two methods are numerically equivalent,
16 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

$$\nabla_\psi \ell(\psi, z)^{\text{TD}} = \nabla_\psi \ell(\psi, z) + \mathbf{J}^\top \nabla_z \ell(\psi, z) \quad (6.117)$$

32 where $\mathbf{J} = \frac{\partial z}{\partial \psi}$ is the $d_z \times d_\psi$ Jacobian matrix of the noise transformation:

$$\mathbf{J} = \begin{pmatrix} \frac{\partial z_1}{\partial \psi_1} & \dots & \frac{\partial z_1}{\partial \psi_{d_\psi}} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_{d_z}}{\partial \psi_1} & \dots & \frac{\partial z_{d_z}}{\partial \psi_{d_\psi}} \end{pmatrix} \quad (6.118)$$

40 Hence we can compute the gradient using

$$\nabla_\psi \mathcal{L}(\psi) = \mathbb{E}_{q_0(\epsilon)} [\nabla_\psi \ell(\psi, z) + \mathbf{J}(\psi, \epsilon)^\top \nabla_z \ell(\psi, r(\psi, \epsilon))] \quad (6.119)$$

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.

6.6.5 The delta method

The **delta method** [Hoe12] approximates the expectation of a function of a random variable by the expectation of the function's Taylor expansion. For example, suppose $\boldsymbol{\theta} \sim q$ with mean $\mathbb{E}_{q(\boldsymbol{\theta})}[\boldsymbol{\theta}] = \mathbf{m}$, and we use a first order expansion. Then we have

$$\mathbb{E}_{q(\boldsymbol{\theta})}[f(\boldsymbol{\theta})] \approx \mathbb{E}_{q(\boldsymbol{\theta})}[f(\mathbf{m}) + (\boldsymbol{\theta} - \mathbf{m})^\top \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}}] \approx f(\mathbf{m}) \quad (6.120)$$

Now let $f(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$. Then we have

$$\mathbb{E}_{q(\boldsymbol{\theta})}[\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})] \approx \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}} \quad (6.121)$$

This is called the **first-order delta method**.

Now let $f(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}}^2 \mathcal{L}(\boldsymbol{\theta})$. Then we have

$$\mathbb{E}_{q(\boldsymbol{\theta})}[\nabla_{\boldsymbol{\theta}}^2 \mathcal{L}(\boldsymbol{\theta})] \approx \nabla_{\boldsymbol{\theta}}^2 \mathcal{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}} \quad (6.122)$$

This is called the **second-order delta method**.

6.6.6 Gumbel softmax trick

When working with discrete variables, we cannot use the reparameterization trick. However, we can often relax the discrete variables to continuous ones in a way which allows the trick to be used, as we explain below.

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 to represent a K -ary categorical variable d . Let $P(d) = \text{Cat}(d|\boldsymbol{\pi})$, where $\pi_k = P(d_k = 1)$, so $0 \leq \pi_k \leq 1$. Alternatively we can parameterize the distribution in terms of $(\alpha_1, \dots, \alpha_K)$, where $\pi_k = \alpha_k / (\sum_{k'=1}^K \alpha_{k'})$. We will denote this by $d \sim \text{Cat}(d|\boldsymbol{\alpha})$.

We can sample a one-hot vector \mathbf{d} from this distribution by computing

$$\mathbf{d} = \text{onehot}(\underset{k}{\operatorname{argmax}} [\epsilon_k + \log \alpha_k]) \quad (6.123)$$

where $\epsilon_k \sim \text{Gumbel}(0, 1)$ is sampled from the **Gumbel distribution** [Gum54]. We can draw such samples by first sampling $u_k \sim \text{Unif}(0, 1)$ and then computing $\epsilon_k = -\log(-\log(u_k))$. This is called the **Gumbel-Max trick** [MTM14], and gives us a reparameterizable representation for the categorical distribution.

Unfortunately, the derivative of the argmax is 0 everywhere except at the boundary of transitions from one label to another, where the derivative is undefined. However, suppose we replace the argmax with a softmax, and replace the discrete one-hot vector \mathbf{d} with a continuous relaxation $\mathbf{x} \in \Delta^{K-1}$, 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 write

$$x_k = \frac{\exp((\log \alpha_k + \epsilon_k)/\tau)}{\sum_{k'=1}^K \exp((\log \alpha_{k'} + \epsilon_{k'})/\tau)} \quad (6.124)$$

where $\tau > 0$ is a temperature parameter. This is called the **Gumbel-Softmax distribution** [JGP17] or the **concrete distribution** [MMT17]. This smoothly approaches the discrete distribution as $\tau \rightarrow 0$, as illustrated in Figure 6.5.

We can now replace $f(\mathbf{d})$ with $f(\mathbf{x})$, which allows us to take reparameterized gradients wrt \mathbf{x} .

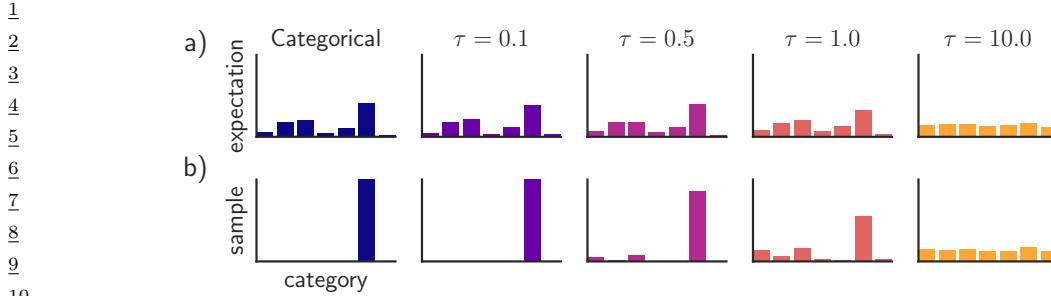


Figure 6.5: Illustration of the Gumbel-Softmax (concrete) distribution with $K = 7$ states at different temperatures τ . The top row shows $\mathbb{E}[z]$, and the bottom row shows samples $z \sim \text{GumbelSoftmax}(\alpha, \tau)$. The left column shows a discrete (categorical) distribution, which 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.

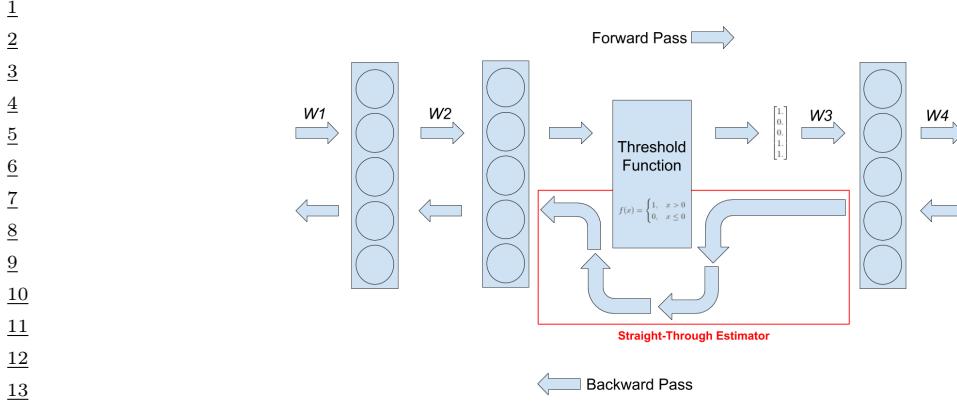


Figure 6.6: Illustration of straight-through estimator when applied to a binary threshold function in the middle of an MLP. From <https://www.hassanaskary.com/python/pytorch/deep%20learning/2020/09/19/intuitive-explanation-of-straight-through-estimators.html>. Used with kind permission of Hassan Askary.

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 $\ell(\boldsymbol{\theta}')$ 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

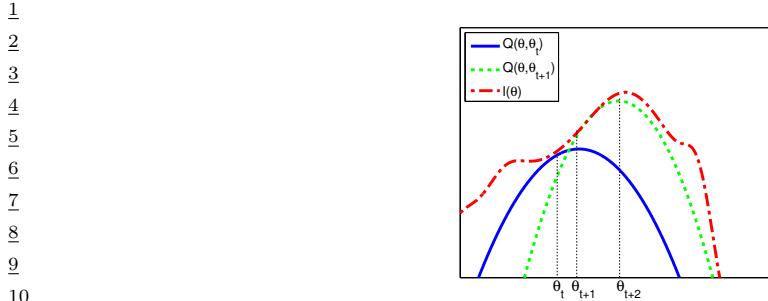


Figure 6.7: Illustration of a bound optimization algorithm. Adapted from Figure 9.14 of [Bis06]. Generated by `emLogLikelihoodMax.py`.

touches the objective function at θ^t . We then set θ^{t+1} to the maximum of the lower bound (blue curve), and fit a new bound at that point (dotted green curve). The maximum of this new bound becomes θ^{t+2} , etc.

6.7.2 Example: logistic regression

If $LL(\theta)$ is a concave function we want to maximize, then one way to obtain a valid lower bound is to use a bound on its Hessian, i.e., to find a matrix a negative definite matrix \mathbf{B} such that $\mathbf{H}(\theta) \succ \mathbf{B}$. In this case, one can show (see [BCN18, App. B]) that

$$LL(\theta) \geq LL(\theta^t) + (\theta - \theta^t)^T g(\theta^t) + \frac{1}{2}(\theta - \theta^t)^T \mathbf{B}(\theta - \theta^t) \quad (6.129)$$

where $g(\theta^t) = \nabla LL(\theta^t)$. Therefore the following function is a valid lower bound:

$$Q(\theta, \theta^t) = \theta^T(g(\theta^t) - \mathbf{B}\theta^t) + \frac{1}{2}\theta^T\mathbf{B}\theta \quad (6.130)$$

The corresponding update becomes

$$\theta^{t+1} = \theta^t - \mathbf{B}^{-1}g(\theta^t) \quad (6.131)$$

This is similar to a Newton update, except we use \mathbf{B} , which is a fixed matrix, rather than $\mathbf{H}(\theta^t)$, which changes at each iteration. This can give us some of the advantages of second order methods at lower computational cost.

For example, let us fit a multi-class logistic regression model using MM. (We follow the presentation of [Kri+05], who also consider the more interesting case of *sparse* logistic regression.) The probability that example n belongs to class $c \in \{1, \dots, C\}$ is given by

$$p(y_n = c | \mathbf{x}_n, \mathbf{w}) = \frac{\exp(\mathbf{w}_c^T \mathbf{x}_n)}{\sum_{i=1}^C \exp(\mathbf{w}_i^T \mathbf{x}_n)} \quad (6.132)$$

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, in binary logistic regression, where $C = 2$, we only learn a single weight vector.) Therefore the parameters θ correspond to a weight matrix \mathbf{w} of size $D(C - 1)$, where $\mathbf{x}_n \in \mathbb{R}^D$.

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**

18 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:

$$\underline{\mathcal{L}}(\boldsymbol{\theta}, q_n | \mathbf{y}_n) = \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log \frac{p(\mathbf{y}_n, \mathbf{z}_n | \boldsymbol{\theta})}{q_n(\mathbf{z}_n)} \quad (6.146)$$

$$= \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log \frac{p(\mathbf{z}_n | \mathbf{y}_n, \boldsymbol{\theta}) p(\mathbf{y}_n | \boldsymbol{\theta})}{q_n(\mathbf{z}_n)} \quad (6.147)$$

$$= \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log \frac{p(\mathbf{z}_n | \mathbf{y}_n, \boldsymbol{\theta})}{q_n(\mathbf{z}_n)} + \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log p(\mathbf{y}_n | \boldsymbol{\theta}) \quad (6.148)$$

$$= -D_{\text{KL}}(q_n(\mathbf{z}_n) \| p(\mathbf{z}_n | \mathbf{y}_n, \boldsymbol{\theta})) + \log p(\mathbf{y}_n | \boldsymbol{\theta}) \quad (6.149)$$

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 $\underline{\mathcal{L}}(\boldsymbol{\theta}, \{q_n\} | \mathcal{D})$ 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:

$$\underline{\mathcal{L}}(\boldsymbol{\theta}, \{q_n^*\} | \mathcal{D}) = \sum_n \log p(\mathbf{y}_n | \boldsymbol{\theta}) = LL(\boldsymbol{\theta} | \mathcal{D}) \quad (6.150)$$

To see how this connects to bound optimization, let us define

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t) = \underline{\mathcal{L}}(\boldsymbol{\theta}, \{p(\mathbf{z}_n | \mathbf{y}_n; \boldsymbol{\theta}^t)\}) \quad (6.151)$$

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 $\underline{\mathcal{L}}(\boldsymbol{\theta}, \{q_n^t\})$ 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 In the case of the exponential family, the maximization can be solved in closed-form by matching the
6 moments of the expected sufficient statistics (Section 2.5.5).
7

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 6.7.4 Example: EM for an MVN with missing data

14 It is easy to compute the MLE for a multivariate normal when we have a fully observed data matrix:
15 we just compute the sample mean and covariance. In this section, we consider the case where we have
16 **missing data or partially observed data**. For example, we can think of the entries of \mathbf{Y} as being
17 answers to a survey; some of these answers may be unknown. There are many kinds of missing data,
18 as we discuss in Section 22.3.5. In this section, we make the missing at random (MAR) assumption,
19 for simplicity. Under the MAR assumption, the log likelihood of the visible data has the form
20

$$\underline{22} \quad \underline{23} \quad \log p(\mathbf{X} | \boldsymbol{\theta}) = \sum_n \log p(\mathbf{x}_n | \boldsymbol{\theta}) = \sum_n \log \left[\int p(\mathbf{x}_n, \mathbf{z}_n | \boldsymbol{\theta}) d\mathbf{z}_n \right] \quad (6.155)$$

24 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
25 the variables. Unfortunately, this objective is hard to maximize. since we cannot push the log inside
26 the expectation. Fortunately, we can easily apply EM, as we explain below.
27

28 6.7.4.1 E step

29 Suppose we have the parameters $\boldsymbol{\theta}^{t-1}$ from the previous iteration. Then we can compute the expected
30 complete data log likelihood at iteration t as follows:

$$\underline{31} \quad Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1}) = \mathbb{E} \left[\sum_{n=1}^N \log \mathcal{N}(\mathbf{y}_n | \boldsymbol{\mu}, \boldsymbol{\Sigma}) | \mathcal{D}, \boldsymbol{\theta}^{t-1} \right] \quad (6.156)$$

$$\underline{32} \quad = -\frac{N}{2} \log |2\pi\boldsymbol{\Sigma}| - \frac{1}{2} \sum_n \mathbb{E} [(\mathbf{y}_n - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{y}_n - \boldsymbol{\mu})] \quad (6.157)$$

$$\underline{33} \quad = -\frac{N}{2} \log |2\pi\boldsymbol{\Sigma}| - \frac{1}{2} \text{tr}(\boldsymbol{\Sigma}^{-1} \sum_n \mathbb{E} [(\mathbf{y}_n - \boldsymbol{\mu})(\mathbf{y}_n - \boldsymbol{\mu})^\top]) \quad (6.158)$$

$$\underline{34} \quad = -\frac{N}{2} \log |\boldsymbol{\Sigma}| - \frac{ND}{2} \log(2\pi) - \frac{1}{2} \text{tr}(\boldsymbol{\Sigma}^{-1} \mathbb{E} [\mathbf{S}(\boldsymbol{\mu})]) \quad (6.159)$$

35 where

$$\underline{36} \quad \mathbb{E} [\mathbf{S}(\boldsymbol{\mu})] \triangleq \sum_n \left(\mathbb{E} [\mathbf{y}_n \mathbf{y}_n^\top] + \boldsymbol{\mu} \boldsymbol{\mu}^\top - 2\boldsymbol{\mu} \mathbb{E} [\mathbf{y}_n]^\top \right) \quad (6.160)$$

(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

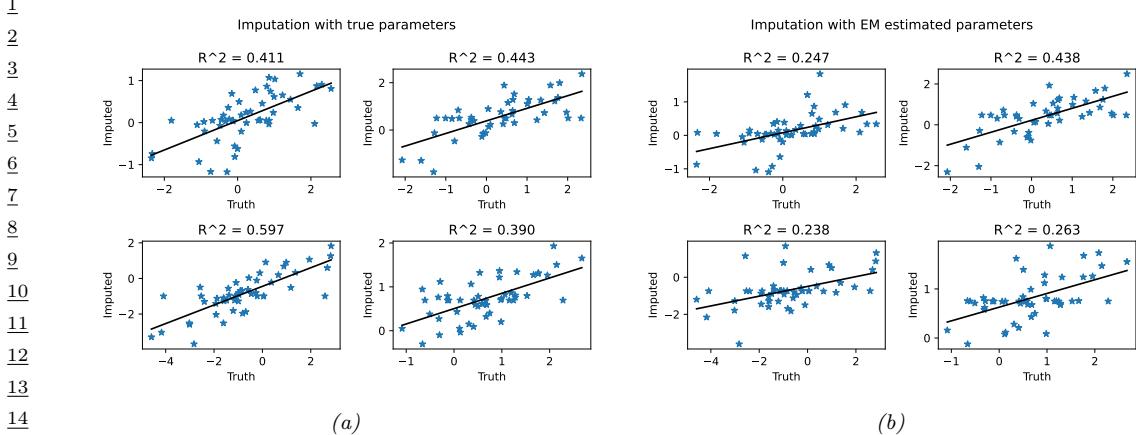


Figure 6.8: Illustration of data imputation using a multivariate Gaussian. (a) Scatter plot of true values vs imputed values using true parameters. (b) Same as (a), but using parameters estimated with EM. We just show the first four variables, for brevity. Generated by [gauss_imputation_em_demo.py](#).

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

$$\log p(\mathbf{y}, \mathbf{z} | \mathbf{X}, \mathbf{w}, \sigma^2, \nu) = \sum_n -\frac{1}{2} \log(2\pi z_n \sigma^2) - \frac{1}{2z_n \sigma^2} (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 \quad (6.170)$$

$$+ \left(\frac{\nu}{2} - 1\right) \log(z_n) - z_n \frac{\nu}{2} + \text{const} \quad (6.171)$$

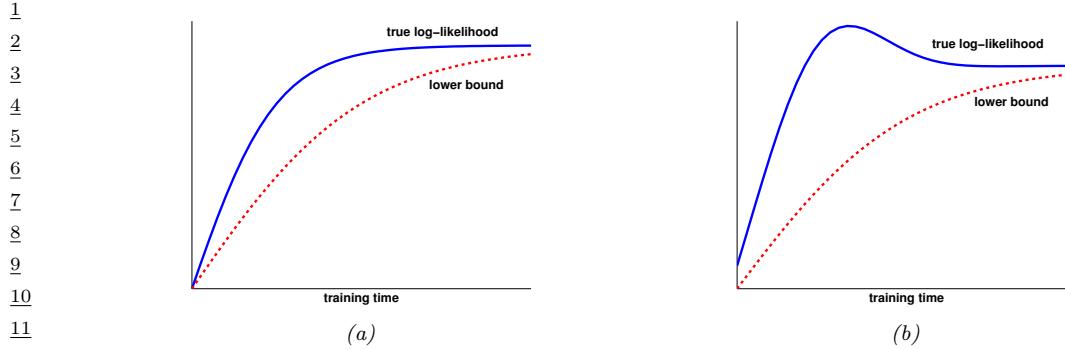


Figure 6.9: Illustration of possible behaviors of variational EM. (a) The lower bound increases at each iteration, and so does the likelihood. (b) The lower bound increases but the likelihood decreases. In this case, the algorithm is closing the gap between the approximate and true posterior. This can 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 $\lambda_n^t \triangleq \mathbb{E}[1/z_n | y_n, \mathbf{x}_n, \boldsymbol{\theta}^t]$. 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 2.2.8, 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; Neal12].

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

³³ 6.7.6.4 Generalized EM

³⁴ Sometimes we can perform the E step exactly, but we cannot perform the M step exactly. However,
³⁵ we can still monotonically increase the log likelihood by performing a “partial” M step, in which we
³⁶ merely increase the expected complete data log likelihood, rather than maximizing it. For example,
³⁷ we might follow a few gradient steps. This is called the **generalized EM** or **GEM** algorithm.
³⁸

³⁹

⁴⁰ 6.7.6.5 ECM algorithm

⁴¹ The **ECM** algorithm stands for “expectation conditional maximization”, and refers to optimizing the
⁴² parameters in the M step sequentially, if they turn out to be dependent. The **ECME** algorithm,
⁴³ which stands for “ECM either” [LR95], is a variant of ECM in which we maximize the expected
⁴⁴ complete data log likelihood (the Q function) as usual, or the observed data log likelihood, during
⁴⁵ one or more of the conditional maximization steps. The latter can be much faster, since it ignores
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 $LL_n(\theta) = \log p(\mathbf{x}_n | \theta)$ 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

$$\bar{\ell}(\theta) = \sum_{n=1}^N \ell(\mathbf{y}_n, f_{\theta}(\mathbf{x}_n)) + R(\theta) \quad (6.175)$$

where $f_{\theta}(\mathbf{x})$ is a prediction function, $\ell(\mathbf{y}, \hat{\mathbf{y}})$ 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:

$$\mathcal{L}(q) = \mathbb{E}_{q(\theta)} \left[\sum_{n=1}^N \ell(\mathbf{y}_n, f_{\theta}(\mathbf{x}_n)) \right] + D_{\text{KL}}(q(\theta) \| \pi_0(\theta)) \quad (6.176)$$

We can rewrite the KL term as

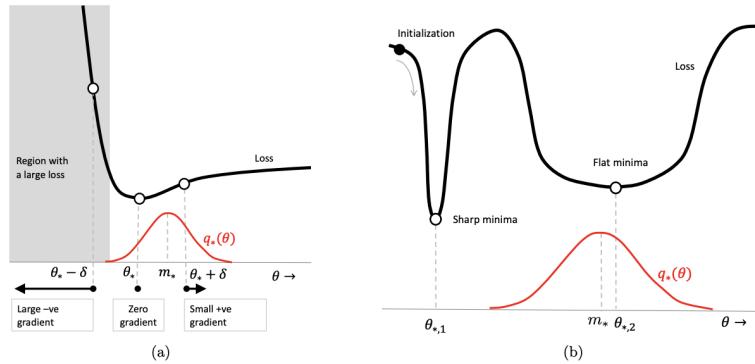
$$D_{\text{KL}}(q(\theta) \| \pi_0(\theta)) = \mathbb{E}_{q(\theta)} [R(\theta)] + \mathbb{H}(q(\theta)) \quad (6.177)$$

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.

1
2
3
4
5
6
7
8
9
10
11
12
13



14 *Figure 6.10: Illustration of the robustness obtained by using a Bayesian approach to parameter estimation.*
 15 (a) When the minimum θ_* lies next to a “wall”, the Bayesian solution shifts away from the boundary to avoid
 16 large losses due to perturbations of the parameters. (b) The Bayesian solution prefers flat minima over sharp
 17 minima, to avoid large losses due to perturbations of the parameters. From Figure 1 of [KR21a]. Used with
 18 kind permission of Emtiyaz Khan.

19
20

21 6.8.1 Deriving inference algorithms from BLR

22 In this section we show how to derive several different inference algorithms from BLR. (We discuss
 23 such algorithms in more detail in Chapter 10.)

24
25

26 6.8.1.1 Bayesian inference as optimization

27 The BLR objective includes standard exact Bayesian inference as a special case, as first shown in
 28 [Opt88]. To see this, let us assume the loss function is derived from a log-likelihood:
 29

$$30 \quad \ell(y, f_{\theta}(x)) = -\log p(y|f_{\theta}(x)) \quad (6.179)$$

31

32 Let $\mathcal{D} = \{(x_n, y_n) : n = 1 : N\}$ be the data we condition on. The Bayesian posterior can be written
 33 as

34

$$35 \quad p(\theta|\mathcal{D}) = \frac{1}{Z(\mathcal{D})} \pi_0(\theta) \prod_{n=1}^N p(y_n|f_{\theta}(x_n)) \quad (6.180)$$

36

37 This can be derived by minimizing the BLR, since

38

$$39 \quad \mathcal{L}(q) = -\mathbb{E}_{q(\theta)} \left[\sum_{n=1}^N \log p(y_n|f_{\theta}(x_n)) \right] + D_{\text{KL}}(q(\theta)\|p(\theta|\mathcal{D})) \quad (6.181)$$

40

$$41 \quad = \mathbb{E}_{q(\theta)} \left[\log \frac{q(\theta)}{\frac{1}{Z(\mathcal{D})} \prod_{n=1}^N p(y_n|f_{\theta}(x_n))} \right] - \log Z(\mathcal{D}) \quad (6.182)$$

42

$$43 \quad = D_{\text{KL}}(q(\theta)\|p(\theta|\mathcal{D})) - \log Z(\mathcal{D}) \quad (6.183)$$

44

45

46

47

1 Since $Z(\mathcal{D})$ is a constant, we can minimize the loss by by setting $q(\boldsymbol{\theta}) = p(\boldsymbol{\theta}|\mathcal{D})$.

2 Of course, we can use other kinds of loss, not just log likelihoods. This results in a framework
3 known as **generalized Bayesian inference** [BHW16; KJD19; KJD21]. See Section 14.1.3 for more
4 discussion.

5 6.8.1.2 Optimization of BLR using natural gradient descent

6 In general, we cannot compute the exact posterior $q(\boldsymbol{\theta}) = p(\boldsymbol{\theta}|\mathcal{D})$, so we seek an approximation. We
7 will assume that $q(\boldsymbol{\theta})$ is an exponential family distribution, such as a multivariate Gaussian, where
8 the mean represents the standard point estimate of $\boldsymbol{\theta}$ (as in ERM), and the covariance represents our
9 uncertainty (as in Bayes). Hence q can be written as follows:

$$\underline{13} \quad q(\boldsymbol{\theta}) = h(\boldsymbol{\theta}) \exp[\boldsymbol{\lambda}^T \mathcal{T}(\boldsymbol{\theta}) - A(\boldsymbol{\lambda})] \quad (6.184)$$

14 where $\boldsymbol{\lambda}$ are the natural parameters, $\mathcal{T}(\boldsymbol{\theta})$ are the sufficient statistics, $A(\boldsymbol{\lambda})$ is the log partition
15 function, and $h(\boldsymbol{\theta})$ is the base measure, which is usually a constant. The BLR loss becomes

$$\underline{16} \quad \mathcal{L}(\boldsymbol{\lambda}) = \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] - \mathbb{H}(q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})) \quad (6.185)$$

17 We can optimize this using natural gradient descent (Section 6.4). The update becomes

$$\underline{18} \quad \boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t - \eta_t \tilde{\nabla}_{\boldsymbol{\lambda}} \left[\mathbb{E}_{q_{\boldsymbol{\lambda}_t}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] - \mathbb{H}(q_{\boldsymbol{\lambda}_t}) \right] \quad (6.186)$$

19 where $\tilde{\nabla}_{\boldsymbol{\lambda}}$ denotes the natural gradient. We discuss how to compute these natural gradients in
20 Section 6.4.5. In particular, we can convert it to regular gradients wrt the moment parameters
21 $\boldsymbol{\mu}_t = \boldsymbol{\mu}(\boldsymbol{\lambda}_t)$. This gives

$$\underline{22} \quad \boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t - \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}_t}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] + \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{H}(q_{\boldsymbol{\mu}_t}) \quad (6.187)$$

23 From Equation (6.84) we have

$$\underline{24} \quad \nabla_{\boldsymbol{\mu}} \mathbb{H}(q) = -\boldsymbol{\lambda} - \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\log h(\boldsymbol{\theta})] \quad (6.188)$$

25 Hence the update becomes

$$\underline{26} \quad \boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t - \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}_t}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] - \eta_t \boldsymbol{\lambda}_t - \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\log h(\boldsymbol{\theta})] \quad (6.189)$$

$$\underline{27} \quad = (1 - \eta_t) \boldsymbol{\lambda}_t - \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta}) + \log h(\boldsymbol{\theta})] \quad (6.190)$$

28 For distributions q with constant base measure $h(\boldsymbol{\theta})$, this simplifies to

$$\underline{29} \quad \boldsymbol{\lambda}_{t+1} = (1 - \eta_t) \boldsymbol{\lambda}_t - \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] \quad (6.191)$$

30 Hence at the fixed point we have

$$\underline{31} \quad \boldsymbol{\lambda}_* = (1 - \eta) \boldsymbol{\lambda}_* - \eta \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] \quad (6.192)$$

$$\underline{32} \quad \boldsymbol{\lambda}_* = \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [-\bar{\ell}(\boldsymbol{\theta})] = \tilde{\nabla}_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})} [-\bar{\ell}(\boldsymbol{\theta})] \quad (6.193)$$

1 **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
$$\lambda_* = \nabla_{\mu} \mathbb{E}_{q_*} [-\bar{\ell}(\theta)] = \lambda_0 + \sum_{i=1}^N \underbrace{\nabla_{\mu} \mathbb{E}_{q_*} [\log p(y_i | \theta)]}_{\tilde{\lambda}_i(y_i)} \quad (6.194)$$

7
8
9

10 where $\tilde{\lambda}_i(y_i)$ are the sufficient statistics for the i 'th likelihood term.

11 For models where the joint distribution over the latents factorizes (using a graphical model), we
12 can further decompose this update into a series of local terms. This gives rise to the variational
13 message passing scheme discussed in Section 10.2.7.
14

15 **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 **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
$$\bar{\ell}(\theta) = \sum_{n=1}^N \ell(y_n, f_{\theta}(x_n)) + R(\theta) \quad (6.195)$$

27
28

29 we will instead minimize
30

31
$$\mathcal{L}(\lambda) = \mathbb{E}_{q(\theta|\lambda)} [\bar{\ell}(\theta)] - \mathbb{H}(q(\theta|\lambda)) \quad (6.196)$$

32

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 **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 $q(\theta) = \mathcal{N}(\theta|m, I)$. In this case the natural and moment parameters are equal,
40 $\mu = \lambda = m$. The base measure satisfies
41

42
$$2 \log h(\theta) = -D \log(2\pi) - \theta^T \theta \quad (6.197)$$

43

44 Hence
45

46
$$\nabla_{\mu} \mathbb{E}_q [\log h(\theta)] = \nabla_{\mu} (-D \log(2\pi) - \mu^T \mu - D) = -\mu = -\lambda = -m \quad (6.198)$$

47

1 Thus the BLR update becomes
2

$$\underline{3} \quad \underline{4} \quad \mathbf{m}_{t+1} = (1 - \eta_t) \mathbf{m}_t + \eta_t \mathbf{m}_t - \eta_t \nabla_{\mathbf{m}} \mathbb{E}_{q_{\mathbf{m}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] \quad (6.199)$$

5 We can remove the expectation using the first order delta method (Section 6.6.5):
6

$$\underline{7} \quad \underline{8} \quad \nabla_{\mathbf{m}} \mathbb{E}_{q_{\mathbf{m}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] \approx \nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}} \quad (6.200)$$

9 Putting these together gives the gradient descent update:
10

$$\underline{11} \quad \underline{12} \quad \mathbf{m}_{t+1} = \mathbf{m}_t - \eta_t \nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}_t} \quad (6.201)$$

13 6.8.2.2 Newton's method

14 In this section, we show how to derive Newton's second order optimization method as a special case
15 of BLR, as first shown in [Kha+18].

16 Suppose we assume $q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{m}, \mathbf{S}^{-1})$. The natural parameters are
17

$$\underline{18} \quad \underline{19} \quad \boldsymbol{\lambda}^{(1)} = \mathbf{S}\mathbf{m}, \quad \boldsymbol{\lambda}^{(2)} = -\frac{1}{2}\mathbf{S} \quad (6.202)$$

21 The mean (moment) parameters are
22

$$\underline{23} \quad \underline{24} \quad \boldsymbol{\mu}^{(1)} = \mathbf{m}, \quad \boldsymbol{\mu}^{(2)} = \mathbf{S}^{-1} + \mathbf{m}\mathbf{m}^T \quad (6.203)$$

25 Since the base measure is consant, fom Equation (6.191) we have
26

$$\underline{27} \quad \mathbf{S}_{t+1} \mathbf{m}_{t+1} = (1 - \eta_t) \mathbf{S}_t \mathbf{m}_t - \eta_t \nabla_{\boldsymbol{\mu}^{(1)}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] \quad (6.204)$$

$$\underline{28} \quad \mathbf{S}_{t+1} = (1 - \eta_t) \mathbf{S}_t + 2\eta_t \nabla_{\boldsymbol{\mu}^{(2)}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] \quad (6.205)$$

30 In Section 6.4.5.1 we show that
31

$$\nabla_{\boldsymbol{\mu}^{(1)}} \mathbb{E}_{q(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] = \mathbb{E}_{q(\boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})] - \mathbb{E}_{q(\boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta})] \mathbf{m} \quad (6.206)$$

$$\nabla_{\boldsymbol{\mu}^{(2)}} \mathbb{E}_{q(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] = \frac{1}{2} \mathbb{E}_{q(\boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta})] \quad (6.207)$$

35 Hence the update for the precision matrix becomes
36

$$\underline{37} \quad \underline{38} \quad \mathbf{S}_{t+1} = (1 - \eta_t) \mathbf{S}_t + \eta_t \mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta})] \quad (6.208)$$

39 For the precision weighted mean, we have
40

$$\underline{41} \quad \underline{42} \quad \mathbf{S}_{t+1} \mathbf{m}_{t+1} = (1 - \eta_t) \mathbf{S}_t \mathbf{m}_t - \eta_t \mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})] + \eta_t \mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta})] \mathbf{m}_t \quad (6.209)$$

$$= \mathbf{S}_{t+1} \mathbf{m}_t - \eta_t \mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})] \quad (6.210)$$

44 Hence
45

$$\underline{46} \quad \underline{47} \quad \mathbf{m}_{t+1} = \mathbf{m}_t - \eta_t \mathbf{S}_{t+1}^{-1} \mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})] \quad (6.211)$$

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:

$$\hat{\nabla}_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta}) = \frac{N}{M} \sum_{i \in \mathcal{M}} \nabla_{\boldsymbol{\theta}} \ell(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i)) + \nabla_{\boldsymbol{\theta}} R(\boldsymbol{\theta}) \quad (6.217)$$

$$\hat{\nabla}_{\boldsymbol{\theta}_j}^2 \bar{\ell}(\boldsymbol{\theta}) = \frac{N}{M} \sum_{i \in \mathcal{M}} \nabla_{\boldsymbol{\theta}_j}^2 \ell(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i)) + \nabla_{\boldsymbol{\theta}_j}^2 R(\boldsymbol{\theta}) \quad (6.218)$$

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:

$$\hat{\nabla}_{\boldsymbol{\theta}_j}^2 \bar{\ell}(\boldsymbol{\theta}) \approx \frac{N}{M} \sum_{i \in \mathcal{M}} [\nabla_{\boldsymbol{\theta}_j} \ell(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i))]^2 + \nabla_{\boldsymbol{\theta}_j}^2 R(\boldsymbol{\theta}) \quad (6.219)$$

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

$$\mathbb{E}_{q_t} \left[\hat{\nabla}_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta}) \right] \approx \hat{\nabla}_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta}_t + \boldsymbol{\epsilon}_t) \quad (6.220)$$

where $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \text{diag}(\mathbf{s}_t))$. 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, $q_t(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta} | \boldsymbol{\theta}_t, \mathbf{S}_t^{-1})$, where $\mathbf{S}_t = \text{diag}(\mathbf{s}_t)$ is a vector of precisions. We then use the delta method to eliminate expectations:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \frac{1}{\mathbf{s}_{t+1}} \odot \nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta}_t) \quad (6.221)$$

$$\mathbf{s}_{t+1} = (1 - \eta_t) \mathbf{s}_t + \eta_t \text{diag}(\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta}_t)) \quad (6.222)$$

where \odot is elementwise multiplication. If we allow for different learning rates we get

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha_t \frac{1}{\mathbf{s}_{t+1}} \odot \nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta}_t) \quad (6.223)$$

$$\mathbf{s}_{t+1} = (1 - \beta_t) \mathbf{s}_t + \beta_t \text{diag}(\hat{\nabla}_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta}_t)) \quad (6.224)$$

Now suppose we replace the diagonal Hessian approximation with the sum of the squares per-sample gradients:

$$\text{diag}(\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta}_t)) \approx \hat{\nabla} \bar{\ell}(\boldsymbol{\theta}_t) \odot \hat{\nabla} \bar{\ell}(\boldsymbol{\theta}_t) \quad (6.225)$$

If we also change some scaling factors we can get the RMSprop updates:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \frac{1}{\sqrt{\mathbf{v}_{t+1} + c\mathbf{1}}} \odot \hat{\nabla}_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta}_t) \quad (6.226)$$

$$\mathbf{v}_{t+1} = (1 - \beta) \mathbf{v}_t + \beta [\hat{\nabla} \bar{\ell}(\boldsymbol{\theta}_t) \odot \hat{\nabla} \bar{\ell}(\boldsymbol{\theta}_t)] \quad (6.227)$$

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:

$$\frac{10}{11} \quad \mathcal{L}(\boldsymbol{\lambda}) = \int \bar{\ell}(\boldsymbol{\theta}) q(\boldsymbol{\theta}|\boldsymbol{\lambda}) d\boldsymbol{\theta} \quad (6.228)$$

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.

²³ 6.9 Bayesian optimization

²⁶ In this section, we discuss **Bayesian optimization** or **BayesOpt**, which is a model-based approach
²⁷ to black-box optimization, designed for the case where the objective function $f : \mathcal{X} \rightarrow \mathbb{R}$ is expensive
²⁸ to evaluate (e.g., if it requires running a simulation, or training and testing a particular neural net
²⁹ architecture).

³⁰ Since the true function f is expensive to evaluate, we want to make as few function calls (i.e., make as
³¹ few **queries** \mathbf{x} to the **oracle** f) as possible. This suggests that we should build a **surrogate function**
³² (also called a **response surface model**) based on the data collected so far, $\mathcal{D}_n = \{(\mathbf{x}_i, y_i) : i = 1 : n\}$,
³³ which we can use to decide which point to query next. There is an inherent tradeoff between picking
³⁴ the point \mathbf{x} where we think $f(\mathbf{x})$ is large (we follow the convention in the literature and assume
³⁵ we are trying to maximize f), and picking points where we are uncertain about $f(\mathbf{x})$ but where
³⁶ observing the function value might help us improve the surrogate model. This is another instance of
³⁷ the exploration-exploitation dilemma.

³⁸ In the special case where the domain we are optimizing over is finite, so $\mathcal{X} = \{1, \dots, A\}$, the
³⁹ BayesOpt problem becomes similar to the **best arm identification** problem in the bandit literature
⁴⁰ (Section 36.4). An important difference is that in bandits, we care about the cost of every action we
⁴¹ take, whereas in optimization, we usually only care about the cost of the final solution we find. In
⁴² other words, in bandits, we want to minimize cumulative regret, whereas in optimization we want to
⁴³ minimize simple or final regret.

⁴⁴ Another related topic is **active learning**. Here the goal is to identify the whole function f with
⁴⁵ as few queries as possible, whereas in BayesOpt, the goal is just to identify the maximum of the
⁴⁶ function.

1 Bayesian optimization is a large topic, and we only give a brief overview below. For more details,
2 see e.g., [Sha+16; Fra18; Gar22].
3

4 5 6.9.1 Sequential model-based optimization

6 BayesOpt is an instance of a strategy known as sequential model-based optimization (**SMBO**)
7 [HHLB11]. In this approach, we alternate between querying the function at a point, and updating
8 our estimate of the surrogate based on the new data. More precisely, at each iteration n , we have a
9 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
10 corresponding function values $y_i = f(\mathbf{x}_i) + \epsilon_i$, where ϵ_i is an optional noise term. We use this data to
11 estimate a probability distribution over the true function f ; we will denote this by $p(f|\mathcal{D}_n)$. We then
12 choose the next point to query \mathbf{x}_{n+1} using an **acquisition function** $\alpha(\mathbf{x}; \mathcal{D}_n)$, which computes the
13 expected utility of querying \mathbf{x} . (We discuss acquisition functions in Section 6.9.3). After we observe
14 $y_{n+1} = f(\mathbf{x}_{n+1}) + \epsilon_{n+1}$, we update our beliefs about the function, and repeat. See Algorithm 5 for
15 some pseudocode.
16

17 18 19 Algorithm 5: Bayesian optimization

```
20 1 Collect initial dataset  $\mathcal{D}_0 = \{(\mathbf{x}_i, y_i) : \}$  from random queries  $\mathbf{x}_i$ ;  

21 2 Initialize model  $p(f|\mathcal{D}_0)$  ;  

22 3 for  $n = 1, 2, \dots$  until convergence do  

23 4   Choose next query point  $\mathbf{x}_{n+1} = \operatorname{argmax}_{\mathbf{x} \in \mathcal{X}} \alpha(\mathbf{x}; \mathcal{D}_n)$ ;  

24 5   Measure function value,  $y_{n+1} = f(\mathbf{x}_{n+1}) + \epsilon_n$  ;  

25 6   Augment dataset,  $\mathcal{D}_{n+1} = \{\mathcal{D}_n, (\mathbf{x}_{n+1}, y_{n+1})\}$  ;  

26 7   Update model,  $p(f|\mathcal{D}_{n+1}) \propto p(f|\mathcal{D}_n)p(y_{n+1}|\mathbf{x}_{n+1}, f)$ 
```

27
28
29 This method is illustrated in Figure 6.11. The goal is to find the global optimum of the solid black
30 curve. In the first row, we show the 2 previously queried points, x_1 and x_2 , and their corresponding
31 function values. $y_1 = f(x_1)$ and $y_2 = f(x_2)$. Our uncertainty about the value of f at those locations
32 is 0 (if we assume no observation noise), as illustrated by the posterior credible interval (shaded
33 blue arc) becoming “pinched”. Consequently the acquisition function (shown in green at the bottom)
34 also has value 0 at those previously queried points. The red triangle represents the maximum of the
35 acquisition function, which becomes our next query, x_3 . In the second row, we show the result of
36 observing $y_3 = f(x_3)$; this further reduces our uncertainty about the shape of the function. In the
37 third row, we show the result of observing $y_4 = f(x_4)$. This process repeats until we run out of time,
38 or until we are confident there are no better unexplored points to query.
39

40 The two main “ingredients” that we need to provide to a BayesOpt algorithm are (1) a way to
41 represent and update the posterior surrogate $p(f|\mathcal{D}_n)$, and (2) a way to define and optimize the
42 acquisition function $\alpha(\mathbf{x}; \mathcal{D}_n)$. We discuss both of these topics below.

43 44 6.9.2 Surrogate functions

45 In this section, we discuss ways to represent and update the posterior over functions, $p(f|\mathcal{D}_n)$.

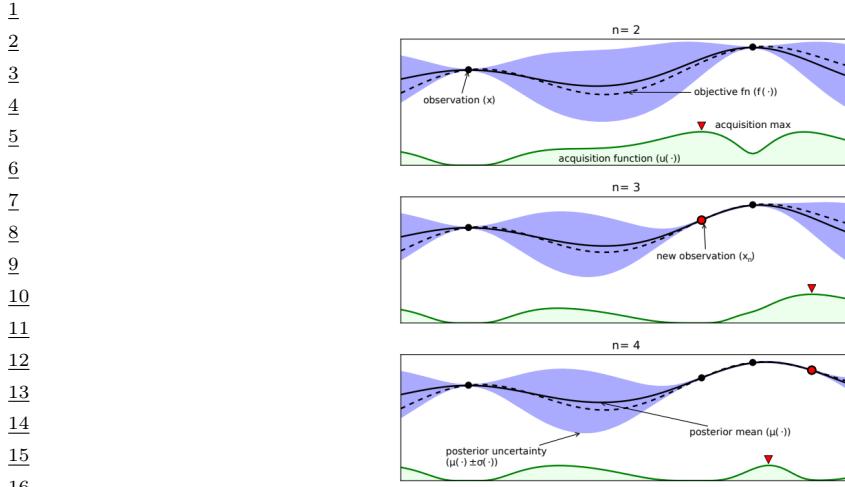


Figure 6.11: Illustration of sequential Bayesian optimization over three iterations. The rows correspond to a training set of size $t = 2, 3, 4$. The solid black line is the true, but unknown, function $f(x)$. The dotted black line is the posterior mean, $\mu(x)$. The shaded blue intervals are the 95% 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 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

$$\alpha_{EI}(\mathbf{x}; \mathcal{D}_n) = \mathbb{E}_{\mathcal{D}_n} [U(\mathbf{x}, y)] = \mathbb{E}_{\mathcal{D}_n} [(f(\mathbf{x}) - V_n)\mathbb{I}(f(\mathbf{x}) > V_n)] \quad (6.231)$$

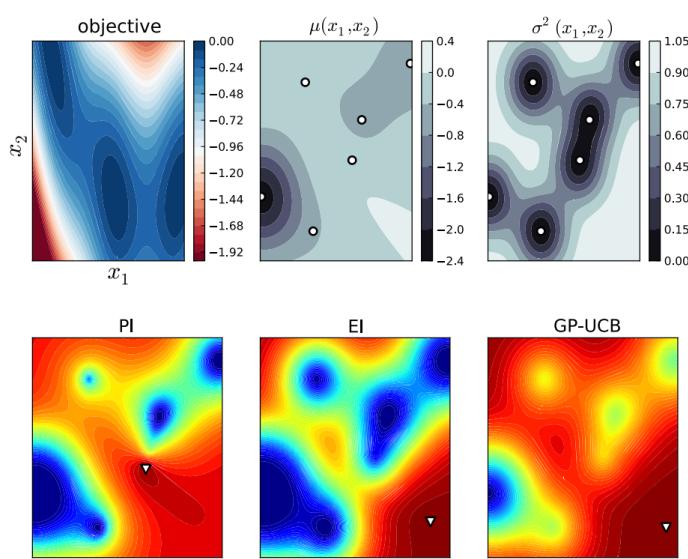


Figure 6.12: The first row shows the objective function, (the Branin function defined on \mathbb{R}^2), and its posterior mean and variance using a GP estimate. White dots are the observed data points. The second row shows 3 different acquisition functions (probability of improvement, expected improvement, 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

$$\underline{4} \quad V_{n+1}(\mathbf{x}, y) = \max_{\mathbf{x}'} \mathbb{E}_{p(f|\mathbf{x}, y, \mathcal{D}_n)} [f(\mathbf{x}')] \quad (6.238)$$

$$\underline{5} \quad V_{n+1}(\mathbf{x}) = \mathbb{E}_{p(y|\mathbf{x}, \mathcal{D}_n)} [V_{n+1}(\mathbf{x}, y)] \quad (6.239)$$

6 We define the KG acquisition function as follows:
7

$$\underline{8} \quad \alpha_{KG}(\mathbf{x}; \mathcal{D}_n) = \mathbb{E}_{\mathcal{D}_n} [(V_{n+1}(\mathbf{x}) - V_n) \mathbb{I}(V_{n+1}(\mathbf{x}) > V_n)] \quad (6.240)$$

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

41 We can formalize such problems by defining the search space in terms of a tree or DAG (directed
42 acyclic graph), where different subsets of the parameters are defined at each leaf. Applying GPs to
43

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

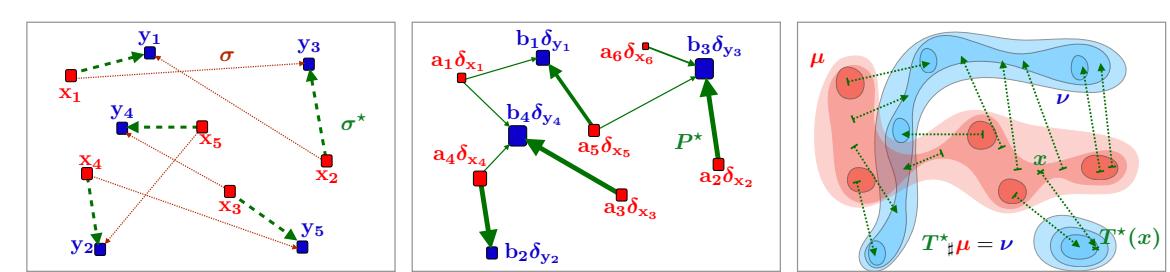


Figure 6.13: (left) Matching a family of 5 points to another is equivalent to considering a permutation in $\{1, \dots, n\}$. When to each pair $(\mathbf{x}_i, \mathbf{y}_j) \in \mathbb{R}^2$ is associated a cost equal to the distance $\|\mathbf{x}_i - \mathbf{y}_j\|$, the optimal matching problem involves 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 .

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:

$$\text{OT}_C(\mathbf{a}, \mathbf{b}) \triangleq \min_{P \in \mathbf{R}_+^{n \times m}, P\mathbf{1}_m = \mathbf{a}, P^T\mathbf{1}_m = \mathbf{b}} \langle P, C \rangle \triangleq \sum_{i,j} P_{ij} C_{ij}. \quad (\text{K})$$

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 statement is equivalent, thanks to the change of variables formula, to ensuring almost everywhere
2 that:

$$\frac{4}{5} \quad q(T(x)) = p(x)|J_T(x)|, \quad (6.242)$$

6 where $|J_T(x)|$ stands for the determinant of the Jacobian matrix of T evaluated at x .

7 Writing $T_\sharp\mu = \nu$ when T does satisfy these conditions, the Monge [Mon81] problem consists in
8 finding the best map T that minimizes the average cost between \mathbf{x} and its displacement $T(\mathbf{x})$,

$$\frac{10}{11} \quad \inf_{T:T_\sharp\mu=\nu} \int_{\mathcal{X}} c(\mathbf{x}, T(\mathbf{x})) \mu(d\mathbf{x}). \quad (\text{M})$$

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.

19

20 6.10.2.3 Kantorovich formulation

22 The Kantorovich problem (K) can also be extended to a continuous setting: Instead of optimizing
23 over a subset of matrices in $\mathbb{R}^{n \times m}$, consider $\Pi(\mu, \nu)$, the subset of joint probability distributions
24 $\mathcal{P}(\mathcal{X} \times \mathcal{X})$ 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

$$\frac{30}{31} \quad \text{OT}_c(\mu, \nu) \triangleq \inf_{\pi \in \Pi(\mu, \nu)} \int_{\mathcal{X}^2} c d\pi. \quad (\text{K2})$$

33 Notice that (K2) subsumes directly (K), since one can check that they coincide when μ, ν are
34 discrete measures, with respective probability weights \mathbf{a}, \mathbf{b} and locations $(\mathbf{x}_1, \dots, \mathbf{x}_n)$ and $(\mathbf{y}_1, \dots, \mathbf{y}_m)$.

36 6.10.2.4 Wasserstein distances

38 When c is equal to a metric d exponentiated by an integer, the optimal value of the Kantorovich
39 problem is called the Wasserstein *distance* between μ and ν :

$$\frac{40}{41} \quad W_p(\mu, \nu) \triangleq \left(\inf_{\pi \in \Pi(\mu, \nu)} \int_{\mathcal{X}^2} d(\mathbf{x}, \mathbf{y})^p d\pi(\mathbf{x}, \mathbf{y}) \right)^{1/p}. \quad (6.244)$$

44 While the symmetry and the fact that $W_p(\mu, \nu) = 0 \Rightarrow \mu = \nu$ are relatively easy to prove provided d
45 is a metric, proving the triangle inequality is slightly more challenging, and builds on a result known
46 as the gluing lemma ([Vil08, p.23]). The p -th power of $W_p(\mu, \nu)$ is often abbreviated as $W_p^p(\mu, \nu)$.

6.10.3 Solving optimal transport

6.10.3.1 Duality and cost concavity

Both (K) and (K2) are linear programs: their constraints and objective functions only involve summations. In that sense they admit a dual formulation (here, again, (DK2) subsumes (DK)):

$$\max_{\mathbf{f} \in \mathbb{R}^n, \mathbf{g} \in \mathbb{R}^m} \mathbf{f}^T \mathbf{a} + \mathbf{g}^T \mathbf{b} \quad (\text{DK})$$

$$\text{subject to } \mathbf{f} \oplus \mathbf{g} \leq C$$

$$\sup_{f \oplus g \leq c} \int_{\mathcal{X}} f \, d\mu + \int_{\mathcal{X}} g \, d\nu; \quad (\text{DK2})$$

where the sign \oplus denotes tensor addition for vectors, $\mathbf{f} \oplus \mathbf{g} = [\mathbf{f}_i + \mathbf{g}_j]_{ij}$, or functions, $f \oplus g : \mathbf{x}, \mathbf{y} \mapsto f(\mathbf{x}) + g(\mathbf{y})$. In other words, the dual problem looks for a pair of vectors (or functions) that attain the highest possible expectation when summed against \mathbf{a} and \mathbf{b} (or integrated against μ, ν), pending the constraint that they do not differ too much across points \mathbf{x}, \mathbf{y} , as measured by c .

The dual problems in (K) and (K2) have two variables. Focusing on the continuous formulation, a closer inspection shows that it is possible, given a function f for the first measure, to compute the best possible candidate for function g . That function g should be as large as possible, yet satisfy the constraint that $g(\mathbf{y}) \leq c(\mathbf{x}, \mathbf{y}) - f(\mathbf{x})$ for all \mathbf{x}, \mathbf{y} , making

$$\forall \mathbf{y} \in \mathcal{X}, \bar{f}(\mathbf{y}) \triangleq \inf_{\mathbf{x}} c(\mathbf{x}, \mathbf{y}) - f(\mathbf{x}), \quad (6.245)$$

the optimal choice. \bar{f} is called the c -transform of f . Naturally, one may choose to start instead from g , to define an alternative c -transform:

$$\forall \mathbf{x} \in \mathcal{X}, \tilde{g}(\mathbf{x}) \triangleq \inf_{\mathbf{y}} c(\mathbf{x}, \mathbf{y}) - g(\mathbf{y}). \quad (6.246)$$

Since these transformations can only improve solutions, one may even think of applying alternatively these transformations to an arbitrary f , to define \bar{f} , \tilde{f} and so on. One can show, however, that this has little interest, since

$$\tilde{\bar{f}} = \bar{f}. \quad (6.247)$$

This remark allows, nonetheless, to narrow down the set of candidate functions to those that have already undergone such transformations. This reasoning yields the so-called set of c -concave functions, $\mathcal{F}_c \triangleq \{f \mid \exists g : \mathcal{X} \rightarrow \mathbb{R}, f = \tilde{g}\}$, which can be shown, equivalently, to be the set of functions f such that $f = \bar{f}$. One can therefore focus our attention to c -concave functions to solve (DK2) using a so-called semi-dual formulation,

$$\sup_{f \in \mathcal{F}_c} \int_{\mathcal{X}} f \, d\mu + \int_{\mathcal{X}} \bar{f} \, d\nu. \quad (\text{DK2})$$

Going from (DK2) to (DK2), we have removed a dual variable g and narrowed down the feasible set to \mathcal{F}_c , at the cost of introducing the highly non-linear transform \bar{f} . This reformulation is, however, very useful, in the sense that it allows to restrict our attention on c -concave functions, notably for two important classes of cost functions c : distances and squared-Euclidean norms.

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

$$\sup_{f \text{1-Lipschitz}} \int_{\mathcal{X}} f \, (d\mu - d\nu). \quad (6.248)$$

⁸ This result has numerous practical applications. This supremum over 1-Lipschitz functions can be
⁹ efficiently approximated using Wavelet coefficients of densities in low-dimensions [SJ08], or heuristically
¹⁰ in more general cases by training neural networks parameterized to be 1-Lipschitz [ACB17] using
¹¹ ReLU activation functions, and bounds on the entries of the weight matrices.
¹²

¹³ **6.10.3.3 Monge maps as gradients of convex functions: the Brenier theorem**

¹⁴ 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
¹⁵ factor $\frac{1}{2}$, to the squared W_2 distance used between densities in an Euclidean space. The remarkable
¹⁶ result, shown first by [Bre91], is that the Monge map solving (M) between two measures for that
¹⁷ cost (taken for granted μ is regular enough, here assumed to have a density w.r.t. Lebesgue measure)
¹⁸ exists and is necessarily the gradient of a convex function. In loose terms, one can show that
¹⁹

$$T^* = \arg \min_{T: T\# \mu = \nu} \int_{\mathcal{X}} \frac{1}{2}\|\mathbf{x} - T(\mathbf{x})\|_2^2 \mu(d\mathbf{x}). \quad (\text{M})$$

²⁰ exists, and is the gradient of a convex function $u : \mathbb{R}^d \rightarrow \mathbb{R}$, namely $T^* = \nabla u$. Conversely, for any
²¹ convex function u , the optimal transport map between μ and the displacement $\nabla u \# \mu$ is necessarily
²² equal to ∇u .

²³ We provide a sketch of the proof: one can always exploit, for any reasonable cost function c
²⁴ (e.g. lower bounded and lower semi continuous), primal-dual relationships: Consider an optimal
²⁵ coupling P^* for (K2), as well as an optimal c -concave dual function f^* for (DK2). This implies in
²⁶ particular that $(f^*, g^* = \bar{f}^*)$ is optimal for (DK2). Complementary slackness conditions for this pair
²⁷ of linear programs imply that if $\mathbf{x}_0, \mathbf{y}_0$ is in the support of P^* , then necessarily (and sufficiently)
²⁸ $f^*(\mathbf{x}_0) + \bar{f}^*(\mathbf{y}_0) = c(\mathbf{x}_0, \mathbf{y}_0)$. Suppose therefore that $\mathbf{x}_0, \mathbf{y}_0$ is indeed in the support of P^* . From the
²⁹ 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,
³⁰ 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
³¹ 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
³² to be $c(\mathbf{x}, \mathbf{y}) = \frac{1}{2}\|\mathbf{x} - \mathbf{y}\|^2$, one has therefore that \mathbf{x}_0 verifies
³³

$$\mathbf{x}_0 \in \operatorname{argmin}_{\mathbf{x}} \frac{1}{2}\|\mathbf{x} - \mathbf{y}_0\|^2 - f^*(\mathbf{x}). \quad (6.249)$$

³⁴ Assuming f^* is differentiable, which one can prove by c -concavity, this yields the identity
³⁵

$$\mathbf{y}_0 - \mathbf{x}_0 - \nabla f^*(\mathbf{x}_0) = 0 \Rightarrow \mathbf{y}_0 = \mathbf{x}_0 - \nabla f^*(\mathbf{x}_0) = \nabla \left(\frac{1}{2}\|\cdot\|^2 - f^* \right) (\mathbf{x}_0). \quad (6.250)$$

³⁶ 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
³⁷ Monge map “disguised” as a coupling, namely
³⁸

$$P^* = (\operatorname{Id}, \nabla \left(\frac{1}{2}\|\cdot\|^2 - f^* \right)) \# \mu. \quad (6.251)$$

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 $T_{\#}\mathcal{N}(\mathbf{m}_1, \Sigma_1) = \mathcal{N}(\mathbf{m}_2, \Sigma_2)$ 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

$$44 \quad W_{c,\gamma}(\mu, \nu) \triangleq \inf_{\pi \in \Pi(\mu, \nu)} \int_{\mathcal{X}^2} d(\mathbf{x}, \mathbf{y})^p \, d\pi(\mathbf{x}, \mathbf{y}) + \gamma D_{\text{KL}}(\pi \| \mu \otimes \nu). \quad (6.256)$$

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].

¹ more than adding it to a larger set. Like convexity, submodularity allows one to efficiently find
² provably optimal or near-optimal solutions. In contrast to convexity, however, where little regarding
³ maximization is guaranteed, submodular functions can be both minimized and (approximately)
⁴ maximized. Submodular maximization and minimization, however, require very different algorithmic
⁵ solutions and have quite different applications. It is sometimes said that submodular functions are
⁶ a discrete form of convexity. This is not quite true, as submodular functions are like both convex
⁷ and concave functions, but also have properties that are similar simultaneously to both convex and
⁸ concave functions at the same time, but then some properties of submodularity are like neither
⁹ convexity nor concavity. Convexity and concavity, for example, can be conveyed even as univariate
¹⁰ functions. This is impossible for submodularity, as submodular functions are defined based only on
¹¹ the response of the function to changes amongst different variables in a multidimensional discrete
¹² space.
¹³

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

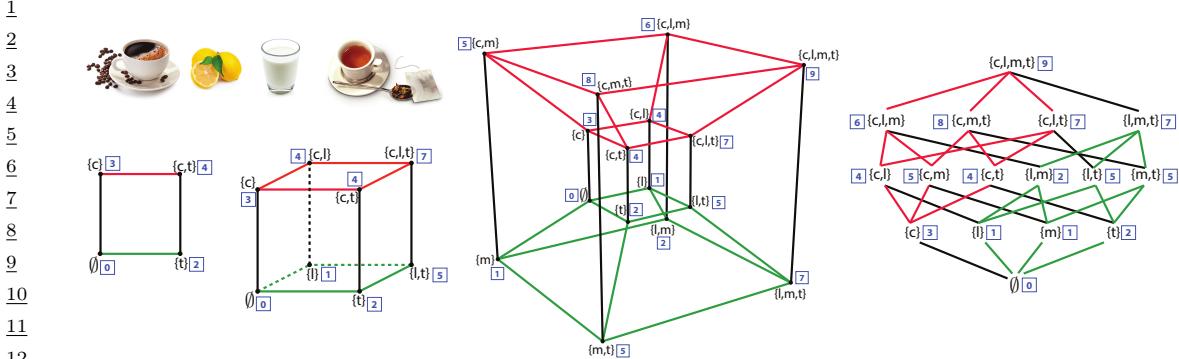


Figure 6.14: The value relationships between coffee (c), lemon (l), milk (m), and tea (t). On the left, we first see a simple square showing the relationships between coffee and tea, and see that they are substitutive (or submodular). In this, and all of the shapes, the vertex label set is indicated 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.

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:

$$\text{²⁷ } f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y) \quad (6.261)$$

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:

$$\text{³⁸ } f(X + v) + f(X) \geq f(Y + v) + f(Y) \quad (6.262)$$

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

²⁹ While the facility location looks quite different from a feature based function, it is possible
³⁰ to precisely represent any facility location function with a feature based function. Consider
³¹ 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
³² $\max_{a \in A} x_a = \sum_{i=1}^n y_i \min(|A \cap \{i, i+1, \dots, n\}|, 1)$ where $y_i = x_i - x_{i-1}$ and we set $x_0 = 0$. We note
³³ that this is a sum of weighted concave composed with modular functions since $\min(\alpha, 1)$ is concave
³⁴ in α , and $|A \cap \{i, i+1, \dots, n\}|$ is a modular function in A . Thus, the facility location function, a
³⁵ sum of these, is merely a feature based function.

³⁶ Feature based functions, in fact, are quite expressive, and can be used to represent many different
³⁷ submodular functions including set cover and graph-based functions. For example, we can define a *set*
³⁸ *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 $U = \bigcup_{v \in V} U_v$
³⁹ then X indexes a set that fully covers U . This can also be represented as $f(X) = \sum_{u \in U} \min(1, m_u(X))$
⁴⁰ where $m_u(X)$ is a modular function where $m_u(v) = 1$ if and only if $u \in U_v$ and otherwise $m_u(v) = 0$.
⁴¹ We see that this is a feature based submodular function since $\min(1, x)$ is concave in x , and U is a
⁴² set of features.

⁴³ This construct can be used to produce the vertex cover function if we set $U = V$ to be the set of
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 .

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 $\mathbb{P}[B_{u,v} = 1]$ 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 $\mathbb{P}[B_{u,v} = 1] = 1 - \mathbb{P}[B_{u,v} = 0]$. Then we can define the probabilistic coverage function as $f(X) = \sum_{u \in U} f_u(X)$ where, for feature u , we have $f_u(X) = 1 - \prod_{v \in X} (1 - \mathbb{P}[B_{u,v} = 1])$ which indicates the degree to which feature u is “covered” by X . If we set $\mathbb{P}[B_{u,v} = 1] = 1$ if and only if $u \in U_v$ and otherwise $\mathbb{P}[B_{u,v} = 1] = 0$, 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 $\mathbb{P}[B_{u,v} = 1]$ 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 $1 - \prod_{v \in X} (1 - \mathbb{P}[B_{u,v} = 1]) = 1 - \exp(-m_u(X)) = \phi(m_u(X))$ where m_u is a modular function with evaluation $m_u(X) = \sum_{v \in X} \log(1/(1 - \mathbb{P}[B_{u,v} = 1]))$ 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:

$$f(X) = \sum_{i \in X, j \in \bar{X}} w_{i,j} = \sum_{i,j \in V} w_{i,j} \mathbf{1}\{i \in X, j \in \bar{X}\} \quad (6.263)$$

$$= \frac{1}{2} \sum_{i,j \in V} w_{i,j} \min(|X \cap \{i, j\}|, 1) + \frac{1}{2} \sum_{i,j \in V} w_{i,j} \min(|(\bar{X}) \cap \{i, j\}|, 1) - \frac{1}{2} \sum_{i,j \in V} w_{i,j} \quad (6.264)$$

$$= \tilde{f}(X) + \tilde{f}(\bar{X}) - \tilde{f}(V) \quad (6.265)$$

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 $\tilde{f}(\bar{X})$ (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 $f(X) = w(\{(a, b) \in E : a \in X, b \in \bar{X}\})$. We view $\{(a, b) \in E : a \in X, b \in \bar{X}\}$ 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 **6.11.4 Submodular Optimization**

20 Submodular functions, while discrete, would not be very useful if it was not possible to optimize
21 over them efficiently. There are many natural problems in machine learning that can be cast as
22 submodular optimization and that can be addressed relatively efficiently.

23 When one wishes to encourage diversity, information, spread, high complexity, independence,
24 coverage, or dispersion, one usually will maximize a submodular function, in the form of $\max_{A \in \mathcal{C}} f(A)$
25 where $\mathcal{C} \subseteq 2^V$ is a constraint set, a set of subsets we are willing to accept as feasible solutions (more
26 on this below).

27 Why is submodularity, in general, a good model for diversity? Submodular functions are such
28 that once you have some elements, any other elements not in your possession but that are similar
29 to, explained by, or represented by the elements in your possession become less valuable. Thus, in
30 order to maximize the function, one must choose other elements that are dissimilar to, or not well
31 represented by, the ones you already have. That is, the elements similar to the ones you own are
32 diminished in value relative to their original values, while the elements dissimilar to the ones you
33 have do not have diminished value relative to their original values. Thus, maximizing a submodular
34 function successfully involves choosing elements that are jointly dissimilar amongst each other, which
35 is a definition of diversity. Diversity in general is a critically important aspect in machine learning
36 and artificial intelligence. For example, bias in data science and machine learning can often be seen
37 as some lack of diversity somewhere. Submodular functions have the potential to encourage (and
38 even ensure) diversity, enhance balance, and reduce bias in artificial intelligence.

39 Note that in order for a submodular function to appropriately model diversity, it is important
40 for it to be instantiated appropriately. Figure 6.15 shows an example in two dimensions. The plot
41 compares the ten points chosen according to a facility location instantiated with a Gaussian kernel,
42 along with the random samples of size ten. We see that the facility location selected points are more
43 diverse and tend to cover the space much better than any of the randomly selected points each of
44 which miss large regions of the space and/or show cases where points near each other are jointly
45

46

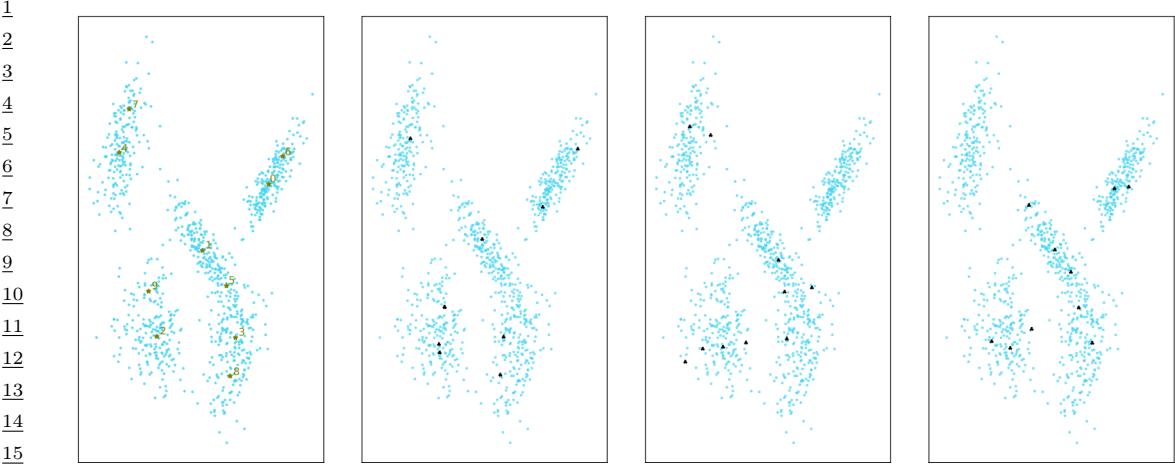


Figure 6.15: Far Left: cardinality constrained (to ten) submodular maximization of a facility location function over 1000 points in two dimensions. Similarities are based on a Gaussian kernel $\text{sim}(a, v) = \exp(-d(a, v))$ where $d(\cdot, \cdot)$ is a distance. Selected points are green stars and 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

⁴³ Then (V, \mathcal{I}_p) is a matroid. The corresponding submodular maximization problem is a natural
⁴⁴ generalization of the cardinality constraint in that, rather than having a fixed number of elements
⁴⁵ beyond which we are uninterested, the set of elements V is organized into groups, and here we have
⁴⁶

⁴⁷

1 a fixed per-group limit beyond which we are uninterested. This is useful for fairness applications
2 since the solution must be distributed over the blocks of the matroid. Still, there are many much
3 more powerful types of matroids that one can use [Oxl11; GM12].

4 Regardless of the matroid, the problem $\max_{X \in \mathcal{I}} f(X)$ can be solved, with a $1/2$ approximation
5 factor, using the same greedy algorithm as above [NWF78; FNW78]. Indeed, the greedy algorithm
6 has an intimate relationship with submodularity, a fact that is well studied in some of the seminal
7 works on submodularity [Edm70; Lov83; Sch04]. It is also possible to define constraints consisting
8 of an *intersection of matroids*, meaning that the solution must be simultaneously independent in
9 multiple distinct matroids. Adding on to this, we might wish a set to be independent in multiple
10 matroids and also satisfy a knapsack constraint. Knapsack constraints are not matroid constraints,
11 since there can be multiple maximal cost solutions that are not the same size (as must be the case in
12 a matroid). It is also possible to define discrete constraints using level sets of another completely
13 different submodular function [IB13] — given two submodular functions f and g , this leads to
14 optimization problems of the form $\max_{X \subseteq V: g(X) \leq \alpha} f(X)$ (the submodular cost submodular knapsack,
15 or SCSK, problem) and $\min_{X \subseteq V: g(X) \geq \alpha} f(X)$ (the submodular cost submodular cover, or SCSC,
16 problem). Other examples include covering constraints [IN09], and cut constraints [JB16a]. Indeed,
17 the type of constraints on submodular maximization for which good and scalable algorithms exist is
18 quite vast, and still growing.

19 One last note on submodular maximization. In the above, the function f has been assumed to be
20 a polymatroid function. There are many submodular functions that are not monotone [Buc+12].
21 One example we saw before, namely the graph cut function. Another example is the log of the
22 determinant (log-determinant) of a submatrix of a positive-definite matrix (which is the Gaussian
23 entropy plus a constant). Suppose that \mathbf{M} is an $n \times n$ symmetric positive-definite (SPD) matrix,
24 and that \mathbf{M}_X is a row-column submatrix (i.e., it is an $|X| \times |X|$ matrix consisting of the rows and
25 columns of \mathbf{M} consisting of the elements in X). Then the function defined as $f(X) = \log \det(\mathbf{M}_X)$
26 is submodular but not necessarily monotone non-decreasing. In fact, the submodularity of the
27 log-determinant function is one of the reasons that *determinantal point processes* (DPPs), which
28 instantiate probability distributions over sets in such a way that high probability is given to those
29 subsets that are diverse according to \mathbf{M} , are useful for certain tasks where we wish to probabilistically
30 model diversity [KT11a]. Diversity of a set X here is measured by the volume of the parallelepiped
31 which is known to be computed as the determinant of the submatrix \mathbf{M}_X and taking the log of this
32 volume makes the function submodular in X . A DPP in fact is an example of a log-submodular
33 probabilistic model (more in Section 6.11.10).

35 6.11.4.3 Submodular Function Minimization

36 In the case of a polymatroid function, unconstrained minimization is again trivial. However, even in
37 the unconstrained case, the minimization of an arbitrary (i.e., not necessarily monotone) submodular
38 function $\min_{X \subseteq V} f(X)$ might seem hopelessly intractable. Unconstrained submodular maximization
39 is NP-hard (albeit approximable) and this is not surprising given that there are an exponential
40 number of sets needing to be considered. Remarkably, submodular minimization does not require
41 exponential computation, is not NP-hard, and in fact, there are polynomial time algorithms for
42 doing so, something that is not at all obvious. This is one of the important characteristics that
43 submodular functions share with convex functions, their common amenability to minimization.
44 Starting in the very late 1960s, and spearheaded by individuals such as Jack Edmonds [Edm70],
45

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

26 6.11.5 Applications of Submodularity in Machine Learning and AI

27 Submodularity arises naturally in applications in machine learning and artificial intelligence, but its
28 utility has still not yet been as widely recognized and exploited as other techniques. For example,
29 while information theoretic concepts like entropy and mutual information are extremely widely used
30 in machine learning (e.g., the cross entropy loss for classification is ubiquitous), the submodularity
31 property of entropy is not nearly as widely explored.
32

33 Still, the last several decades, submodularity has been increasingly studied and utilized in the
34 context of machine learning. The below we begin to provide only a brief survey of some of the major
35 sub-areas within machine learning that have been touched by submodularity. The list is not meant
36 to be exhaustive, or even extensive. It is hoped that the below should, at least, offer a reasonable
37 introduction into how submodularity has been and can continue to be useful in machine learning and
38 artificial intelligence.

39 6.11.6 Sketching, CoreSets, Distillation, and Data Subset & Feature Selection

40 A summary is a concise representation of a body of data that can be used as an effective and efficient
41 substitute for that data. There are many types of summary and some are extremely simple. For
42 example, the mean or median of a list of numbers summarizes some property (the central tendency)
43 of that list. A random subset is also a form of summary.
44

45 _____
46 6. This is the same Wolfe as the Wolfe in Frank-Wolfe but not the same algorithm.
47

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 coreset
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 coreset 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 coreset 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 coreset 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

1 use of a deep submodular function and the latter does this in an unsupervised label-free fashion);
2 [SS18a] which is a form of robust submodularity for producing coresets for training CNNs; [Kau+19]
3 which uses a facility location to facilitate diversity selection in active learning; [Bai+15; CTN17]
4 which develops a mixture of submodular functions for document summarization where the mixture
5 coefficients are also included in the hyperparameter set; [Xu+15] uses a symmetrized submodular
6 function for the purposes of video summarization.
7

8 The learnability and identifiability of submodular functions has received a good amount of study
9 from a theoretical perspective. Starting with the strictest learning settings, the problem looks pretty
10 dire. For example, [SF08; Goe+09] shows that if one is restricted to making a polynomial number of
11 queries (i.e., training pairs of the form $(S, f(S))$) of a monotone submodular function, then it is not
12 possible to approximate f with a multiplicative approximation factor better than $\tilde{\Omega}(\sqrt{n})$. In [BH11],
13 goodness is judged multiplicatively, meaning for a set $A \subseteq V$ we wish that $\tilde{f}(A) \leq f(A) \leq g(n)f(A)$
14 for some function $g(n)$, and this is typically a probabilistic condition (i.e., measured by distribution,
15 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,
16 goodness may also be measured by an additive approximation error, say by a norm. I.e., defining
17 $\text{err}_p(f, \tilde{f}) = \|f - \tilde{f}\|_p = (E_{A \sim \mathbf{Pr}}[|f(A) - \tilde{f}(A)|^p])^{1/p}$, we may wish $\text{err}_p(f, \tilde{f}) < \epsilon$ for $p = 1$ or $p = 2$.
18 In the PAC (probably approximately correct) model, we probably ($\delta > 0$) approximately ($\epsilon > 0$ or
19 $g(n) > 1$) learn ($\beta = 0$) with a sample or algorithmic complexity that depends on δ and $g(n)$. In the
20 PMAC (probably mostly approximately correct) model [BH11], we also “mostly” $\beta > 0$ learn. In some
21 cases we wish to learn the best submodular approximation to a non-submodular function. In other
22 cases, we are allowed to deviate from submodularity as long as the error is small. Learning special
23 cases includes coverage functions [FK14; FK13a], and low-degree polynomials [FV15], curvature
24 limited functions [IJB13], functions with a limited “goal” [DHK14; Bac+18], functions that are
25 Fourier sparse [Wen+20a], or that are of a family called “juntas” [FV16], or that come from families
26 other than submodular [DFF21], and still others [BRS17; FKV14; FKV17; FKV20; FKV13; YZ19].
27 Other results include that one can not minimize a submodular function by learning it first from
28 samples [BS17]. The essential strategy of learning is to attempt to construct a submodular function
29 approximation \hat{f} from an underlying submodular function f querying the latter only a small number
30 of times. The overall gist of these results is that it is hard to learn everywhere and accurately.

31 In the machine learning community, learning can be performed extremely efficiently in practice,
32 although there are not the types of guarantees as one finds above. For example, given a mixture
33 of submodular components of the form $f(A) = \sum_i \alpha_i f_i(A)$, if each f_i is considered fixed, then the
34 learning occurs only over the mixture coefficients α_i . This can be solved as a linear regression problem
35 where the optimal coefficients can be computed in a linear regression setting. Alternatively, such
36 functions can be learnt in a max-margin setting where the goal is primarily to adjust α_i to ensure
37 that $f(A)$ is large on certain subsets [SSJ12; LB12; Tsc+14]. Even here there are practical challenges,
38 however, since it is in general hard in practice to obtain a training set of pairs $\{(S_i, F(S_i))\}_i$.
39 Alternatively, one also “learn” a submodular function in a reinforcement learning setting [CKK17] by
40 optimizing the implicit function directly from gain vectors queried from an environment. In general,
41 such practical learning algorithms have been used for image summarization [Tsc+14], document
42 summarization [LB12], and video summarization [GGG15; Vas+17a; Gon+14; SGS16; SLG17]. While
43 none of these learning approaches claim to approximate some true underlying submodular function,
44 in practice, they do perform better than the by-hand crafting of a submodular functions mentioned
45 above.

46 By a submodularity based coresset, we mean one where the direct optimization of a submodular
47

function offers a theoretical guarantee for some specific problem. This is distinct from above where the submodular function is used as a surrogate heuristic objective function and for which, even if the submodular function is learnt, optimizing it is only a heuristic for the original problem. In some limited cases, it can be shown that the function we wish to approximate is already submodular, e.g., in the case of certain naive Bayes and k-NN classifiers [WIB15] where the training accuracy, as a function of the training data subset, can be shown to be submodular. Hence, maximizing this function offers the same guarantee on the training accuracy as it does on the submodular function. Unfortunately, the accuracy function for many models are not submodular, although they do have a difference of submodular [NB05; IB12] decomposition.

In other cases, it can be shown that certain desirable coresets objectives are inherently submodular. For example, in [MBL20], it is shown that the normed difference between the overall gradient (from summing over all samples in the training data) and an approximate gradient (from summing over only samples in a summary) can be upper bounded with a supermodular function that, when converted to a submodular facility location function and maximized, will select a set that reduces this difference, and will lead to similar convergence rates to an approximate optimum solution in the convex case. A similar example of this in a DPP context is shown in [TBA19]. In other cases, subsets of the training data and training occur simultaneously using a continuous-discrete optimization framework, where the goal is to minimize the loss on diverse and challenging samples measured by a submodular objective [ZB18]. In still other cases, bi-level objectives related to but not guaranteed to be submodular can be formed where a set is selected from a training set with the deliberate purpose of doing well on a validation set [Kil+20; BMK20].

The methods above have focused on reducing the number of samples in a training data. Considering the transpose of a design matrix, however, all of the above methods can be used for reducing the features of a machine learning procedure as well. Specifically, any of the extractive summarization, subset selection, or coresets methods can be seen as feature selection while any of the abstract summarization, sketching, or distillation approaches can be seen as dimensionality reduction.

28

29 6.11.7 Combinatorial Information Functions

30

31 The entropy function over a set of random variables X_1, X_2, \dots, X_n is defined as $H(X_1, X_2, \dots, X_n) =$
32 $-\sum_{x_1, x_2, \dots, x_n} p(x_1, \dots, x_n) \log p(x_1, \dots, x_n)$. From this we can define three set-argument conditional
33 mutual information functions as $I_H(A; B|C) = I(X_A; X_B|X_C)$ where the latter is the mutual
34 information between variables indexed by A and B given variables indexed by C . This mutual
35 information expresses the residual information between X_A and X_B that is not explained by their
36 common information with X_C .

37 As mentioned above, we may view any polymatroid function as a type of information function over
38 subsets of V . That is, $f(A)$ is the information in set A — to the extent that this is true, this property
39 justifies f 's use as a summarization objective as mentioned above. The reason f may be viewed as an
40 information function stems from f being normalized, f 's non-negativity, f 's monotonicity, and the
41 property that further conditioning reduces valuation (i.e., $f(A|B) \geq f(A|B, C)$ which is identical to
42 the submodularity property). These properties were outlined as being essential to the entropy function
43 in Shannon's original work [Sha48] but are true of any polymatroid function as well. Hence, given any
44 polymatroid function f , it is possible to define a combinatorial mutual information function [Iye+21]
45 in a similar way. Specifically, we can define the combinatorial (submodular) conditional mutual
46 information (CCMI) as $I_f(A; B|C) = f(A+C) + f(B+C) - f(C) - f(A+B+C)$, which has been
47

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].

9 Yet another form of clustering can be seen via the simple cardinality constrained submodular
10 maximization process itself which can be compared to a k -medoids process whenever the objective f
11 is the facility location function. Hence, any such submodular function can be seen as a submodular-
12 function-parameterized form of finding the k “centers” among a set of data items. There have been
13 numerous applications of submodular clustering. For example, using these techniques it is possible to
14 identify parcellations of the human brain [Sal+17a]. Other applications include partitioning data for
15 more effective and accurate and lower variance distributed machine learning training [Wei+15a] and
16 also for more ideal mini-batch construction for training deep neural networks [Wan+19b].

17

18 6.11.9 Active and Semi-Supervised Learning

19 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
20 are unknown. Samples are labeled one at a time or one mini-batch at a time, and after each
21 labeling step t each remaining unlabeled sample is given a score $s_t(x_i)$ that indicates the potential
22 benefit of acquiring a label for that sample. Examples include the entropy of the model’s output
23 distribution on x_i , or a margin based score consisting of the difference between the top and the
24 second-from-the-top posterior probability (what is known as margin sampling [Set09]). This produces
25 a modular function on the unlabeled samples, $m_t(A) = \sum_{a \in A} s_t(x_a)$ where $A \subseteq V$. It is simple to use
26 this modular function to produce a mini-batch active learning procedure where at each stage we form
27 $A_t \in \operatorname{argmax}_{A \subseteq U_t: |A|=k} m_t(A)$ where U_t is the set of labeled samples at stage t . Then A_t is a set of
28 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
29 using active learning with mini-batches of size greater than one is that it is often inefficient to ask for
30 single label at a time. The problem which such a minibatch strategy, however, is that the set A_t
31 can be redundant. The reason is that the uncertainty about every sample in A_t could be owing to
32 the same underlying cause — even though the model is most uncertain about samples in A_t , once
33 one sample in A_t is labeled, it may not be optimal to label the remaining samples in A_t due to this
34 redundancy. Utilizing submodularity, therefore, can help reduce this redundancy. Suppose $f_t(A)$ is
35 a submodular diversity model over samples at step t . At each stage, choosing the set of samples
36 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
37 of both uncertainty (via $m_t(A)$) and diversity (via $f_t(A)$). This is precisely the submodular active
38 learning approach taken in [WIB15; Kau+19].

39 Another quite different approach to a form of submodular “batch” active learning setting where a
40 batch L of labeled samples are selected all at once and then used to label the rest of the unlabeled
41 samples. This also allows the remaining unlabeled samples to be utilized in a semi-supervised
42 framework [GB09; GB11]. In this setting, we start with a graph $G = (V, E)$ where the nodes V
43 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
44 the labels just for node set A . We also define $V(y) \subseteq V$ as $V(y) = \{v \in V : y_v = 1\}$. Hence
45

46

$V(y)$ 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 CCMF 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 unsatisfactory 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.

18

19 6.11.10 Probabilistic Modeling

20 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.

41 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 42 small valuation by a submodular function. Therefore, these distributions have high probability when 43 x consists of a homogeneous and for this reason they are useful for computer vision segmentation 44 problems (e.g., in a segment of an image, the nearby pixels should roughly be homogeneous as 45 that is often what defines an object). The DPPs we saw above, however, are an example of a 46

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

⁴ More generally, $E(x)$ being either a submodular or supermodular function can produce log-
⁵ submodular or log-supermodular distributions, covering both cases above where the partition function
⁶ takes the form $Z = \sum_{A \subseteq V} \exp(f(A))$ for objective f . Moreover, we often wish to perform tasks much
⁷ more than just finding the most probable random variable assignments. This includes marginalization,
⁸ computing the partition function, constrained maximization, and so on. Unfortunately, many of these
⁹ more general probabilistic inference problems do not have polynomial time solutions even though
¹⁰ the objectives are submodular or supermodular. On the other hand, such structure has opened the
¹¹ doors to an assortment of new probabilistic inference procedures that exploit this structure [DK14;
¹² DK15a; DTK16; ZDK15; DJK18]. Most of these methods were of the variational sort and offered
¹³ bounds on the partition function Z , sometimes making use of the fact that submodular functions
¹⁴ have easily computable semi-gradients [IB15; Fuj05] which are modular upper and lower bounds on a
¹⁵ submodular or supermodular function that are tight at one or more subsets. Given a submodular (or
¹⁶ supermodular) function f and a set A , it is possible to easily construct (in linear time) a modular
¹⁷ function upperbound $m^A : 2^V \rightarrow \mathbb{R}$ and a modular function lower bound $m_A : 2^V \rightarrow \mathbb{R}$ having
¹⁸ 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
¹⁹ $m_A(A) = f(A) = m^A(A)$ [IB15]. For any modular function m , the probability function for a
²⁰ characteristic vector $x = \mathbf{1}_A$ becomes $p(\mathbf{1}_A) = 1/Z \exp(E(\mathbf{1}_A)) = \prod_{a \in A} \sigma(m(a)) \prod_{a \notin A} \sigma(-m(a))$
²¹ where σ is the logistic function. Thus, a modular approximation of a submodular function is like a
²² mean-field approximation of the distribution, and makes the assumption that all random variables
²³ are independent. Such an approximation can then be used to compute quantities such as upper and
²⁴ 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

1 due to continuous optimization of x with a penalty term of the form $f(V(x))$. To address this, we can
2 use the Lovász extension $\check{f}(x)$ on a vector x . This function is convex but it is not a norm, but if we
3 consider the construct defined as $\|x\|_f = \check{f}(|x|)$, it can be shown that this satisfies all the properties
4 of a norm for all non-trivial submodular functions [PG98; Bac+13] (i.e., those normalized submodular
5 functions for which $f(v) > 0$ for all v). In fact, the group lasso mentioned above is a special case
6 for a particularly simple feature-based submodular function (a sum of min-truncated cardinality
7 functions). But in principle, the same submodular design strategies mentioned in Section 6.11.6 can
8 be used to produce a submodular function to instantiate an appropriate convex structured norm for
9 a given machine learning problem.
10

11 **6.11.12 Conclusions**

12 We have only barely touched the surface of submodularity and how it applies to and can benefit
13 machine learning. For more details, see [Bil22] and the many references contained therein. Considering
14 once again the innocuous looking submodular inequality, then very much like the definition of
15 convexity, we observe something that belies much of its complexity while opening the gates to wide
16 and worthwhile avenues for machine learning exploration.
17

18 **6.12 Derivative free optimization**

19 **Derivative free optimization** or **DFO** refers to a class of techniques for optimizing functions
20 without using derivatives. This is useful for blackbox function optimization as well as discrete
21 optimization. If the function is expensive to evaluate, we can use Bayesian optimization (Section 6.9).
22 If the function is cheap to evaluate, we can use **stochastic local search** methods or **evolutionary**
23 **search** methods. To save space, we discuss these in the supplementary material.
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.

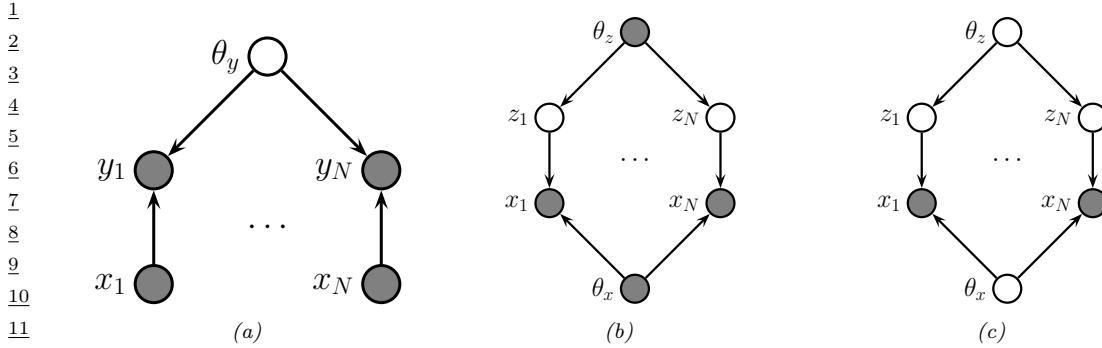


Figure 7.1: Graphical models with (a) Global hidden variables θ_y , (b) Local hidden variables $z_{1:N}$, (c) Local and global hidden variables. In all cases, $\mathbf{x}_{1:N}$ are the observed variables.

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

$$p(\mathbf{y}_{1:N}, \boldsymbol{\theta} | \mathbf{x}_{1:N}) = p(\boldsymbol{\theta}) \left[\prod_{n=1}^N p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta}) \right] \quad (7.4)$$

The goal is to compute the posterior $p(\boldsymbol{\theta} | \mathbf{x}_{1:N}, \mathbf{y}_{1:N})$. 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 $\boldsymbol{\theta}$ are known. This is shown in Figure 7.1b. Now the joint distribution has the form

$$p(\mathbf{x}_{1:N}, \mathbf{z}_{1:N} | \boldsymbol{\theta}) = \left[\prod_{n=1}^N p(\mathbf{x}_n | \mathbf{z}_n, \boldsymbol{\theta}_x) p(\mathbf{z}_n | \boldsymbol{\theta}_z) \right] \quad (7.5)$$

The goal is to compute $p(\mathbf{z}_n | \mathbf{x}_n, \boldsymbol{\theta})$ 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 $\boldsymbol{\theta}$, 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 $p(\mathbf{z}_n | \mathbf{x}_n, \boldsymbol{\theta}_t)$ in the E step for all n simultaneously, and then update $\boldsymbol{\theta}_t$ in the M step. If the inference of \mathbf{z}_n

1 cannot be done exactly, we can use variational inference, a combination known variational EM
2 (Section 6.7.6.1).
3

4 Alternatively, we can use a minibatch approximation to the likelihood, marginalizing out \mathbf{z}_n for
5 each example in the minibatch to get

$$\log p(\mathcal{D}_t | \boldsymbol{\theta}_t) = \sum_{n \in \mathcal{D}_t} \log \left[\sum_{\mathbf{z}_n} p(\mathbf{x}_n, \mathbf{z}_n | \boldsymbol{\theta}_t) \right] \quad (7.6)$$

6 where \mathcal{D}_t is the minibatch at step t . If the marginalization cannot be done exactly, we can use
7 variational inference, a combination known as stochastic variational inference (Section 10.3.2). We
8 can also learn an inference network $q_\phi(\mathbf{z} | \mathbf{x}; \boldsymbol{\theta})$ to perform the inference for us, rather than running
9 an inference engine for each example n in each batch t ; the cost of learning ϕ can be amortized
10 across the batches. This is called amortized SVI, and is commonly used to train deep latent variable
11 models such as VAEs (Section 22.2).

12 7.2.3 Global and local latents

13 The third pattern arises when we need to perform inference in models which have **local and global**
14 **latent variables**. This is shown in Figure 7.1c, and corresponds to the following joint distribution:
15

$$p(\mathbf{x}_{1:N}, \mathbf{z}_{1:N}, \boldsymbol{\theta}) = p(\boldsymbol{\theta}_x)p(\boldsymbol{\theta}_z) \left[\prod_{n=1}^N p(\mathbf{x}_n | \mathbf{z}_n, \boldsymbol{\theta}_x)p(\mathbf{z}_n | \boldsymbol{\theta}_z) \right] \quad (7.7)$$

16 This is essentially a Bayesian version of the latent variable model in Figure 7.1b, where now we model
17 uncertainty in both the local variables \mathbf{z}_n and the shared global variables $\boldsymbol{\theta}$. This approach is less
18 common in the ML community, since it is often assumed that the uncertainty in the parameters $\boldsymbol{\theta}$
19 is negligible compared to the uncertainty in the local variables \mathbf{z}_n . The reason for this is that the
20 parameters are “informed” by all N data cases, whereas each local latent \mathbf{z}_n is only informed by
21 a single data point, namely \mathbf{x}_n . Nevertheless, there are advantages to being “fully Bayesian”, and
22 modeling uncertainty in both local and global variables. We will see some examples of this later in
23 the book.

24 7.3 Exact inference algorithms

25 In some cases, we can perform example posterior inference in a tractable manner. In particular, if
26 the prior is **conjugate** to the likelihood, the posterior will be analytically tractable. In general, this
27 will be the case when the prior and likelihood are from the same exponential family. In particular, if
28 the unknown variables are represented by $\boldsymbol{\theta}$, then we assume

$$p(\boldsymbol{\theta}) \propto \exp(\boldsymbol{\lambda}_0^\top \mathcal{T}(\boldsymbol{\theta})) \quad (7.8)$$

$$p(\mathbf{y}_i | \boldsymbol{\theta}) \propto \exp(\tilde{\boldsymbol{\lambda}}_i(\mathbf{y}_i)^\top \mathcal{T}(\boldsymbol{\theta})) \quad (7.9)$$

29 We can then compute the posterior by just adding the natural parameters:
30

$$p(\boldsymbol{\theta} | \mathbf{y}_{1:N}) = \exp(\boldsymbol{\lambda}_*^\top \mathcal{T}(\boldsymbol{\theta})) \boldsymbol{\lambda}_* = \boldsymbol{\lambda}_0 + \sum_n n = 1^N \tilde{\boldsymbol{\lambda}}_n(\mathbf{y}_n) \quad (7.10)$$

¹ See Section 3.2 for details.
² Another setting where we can compute the posterior exactly arises when the D unknown variables
³ are all discrete, each with K states; in this case, the integral for the normalizing constant becomes a
⁴ sum with K^D terms. In many cases, K^D will be too large to be tractable. However, if the distribution
⁵ satisfies certain conditional independence properties, as expressed by a probabilistic graphical model
⁶ (PGM), then we can write the joint as a product of local terms (see Chapter 4 and Chapter 4).
⁷ This lets us use dynamic programming to make the computation tractable. We discuss this idea for
⁸ chain-structured PGMs in Chapter 8, and for general graphs in Chapter 9.

⁹

¹⁰ 7.4 Approximate inference algorithms

¹¹

¹² For most probability models, we will not be able to compute marginals or posteriors exactly, so we
¹³ must resort to using **approximate inference**. There are many different algorithms, which trade off
¹⁴ speed, accuracy, simplicity, and generality. We briefly discuss some of these algorithms below. We
¹⁵ give more detail in the following chapters.

¹⁶

¹⁷ 7.4.1 MAP estimation

¹⁸

¹⁹ The simplest approximate inference method is to compute the MAP estimate

$$\hat{\boldsymbol{\theta}} = \operatorname{argmax}_{\boldsymbol{\theta}} p(\boldsymbol{\theta} | \mathcal{D}) = \operatorname{argmax}_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}) + \log p(\mathcal{D} | \boldsymbol{\theta}) \quad (7.11)$$

²⁰ and then to assume that the posterior puts 100% of its probability on this single value:

$$p(\boldsymbol{\theta} | \mathcal{D}) \approx \delta(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) \quad (7.12)$$

²¹ The advantage of this approach is that we can compute the MAP estimate using a variety of
²² optimization algorithms, which we discuss in Chapter 6. The disadvantages of the MAP approximation
²³ are discussed in Section 3.1.5.

²⁴

²⁵ 7.4.2 Grid approximation

²⁶

²⁷ If we want to capture uncertainty, we need to allow for the fact that $\boldsymbol{\theta}$ may have a range of possible
²⁸ values, each with non-zero probability. The simplest way to capture this property is to partition
²⁹ the space of possible values into a finite set of possibilities, call them $\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_K$, and then to
³⁰ approximate the normalization constant by brute-force enumeration. This gives us the following
³¹ **grid approximation**:

$$p(\boldsymbol{\theta} = \boldsymbol{\theta}_k | \mathcal{D}) \approx \frac{p(\mathcal{D} | \boldsymbol{\theta}_k)p(\boldsymbol{\theta}_k)}{p(\mathcal{D})} = \frac{p(\mathcal{D} | \boldsymbol{\theta}_k)p(\boldsymbol{\theta}_k)}{\sum_{k'=1}^K p(\mathcal{D}, \boldsymbol{\theta}_{k'})} \quad (7.13)$$

³² As a simple example, we will use the problem of approximating the posterior of a beta-Bernoulli
³³ model. Specifically, the goal is to approximate

$$p(\boldsymbol{\theta} | \mathcal{D}) \propto \left[\prod_{n=1}^N \text{Bin}(y_n | \boldsymbol{\theta}) \right] \text{Beta}(1, 1) \quad (7.14)$$

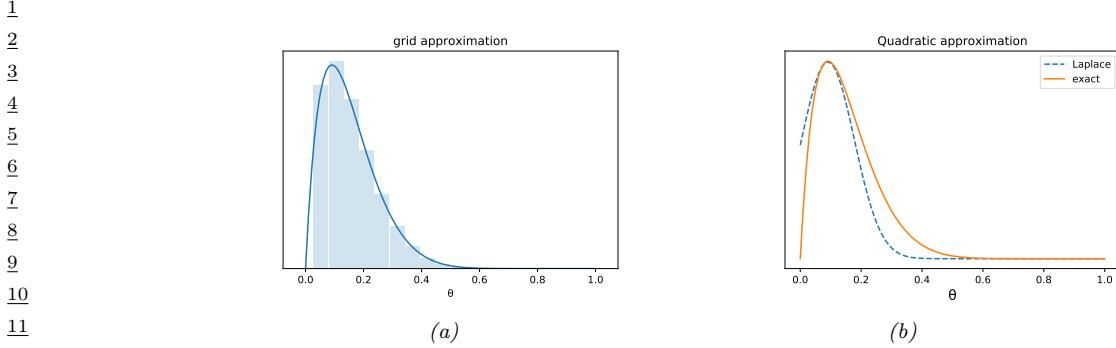


Figure 7.2: Approximating the posterior of a beta-Bernoulli model. (a) Grid approximation using 20 grid points. (b) Laplace approximation. Generated by `beta_binom_approx_post_pymc3.py`.

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

$$\hat{p}(\boldsymbol{\theta}, \mathcal{D}) = e^{-\mathcal{E}(\hat{\boldsymbol{\theta}})} \exp \left[-\frac{1}{2} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T \mathbf{H} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) \right] \quad (7.17)$$

$$\hat{p}(\boldsymbol{\theta}|\mathcal{D}) = \frac{1}{Z} \hat{p}(\boldsymbol{\theta}, \mathcal{D}) = \mathcal{N}(\boldsymbol{\theta}|\hat{\boldsymbol{\theta}}, \mathbf{H}^{-1}) \quad (7.18)$$

$$Z = e^{-\mathcal{E}(\hat{\boldsymbol{\theta}})} (2\pi)^{D/2} |\mathbf{H}|^{-\frac{1}{2}} \quad (7.19)$$

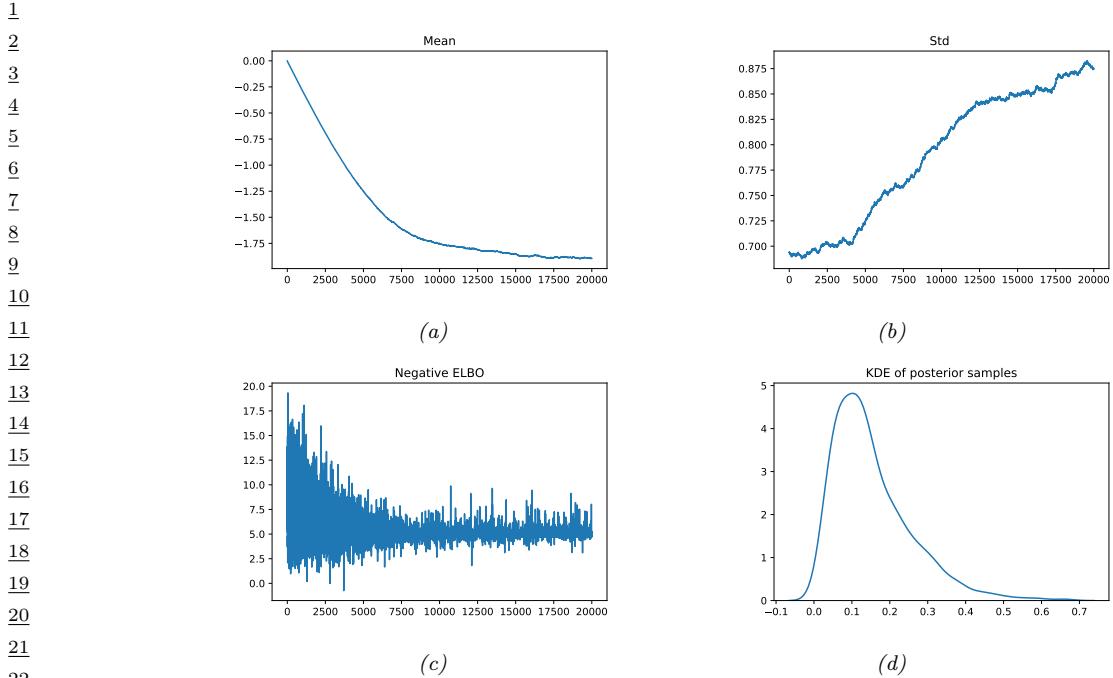


Figure 7.3: Approximating the posterior of a beta-Bernoulli model using Gaussian approximation to the logits, $q(\alpha) = \mathcal{N}(\mu, \sigma)$, where $\alpha = \text{logit}(\theta)$. (a) Estimate of variational mean over time. (b) Estimate of variational standard deviation over time. (c) ELBO over 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 $p(\boldsymbol{\theta}|\mathcal{D})$, with one that is tractable, $q(\boldsymbol{\theta})$, so as to minimize some discrepancy D between the distributions:

$$\underset{q \in \mathcal{Q}}{\operatorname{argmin}} D(q, p) \quad (7.20)$$

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 $\boldsymbol{\psi}$.

It is common to use the KL divergence (Section 5.1) as the discrepancy measure, so $D(q, p) = D_{\text{KL}}(q(\boldsymbol{\theta})\|p(\boldsymbol{\theta}))$. In this case, we need to compute

$$\boldsymbol{\psi}^* = \underset{\boldsymbol{\psi}}{\operatorname{argmin}} D_{\text{KL}}(q(\boldsymbol{\theta}|\boldsymbol{\psi})\|p(\boldsymbol{\theta}|\mathcal{D})) \quad (7.21)$$

$$= \underset{\boldsymbol{\psi}}{\operatorname{argmin}} \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\psi})} \left[\log q(\boldsymbol{\theta}|\boldsymbol{\psi}) - \log \left(\frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})} \right) \right] \quad (7.22)$$

$$= \underset{\boldsymbol{\psi}}{\operatorname{argmin}} \underbrace{\mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\psi})} [-\log p(\mathcal{D}|\boldsymbol{\theta}) - \log p(\boldsymbol{\theta}) + \log q(\boldsymbol{\theta}|\boldsymbol{\psi})]}_{-\mathcal{L}(\boldsymbol{\psi})} + \log p(\mathcal{D}) \quad (7.23)$$

Note that $\log p(\mathcal{D})$ is independent of $\boldsymbol{\psi}$, so we can ignore it when fitting the approximate posterior, and just focus on maximizing the term

$$\mathcal{L}(\boldsymbol{\psi}) \triangleq \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\psi})} [\log p(\mathcal{D}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) - \log q(\boldsymbol{\theta}|\boldsymbol{\psi})] \quad (7.24)$$

Since we have $D_{\text{KL}}(q\|p) \geq 0$, we have $\mathcal{L}(\boldsymbol{\psi}) \leq \log p(\mathcal{D})$. The quantity $\log p(\mathcal{D})$, which is the log marginal likelihood, is also called the **evidence**. Hence $\mathcal{L}(\boldsymbol{\psi})$ 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, $q(\boldsymbol{\theta}|\boldsymbol{\psi}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. This is different from the Laplace approximation, since in VI, we optimize $\boldsymbol{\Sigma}$, rather than equating it to the Hessian. If $\boldsymbol{\Sigma}$ 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 $q(\theta|\boldsymbol{\psi}) = \text{Beta}(\theta|a, b)$, where $\boldsymbol{\psi} = (a, b)$. 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 $p(\alpha|\mathcal{D}) \approx q(\alpha|\boldsymbol{\psi}) = \mathcal{N}(\alpha|\mu, \sigma)$, where $\boldsymbol{\psi} = (\mu, \sigma)$. 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

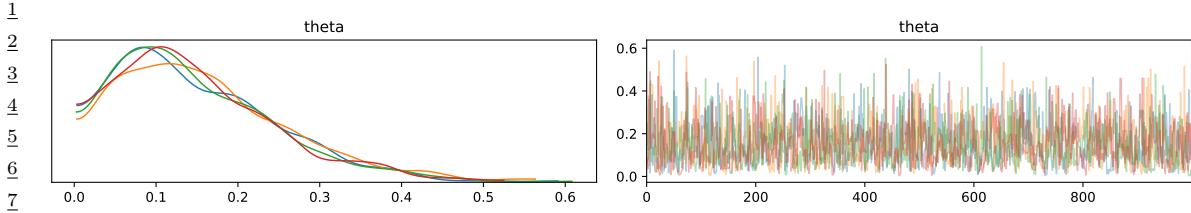


Figure 7.4: Approximating the posterior of a beta-Bernoulli model using MCMC. (a) Kernel density estimate derived from samples from 4 independent chains. (b) Trace plot of the chains as they generate posterior samples. Generated by [beta_binom_approx_post_pymc3.py](#).

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 We see that the input to the algorithm is just a function that computes the log joint density,
38 $\log p(\boldsymbol{\theta}, \mathcal{D})$, as well as a proposal distribution $q(\boldsymbol{\theta}'|\boldsymbol{\theta})$ for deciding which states to visit next. It is
39 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
40 **random walk Metropolis** algorithm. However, this can be very inefficient, since it is blindly
41 walking through the space, in the hopes of finding higher probability regions.

42 In models that have conditional independence structure, it is often easy to compute the **full**
43 **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
44

45

1 like a stochastic analog of coordinate ascent, and is called **Gibbs sampling** (see Section 12.3 for
2 details).
3

4 For models where all unknown variables are continuous, we can often compute the gradient of the
5 log joint, $\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}, \mathcal{D})$. We can use this gradient information to guide the proposals into regions of
6 space with higher probability. This approach is called **Hamiltonian Monte Carlo** or **HMC**, and
7 is one of the most widely used MCMC algorithms due to its speed. For details, see Section 12.5.

8 We apply HMC to our beta-Bernoulli model in Figure 7.4. (We use a logit transformation for
9 the parameter.) In panel b, we show samples generated by the algorithm from 4 parallel Markov
10 chains. We see that they oscillate around the true posterior, as desired. In panel a, we compute a
11 kernel density estimate from the posterior samples from each chain; we see that the result is a good
12 approximation to the true posterior in Figure 7.2.
13

14 7.4.6 Sequential Monte Carlo

15 MCMC is like a stochastic local search algorithm, in that it makes moves through the state space of
16 the posterior distribution, comparing the current value to proposed neighboring values. An alternative
17 approach is to use perform inference using a sequence of different distributions, from simpler to more
18 complex, with the final distribution being equal to the target posterior. This is called **sequential**
19 **Monte Carlo** or **SMC**. This approach, which is more similar to tree search than local search, has
20 various advantages over MCMC, which we discuss in Chapter 13.
21

22 A common application of SMC is to **sequential Bayesian inference**, in which we recursively
23 compute (i.e., in an online fashion) the posterior $p(\boldsymbol{\theta}_t | \mathcal{D}_{1:t})$, where $\mathcal{D}_{1:t} = \{(\mathbf{x}_n, y_n) : n = 1 : t\}$ is all
24 the data we have seen so far. This sequence of distributions converges to the full batch posterior
25 $p(\boldsymbol{\theta} | \mathcal{D})$ once all the data has been seen. However, the approach can also be used when the data is
26 arriving in a continual, unending stream, as in state-space models (see Chapter 31). The application
27 of SMC to such dynamical models is known as **particle filtering**. See Section 13.2 for details.
28

29 7.5 Evaluating approximate inference algorithms

30 There are many different inference algorithms each of which make different tradeoffs between speed,
31 accuracy, generality, simplicity, etc. This makes it hard to compare them on an equal footing.
32 However, a common approach is to evaluate the accuracy of the approximation as a function of
33 compute time.
34

35 We give an example of this in Figure 7.5, where we plot performance for several different inference
36 algorithms applied to the following simple 1d model:
37

$$\text{p}(z) = \mathcal{N}(z | 0, 100) \tag{7.26}$$

$$\text{p}(x_i | z) = 0.5\mathcal{N}(x_i | z, 1) + 0.5\mathcal{N}(x_i | 0, 10) \tag{7.27}$$

41 That is, each measurement x_i is either a noisy copy of the hidden quantity of interest, z , or is an
42 outlier coming from a uniform background noise model, approximated by a Gaussian with a large
43 variance, $\mathcal{N}(0, 10)$. The goal is to compute $p(z | \mathbf{x}_{1:N})$. (Tom Minka (who created this example) calls
44 this the **clutter problem**.)

45 Since this is a 1d problem, we can compute the exact answer using numerical integration. In
46 Figure 7.5, we plot the accuracy of the posterior mean estimate using various approximate inference
47

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17

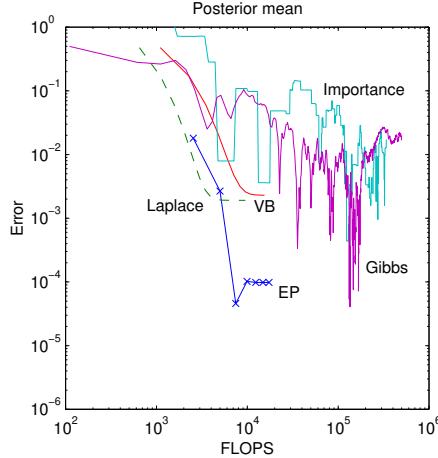


Figure 7.5: Accuracy vs compute time for different inference methods. Code source: <https://github.com/tminka/ep-clutter-example>. From Figure 1 of [Min01b]. Used with kind permission of Tom Minka.

18
19
20

21 methods, namely: the Laplace approximation (Section 7.4.3), variational Bayes (Section 10.2.3),
22 expectation propagation (Section 10.7), importance sampling (Section 11.5), and Gibbs sampling
23 (Section 12.3). We see that the error smoothly decreases for the 3 deterministic methods (Laplace,
24 BP and EP). For the 2 stochastic methods (IS and Gibbs), the error decreases on average, but the
25 performance is quite noisy.
26

In principle, the Monte Carlo methods will converge to a zero overall error, since they are unbiased
27 estimators, but it is clear that their variance is quite high. The deterministic methods, by comparison,
28 converge to a finite error, so they are biased, but their variance is much lower. We can trade off bias
29 and variance by creating hybrid algorithms, that combine different techniques. We will see examples
30 of this in later chapters.
31

When we cannot compute the true target posterior distribution to compare to, evaluation is
32 harder, but there are some approaches than can be used in certain cases. For some methods for
33 assessing variational inference, see e.g., [Yao+18b; Hug+20]. For some methods for assessing Monte
34 Carlo methods, see [CGR06; CTM17; GAR16]. For assessing posterior predictive distributions, see
35 Section 14.2.3.
36

37
38
39
40
41
42
43
44
45
46
47

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.

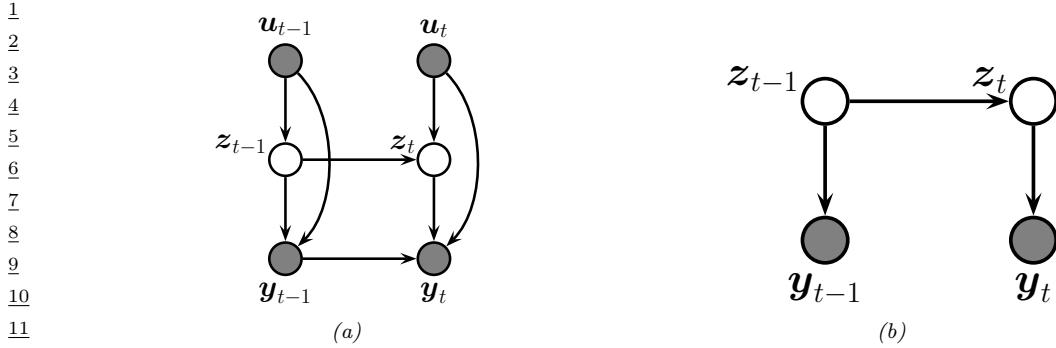


Figure 8.1: State-space model represented as a graphical model. (a) Generic form, with inputs \mathbf{u}_t , hidden state \mathbf{z}_t , and observations \mathbf{y}_t . We assume the observation likelihood is first-order auto-regressive. (b) Simplified form, with no inputs, and Markovian observations.

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

$$p(\mathbf{z}_{1:T}, \mathbf{y}_{1:T}) = \prod_{t=1}^T p(\mathbf{z}_t | \mathbf{z}_{t-1}) p(\mathbf{y}_t | \mathbf{z}_t, \theta) \quad (8.6)$$

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 $\mathbf{z}_t \in \{1, \dots, K\}$, 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.

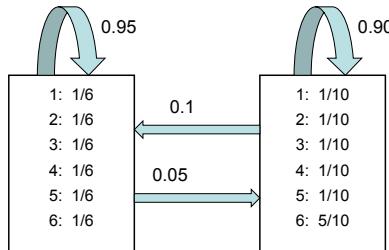


Figure 8.2: An HMM for the occasionally dishonest casino. The blue arrows represent the state transition diagram **A**. The list of numbers in each rectangle are the parameters of the observation matrix **B**. Adapted from [Dur+98, p54].

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 $\mathbf{y}_{1:T}$ 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 \mathbf{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:

$y: 664153216162115234653214356634261655234232315142464156663246$

$z: 222222222222211111122222222222111111111111122222222$

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 $p(z|\mathbf{y})$. In a temporal model, there are several kinds of posteriors we may want to compute, as we discuss in Section 8.1.4.

¹ ² **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

$$p(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{u}_t) = \mathcal{N}(\mathbf{z}_t | \mathbf{A}_t \mathbf{z}_{t-1} + \mathbf{B}_t \mathbf{u}_t, \mathbf{Q}_t) \quad (8.9)$$

$$p(\mathbf{y}_t | \mathbf{z}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{y}_t | \mathbf{C}_t \mathbf{z}_t + \mathbf{D}_t \mathbf{u}_t, \mathbf{R}_t) \quad (8.10)$$

⁷ ⁸ ⁹ 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
¹⁰ ¹¹ model is stationary. See Figure 8.1 for the PGM-D.²

¹² A common application of LG-SSMs is for **tracking** objects, such as airplanes or animals, from
¹³ noisy measurements, such as radar or cameras. We discuss a specific 2d example in Section 31.2.2.
¹⁴ Here we give a brief summary of this model, so we can use this as a running example. The hidden
¹⁵ state \mathbf{z}_t encodes the location, (x, y) , and the velocity, (\dot{x}, \dot{y}) , of the moving object. The observation
¹⁶ \mathbf{y}_t is a noisy version of the location. (The velocity is not observed but can be inferred from the
¹⁷ change in location.) We assume that we obtain measurements with a sampling frequency of Δ . The
¹⁸ new location is the old location plus Δ times the velocity, plus noise added to all terms:
¹⁹

$$\mathbf{z}_t = \underbrace{\begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{A}} \mathbf{z}_{t-1} + \mathbf{q}_t \quad (8.11)$$

²⁰ ²¹ where $\mathbf{q}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_t)$. The observation extracts the location and adds noise: In matrix notation, the
²² ²³ observation model becomes

$$\mathbf{y}_t = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}}_{\mathbf{C}} \mathbf{z}_t + \mathbf{r}_t \quad (8.12)$$

²⁴ ²⁵ where $\mathbf{r}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_t)$.

²⁶ Our goal is to use this model to estimate the unknown location (and velocity) of the object given
²⁷ the noisy observations. We discuss this in more detail in Section 8.1.4.

²⁸ ²⁹ **8.1.4 Inferential goals**

³⁰ When faced with an SSM, there are various forms of inference we may be interested in performing. We
³¹ give a visual summary in Figure 8.3, and give more details below. (See also [Sar13] for detailed coverage
³² of inference in SSMs, as well as <https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python>,
³³ which has lots of excellent tutorial material.)

³⁴ ³⁵ 2. Some authors write \mathbf{F}_t instead of \mathbf{A}_t and \mathbf{H}_t instead of \mathbf{C}_t .

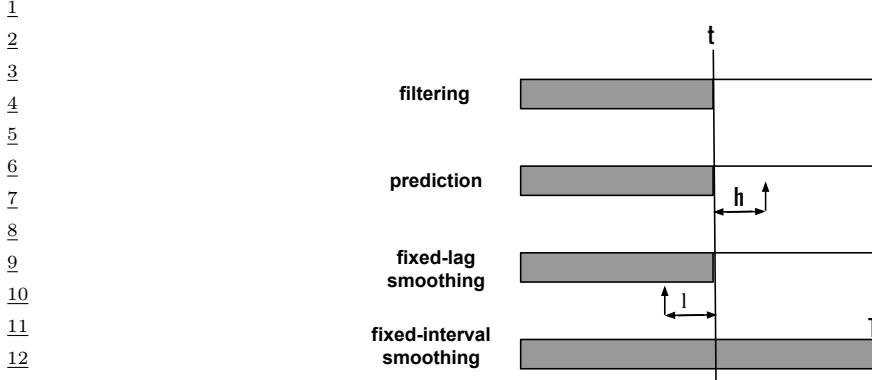


Figure 8.3: The main kinds of inference for state-space models. The shaded region is the interval for which we have data. The arrow represents the time step at which we want to perform inference. t is the current time, T is the sequence length, ℓ is the lag and h is the prediction horizon. See text for details.

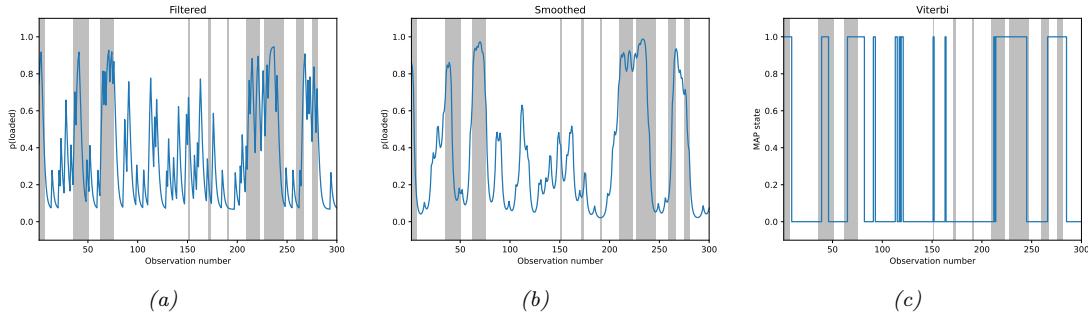


Figure 8.4: Inference in the dishonest casino. Vertical gray bars denote times when the hidden state corresponded to the loaded die. Blue lines represent the posterior probability of being in that state given different subsets of observed data. If we recover the true state exactly, the blue curve will 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

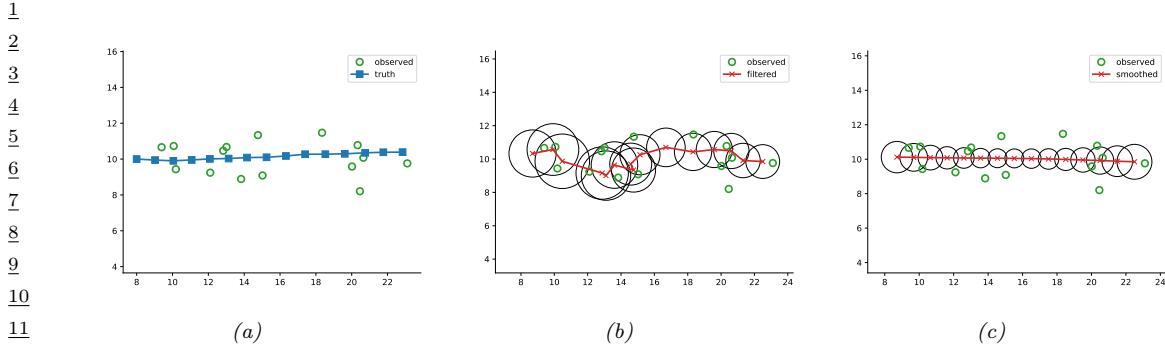


Figure 8.5: Illustration of Kalman filtering and smoothing for a linear dynamical system. (a) Observations (green circles) are generated by an object moving to the right (true location denoted by blue squares). (b) Results of online Kalman filtering. Red cross is the posterior mean, circles are 95% confidence 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.

1 **8.1.4.3 MAP state estimation**

2 The **MAP estimate** is the most probable sequence of states, i.e.,

3

$$\underline{z}_{1:T}^* = \underset{\underline{z}_{1:T}}{\operatorname{argmax}} p(\underline{z}_{1:T} | \underline{y}_{1:T}) \quad (8.13)$$

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

20

$$p(\underline{z}_{t+2} | \underline{y}_{1:t}) = \int p(\underline{z}_{t+2} | \underline{z}_{t+1}) p(\underline{z}_{t+1} | \underline{y}_{1:t}) d\underline{z}_{t+1} \quad (8.15)$$

21

22 \vdots

23

$$p(\underline{z}_{t+h} | \underline{y}_{1:t}) = \int p(\underline{z}_{t+h} | \underline{z}_{t+h-1}) p(\underline{z}_{t+h-1} | \underline{y}_{1:t}) d\underline{z}_{t+h-1} \quad (8.16)$$

24

25 The quantity $p(\underline{z}_{t+h} | \underline{y}_{1:t})$ is a prediction about future hidden states. We can convert this into a
26 prediction about future observations using

27

$$p(\underline{y}_{t+h} | \underline{y}_{1:t}) = \int p(\underline{y}_{t+h} | \underline{z}_{t+h}) p(\underline{z}_{t+h} | \underline{y}_{1:t}) d\underline{z}_{t+h} \quad (8.17)$$

28

29 This is the h -step-ahead predictive distribution for observations, and can be used for time-series
30 forecasting (see Section 19.3).

31 **8.2 Bayesian filtering and smoothing**

32 In this section, we derive the optimal form for the filtered and smoothed posteriors for the simplified
33 SSM in Equation (8.6). (The equations can easily be extended to the more general model in
34 Equation (8.5).) This will require integrating (or summing) over the latent states. In the following
35 sections, we discuss how to compute these integrals in practice, depending on the form of the model,
36 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**:

$$\underline{10} \quad p(\mathbf{z}_t | \mathbf{y}_{1:t-1}) = \int p(\mathbf{z}_t | \mathbf{z}_{t-1}) p(\mathbf{z}_{t-1} | \mathbf{y}_{1:t-1}) d\mathbf{z}_{t-1} \quad (8.18)$$

11 The prediction step computes the one-step-ahead predictive distribution for the latent state, which
12 updates the posterior from the previous time step into the prior for the current step.

14 **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

$$\underline{39} \quad p(\mathbf{z}_t | \mathbf{z}_{t+1}, \mathbf{y}_{1:T}) = p(\mathbf{z}_t | \mathbf{z}_{t+1}, \mathbf{y}_{1:t}) \quad (8.23)$$

$$\underline{40} \quad = \frac{p(\mathbf{z}_t, \mathbf{z}_{t+1} | \mathbf{y}_{1:t})}{p(\mathbf{z}_{t+1} | \mathbf{y}_{1:t})} \quad (8.24)$$

$$\underline{42} \quad = \frac{p(\mathbf{z}_{t+1} | \mathbf{z}_t, \mathbf{y}_{1:t}) p(\mathbf{z}_t | \mathbf{y}_{1:t})}{p(\mathbf{z}_{t+1} | \mathbf{y}_{1:t})} \quad (8.25)$$

$$\underline{45} \quad = \frac{p(\mathbf{z}_{t+1} | \mathbf{z}_t) p(\mathbf{z}_t | \mathbf{y}_{1:t})}{p(\mathbf{z}_{t+1} | \mathbf{y}_{1:t})} \quad (8.26)$$

1 Hence the joint distribution over two consecutive time steps is given by
2

$$p(\mathbf{z}_t, \mathbf{z}_{t+1} | \mathbf{y}_{1:T}) = p(\mathbf{z}_t | \mathbf{z}_{t+1}, \mathbf{y}_{1:T}) p(\mathbf{z}_{t+1} | \mathbf{y}_{1:T}) \quad (8.27)$$

$$= p(\mathbf{z}_t | \mathbf{z}_{t+1}, \mathbf{y}_{1:t}) p(\mathbf{z}_{t+1} | \mathbf{y}_{1:T}) \quad (8.28)$$

$$= \frac{p(\mathbf{z}_{t+1} | \mathbf{z}_t) p(\mathbf{z}_t | \mathbf{y}_{1:t}) p(\mathbf{z}_{t+1} | \mathbf{y}_{1:T})}{p(\mathbf{z}_{t+1} | \mathbf{y}_{1:t})} \quad (8.29)$$

9 Marginalizing out \mathbf{z}_{t+1} gives Equation (8.21).
10

11 8.3 Inference for discrete SSMs

13 In this section, we consider inference for chain-structured graphical models, where all the hidden
14 variables are discrete. This includes HMMs (Section 30.1), linear-chain CRFs (Section 19.2), etc. We
15 represent the hidden variables at time t by $z_t \in \{1, \dots, K\}$, and the visible variables by $\mathbf{y}_t \in \mathbb{R}^D$.

16 As we discussed in Section 8.1.4, there are several kinds of inference we may be interested in when
17 working with temporal or sequential models, namely filtering (i.e., computing $p(z_t | \mathbf{y}_{1:t})$), smoothing
18 (i.e., computing $p(z_t | \mathbf{y}_{1:T})$), and MAP estimation (i.e., computing $\text{argmax}_{\mathbf{z}_{1:T}} p(\mathbf{z}_{1:T} | \mathbf{y}_{1:T})$). In this
19 section, we discuss how to solve all three of these problems in $O(TK^2)$ time, where T is the length of
20 the chain. The basic idea is to exploit the Markov structure of the model so that we can recursively
21 solve smaller inference problems for the left and right half of the model, and then combine the
22 solutions to these subproblems in an optimal way. We give the details below.

24 8.3.1 Forwards filtering

26 In this section, we discuss the **forwards algorithm** for HMMs, which is a way to recursively compute
27 the **belief state** $\alpha_t = p(z_t | \mathbf{y}_{1:t})$, where $\alpha_t(j) = p(z_t = j | \mathbf{y}_{1:t})$ is the probability the hidden state has
28 value j given the evidence seen so far. We can do this using **sequential Bayesian updating** as
29 follows:

$$p(z_t = j | \mathbf{y}_{1:t}) = p(z_t = j | \mathbf{y}_t, \mathbf{y}_{1:t-1}) \propto p(\mathbf{y}_t | z_t = j, \mathbf{y}_{1:t-1}) p(z_t = j | \mathbf{y}_{1:t-1}) \quad (8.30)$$

$$= p(\mathbf{y}_t | z_t = j) \left[\sum_i p(z_t = j | z_{t-1} = i) p(z_{t-1} = i | \mathbf{y}_{1:t-1}) \right] \quad (8.31)$$

35 where we have crossed out $\mathbf{y}_{1:t-1}$ since \mathbf{y}_t is conditionally independent of past observations given z_t .
36 (This assumption is not actually necessary for the algorithm to work: the only requirement is that
37 the *hidden* states be first-order Markov.)

38 The term in the brackets is known as the **one-step ahead prediction distribution**, and is given
39 by

$$\alpha_{t|t-1}(j) \triangleq p(z_t = j | \mathbf{y}_{1:t-1}) = \sum_i \alpha_{t-1}(i) A(i, j) \quad (8.32)$$

43 where $A(i, j) = p(z_t = j | z_{t-1} = i)$ is the state transition probability. We then multiply this by the
44 **local evidence**

$$\lambda_t(j) \triangleq p(\mathbf{y}_t | z_t = j) \quad (8.33)$$

which is the result of evaluating the observation model using \mathbf{y}_t for each possible state. Combining these gives

$$\alpha_t(j) = \frac{1}{Z_t} \lambda_t(j) \left[\sum_i \alpha_{t-1}(i) A(i, j) \right] \quad (8.34)$$

where the normalization constant for each time step is given by

$$Z_t \triangleq p(\mathbf{y}_t | \mathbf{y}_{1:t-1}) = \sum_{j=1}^K p(\mathbf{y}_t | z_t = j) p(z_t = j | \mathbf{y}_{1:t-1}) = \sum_{j=1}^K \lambda_t(j) \left[\sum_i \alpha_{t-1}(i) A(i, j) \right] \quad (8.35)$$

We can therefore view Equation (8.31) in terms of a **predict-update cycle**: in the first step, we predict the distribution over states given the past (i.e., we compute $\boldsymbol{\alpha}_{t|t-1}$), and then we update our predictions using the likelihood $p(\mathbf{y}_t | z_t) = \boldsymbol{\lambda}_t$ to compute the new posterior $\boldsymbol{\alpha}_t$.

Since all the quantities are finite length vectors and matrices, we can write the update equation in matrix-vector notation as follows:

$$\boldsymbol{\alpha}_t = \text{normalize}(\boldsymbol{\lambda}_t \odot (\mathbf{A}^\top \boldsymbol{\alpha}_{t-1})) \quad (8.36)$$

where \odot represents elementwise vector multiplication, and the normalize function just ensures its argument sums to one.

It is useful to keep track of the normalization constants, since they can be used to compute the log likelihood of the sequence as follows:

$$\log p(\mathbf{y}_{1:T}) = \sum_{t=1}^T \log p(\mathbf{y}_t | \mathbf{y}_{1:t-1}) = \sum_{t=1}^T \log Z_t \quad (8.37)$$

8.3.1.1 An aside on normalization

In most publications on HMMs, $\alpha_t(j)$ is defined as the unnormalized *joint probability* $p(z_t = j, \mathbf{y}_{1:t})$, perhaps due to the influential tutorial [Rab89]. That is, the standard definition is to use

$$\alpha'_t(j) = p(z_t = j, \mathbf{y}_{1:t}) = \lambda_t(j) \left[\sum_i \alpha'_{t-1}(i) A(i, j) \right] \quad (8.38)$$

without the Z_t term. We instead define $\alpha_t(j)$ as the normalized *conditional probability* $p(z_t = j | \mathbf{y}_{1:t})$.

The unnormalized form has several problems. First, it rapidly suffers from numerical underflow, since the probability of the joint event that $(z_t = k, \mathbf{y}_{1:t})$ is vanishingly small.³ Second, it is less interpretable, since it is not a distribution over states. Third, it precludes the use of methods which are designed to approximate state distributions (we will see such methods later).

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 or absence of a call to the `normalize` function). Nevertheless, we feel it is better to present the normalized version, since it will encourage readers to implement the method properly (normalizing after each step to avoid underflow), and to think about it in terms of posterior filtering.

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 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:

$$\underline{3} \quad \beta_{t-1}(i) = p(\mathbf{y}_{t:T}|z_{t-1} = i) \quad (8.46)$$

$$\underline{5} \quad = \sum_j p(z_t = j, \mathbf{y}_t, \mathbf{y}_{t+1:T}|z_{t-1} = i) \quad (8.47)$$

$$\underline{7} \quad = \sum_j p(\mathbf{y}_{t+1:T}|z_t = j, \mathbf{y}_t, \cancel{z_{t-1} = i}) p(z_t = j, \mathbf{y}_t|z_{t-1} = i) \quad (8.48)$$

$$\underline{9} \quad = \sum_j p(\mathbf{y}_{t+1:T}|z_t = j) p(\mathbf{y}_t|z_t = j, \cancel{z_{t-1} = i}) p(z_t = j|z_{t-1} = i) \quad (8.49)$$

$$\underline{11} \quad = \sum_j \beta_t(j) \lambda_t(j) A(i, j) \quad (8.50)$$

14

We can write the resulting equation in matrix-vector form as

$$\underline{16} \quad \underline{\beta}_{t-1} = \mathbf{A}(\boldsymbol{\lambda}_t \odot \beta_t) \quad (8.51)$$

18 The base case is
19

$$\underline{20} \quad \beta_T(i) = p(\mathbf{y}_{T+1:T}|z_T = i) = p(\emptyset|z_T = i) = 1 \quad (8.52)$$

22 which is the probability of a non-event.

23

24 8.3.3.1 Another aside on normalization

25 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

$$\underline{30} \quad \underline{31} \quad \beta_{t-1}(i) = \frac{1}{Z'_t} \sum_j \beta_t(j) \lambda_t(j) A(i, j) \quad (8.53)$$

33 where

$$\underline{35} \quad Z'_t = \sum_i \sum_j \beta_t(j) \lambda_t(j) A(i, j) \quad (8.54)$$

38 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.

41

42 8.3.3.2 Two-filter smoothing

43 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**

47

1 smoothing [Kit04], although in the HMM literature, it is called the forwards-backwards algorithm
2 [Rab89]. The disadvantage of this approach compared to backwards smoothing is that the back-
3 wards messages are not probability distributions; this makes it harder to employ approximationg
4 techniques for distributions, which is often necessary when working with continuous latent variables
5 (see Section 8.4.4).

8 8.3.4 Two-slice smoothed marginals

10 When we estimate the parameters of the transition matrix (e.g., using EM, as described in Section
11 30.4.1), we need to compute the expected number of transitions from state i to state j :

$$\underline{12} \quad N_{ij} = \sum_{t=1}^{T-1} \mathbb{E}[\mathbb{I}(z_t = i, z_{t+1} = j) | \mathbf{y}_{1:T}] = \sum_{t=1}^{T-1} p(z_t = i, z_{t+1} = j | \mathbf{y}_{1:T}) \quad (8.55)$$

16 The term $p(z_t, z_{t+1} | \mathbf{y}_{1:T})$ is called a (smoothed) **two-slice marginal**. We can compute this from
17 Equation (8.29) as follows:

$$\underline{19} \quad p(z_t = i, z_{t+1} = j | \mathbf{y}_{1:T}) = p(z_t = i | \mathbf{y}_{1:t}) \sum_j \left[\frac{p(z_{t+1} = j | z_t = i) p(z_{t+1} | \mathbf{y}_{1:T})}{p(z_{t+1} = j | \mathbf{y}_{1:t})} \right] \quad (8.56)$$

22 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:

$$\underline{25} \quad \xi_{t,t+1}(i, j) = \alpha_t(i) \sum_j A(i, j) \frac{\gamma_{t+1}(j)}{\alpha_{t+1|t}(j)} \quad (8.57)$$

27 where

$$\underline{30} \quad \alpha_{t+1|t}(j) = p(z_{t+1} = j | \mathbf{y}_{1:t}) = \sum_i p(z_{t+1} = j | z_t = i) p(z_t = i | \mathbf{y}_{1:t}) \quad (8.58)$$

32 is the one-step-ahead predictive distribution. We can interpret the ratio in Equation (8.57) as dividing
33 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
34 $\mathbf{y}_{1:T}$, namely γ_{t+1} . See [SHJ97] for further insight into these equations.

35 The more traditional approach to deriving the two-slice marginals uses the output of the forwards
36 backwards algorithm as follows:

$$\underline{38} \quad p(z_t, z_{t+1} | \mathbf{y}_{1:T}) = p(z_t, z_{t+1} | \mathbf{y}_{1:t}, \mathbf{y}_{t+1:T}) \quad (8.59)$$

$$\underline{39} \quad \propto p(\mathbf{y}_{t+1:T} | z_t, z_{t+1}, \mathbf{y}_{1:t}) p(z_t, z_{t+1} | \mathbf{y}_{1:t}) \quad (8.60)$$

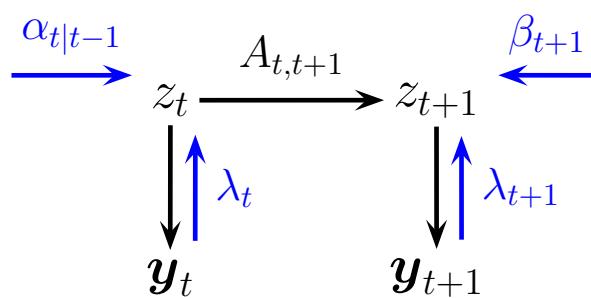
$$\underline{40} \quad = p(\mathbf{y}_{t+1:T} | z_{t+1}) p(z_t, z_{t+1} | \mathbf{y}_{1:t}) \quad (8.61)$$

$$\underline{41} \quad = p(\mathbf{y}_{t+1:T} | z_{t+1}) p(z_t | \mathbf{y}_{1:t}) p(z_{t+1} | z_t) \quad (8.62)$$

$$\underline{42} \quad = p(\mathbf{y}_{t+1}, \mathbf{y}_{t+2:T} | z_{t+1}) p(z_t | \mathbf{y}_{1:t}) p(z_{t+1} | z_t) \quad (8.63)$$

$$\underline{43} \quad = p(\mathbf{y}_{t+1} | z_{t+1}) p(\mathbf{y}_{t+2:T} | z_{t+1}, \mathbf{y}_{t+1}) p(z_t | \mathbf{y}_{1:t}) p(z_{t+1} | z_t) \quad (8.64)$$

$$\underline{44} \quad = p(\mathbf{y}_{t+1} | z_{t+1}) p(\mathbf{y}_{t+2:T} | z_{t+1}) p(z_t | \mathbf{y}_{1:t}) p(z_{t+1} | z_t) \quad (8.65)$$



11 *Figure 8.6: Computing the two-slice joint distribution for an HMM from the forwards messages, backwards
12 messages, and local evidence messages.*

13

14

15 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:

16

$$\xi_{t,t+1}(i, j) \propto \lambda_t(j) \beta_{t+1}(j) \alpha_t(i) A(i, j) \quad (8.66)$$

18

19 Or in matrix-vector form:

20

$$\xi_{t,t+1} \propto \mathbf{A} \odot [\alpha_t(\lambda_{t+1} \odot \beta_{t+1})^\top] \quad (8.67)$$

22

23 Since $\alpha_t \propto \lambda_t \odot \alpha_{t|t-1}$, we can also write the above equation as follows:

24

$$\xi_{t,t+1} \propto (\lambda_t \odot \alpha_{t|t-1}) \odot \mathbf{A} \odot (\lambda_{t+1} \odot \beta_{t+1})^\top \quad (8.68)$$

25

26 This can be interpreted as a product of incoming “messages” and local factors, as shown in
27 Figure 8.6. (This interpretation will be explained in Section 9.2.) In particular, we combine the factors
28 $\alpha_{t|t-1} = p(z_t | \mathbf{y}_{1:t-1})$, $\mathbf{A} = p(z_t | z_{t-1})$, $\lambda_t \propto p(\mathbf{y}_t | z_t)$, $\lambda_{t+1} \propto p(\mathbf{y}_{t+1} | z_{t+1})$, and $\beta_{t+1} \propto p(\mathbf{y}_{t+2:T} | z_{t+1})$
29 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

31 8.3.5 Time and space complexity

32

33 It is clear that a straightforward implementation of the forwards-backwards algorithm takes $O(K^2 T)$
34 time, since we must perform a $K \times K$ matrix multiplication at each step. For some applications,
35 such as speech recognition, K is very large, so the $O(K^2)$ term becomes prohibitive. Fortunately,
36 if the transition matrix is sparse, we can reduce this substantially. For example, in a left-to-right
37 transition matrix, the algorithm takes $O(TK)$ time.

38 In some cases, we can exploit special properties of the state space, even if the transition matrix is
39 not sparse. In particular, suppose the states represent a discretization of an underlying continuous
40 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
41 continuous vector represented by state i . Then one can implement the forwards-backwards algorithm
42 in $O(TK \log K)$ time. Similar ideas can be used to speed up the Viterbi algorithm, to $O(TK)$ time.
43 where the max-product operation can be done in $O(TK)$ time. This is very useful for models with
44 large state spaces. See [FHK03] for details.

45 We can also reduce inference to $O(\log T)$ time by using a **parallel prefix scan** operator, that can
46 be run efficiently on GPUs. For details, see [HSGF21].

47

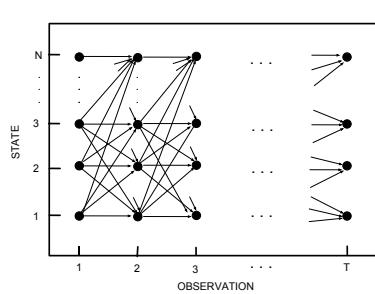


Figure 8.7: The trellis of states vs time for a Markov chain. Adapted from [Rab89].

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

$$\delta_t(j) = \max_i \delta_{t-1}(i) A(i, j) \lambda_t(j) \quad (8.74)$$

⁵ We initialize by setting $\delta_1(j) = \pi_j \lambda_1(j)$.
⁶

⁷ We also need keep track of the most likely previous (ancestor) state, for each possible state that
⁸ we end up in:
⁹

$$a_t(j) = \operatorname{argmax}_i \delta_{t-1}(i) A(i, j) \lambda_t(j) \quad (8.75)$$

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

¹³ 8.3.6.2 Backwards pass

¹⁴ In the backwards pass, we compute the most probable sequence of states using a **traceback** procedure,
¹⁵ 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
¹⁶ chain of ancestors along the MAP path.

¹⁷ If there is a unique MAP estimate, the above procedure will give the same result as picking
¹⁸ $\hat{z}_t = \operatorname{argmax}_j \gamma_t(j)$, computed by forwards-backwards, as shown in [WF01b]. However, if there are
¹⁹ multiple posterior modes, the latter approach may not find any of them, since it chooses each state
²⁰ independently, and hence may break ties in a manner that is inconsistent with its neighbors. The
²¹ traceback procedure avoids this problem, since once z_t picks its most probable state, the previous
²² nodes condition on this event, and therefore they will break ties consistently.

²³ We are free to normalize the δ_t terms at each step to avoid numerical underflow; this will not
²⁴ affect the maximum. However, since $\log \max = \max \log$, we can also do all computation in the log
²⁵ domain, which is often faster. Hence we can use
²⁶

$$\log \delta_t(j) \triangleq \max_{\mathbf{z}_{1:t-1}} \log p(\mathbf{z}_{1:t-1}, z_t = j | \mathbf{y}_{1:t}) \quad (8.76)$$

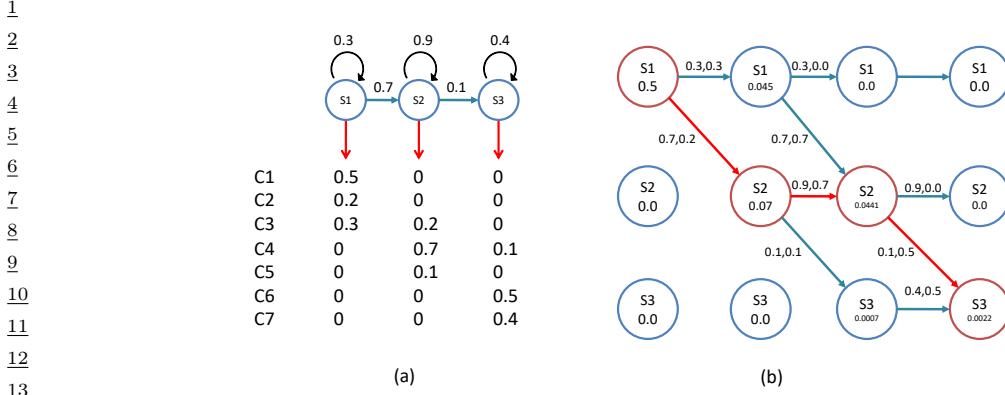
$$= \max_i \log \delta_{t-1}(i) + \log A(i, j) + \log \lambda_t(j) \quad (8.77)$$

³⁴ 8.3.6.3 Example

³⁵ Figure 8.8 gives a worked example of the Viterbi algorithm, based on [Rus+95]. Suppose we observe
³⁶ the sequence of discrete observations $\mathbf{y}_{1:4} = (C_1, C_3, C_4, C_6)$, representing codebook entries in a
³⁷ vector-quantized version of a speech signal. The model starts in state $z_1 = S_1$. The probability of
³⁸ 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
³⁹ can self-transition to S_1 with probability 0.3, or transition to S_2 with probability 0.7. If we end up
⁴⁰ 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
⁴¹ $x_2 = C_3$ is 0.2. Hence we have
⁴²

$$\delta_2(1) = \delta_1(1)A(1, 1)\lambda_2(1) = 0.5 \cdot 0.3 \cdot 0.3 = 0.045 \quad (8.78)$$

$$\delta_2(2) = \delta_1(1)A(1, 2)\lambda_2(2) = 0.5 \cdot 0.7 \cdot 0.2 = 0.07 \quad (8.79)$$



14
15 *Figure 8.8: Illustration of Viterbi decoding in a simple HMM for speech recognition. (a) A 3-state HMM
16 for a single phone. We are visualizing the state transition diagram. We assume the observations have been
17 vector quantized into 7 possible symbols, C_1, \dots, C_7 . Each state s_1, s_2, s_3 has a different distribution over
18 these symbols. Adapted from Figure 15.20 of [RN02]. (b) Illustration of the Viterbi algorithm applied to this
19 model, with data sequence C_1, C_3, C_4, C_6 . The columns represent time, and the rows represent states. The
20 numbers inside the circles represent the $\delta_t(j)$ value for that state. An arrow from state i at $t - 1$ to state
21 j at t is annotated with two numbers: the first is the probability of the $i \rightarrow j$ transition, and the second is
22 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].*

23
24
25 Thus state 2 is more probable at $t = 2$; see the second column of Figure 8.8(b). The algorithm
26 continues in this way until we have reached the end of the sequence. Once we have reached the end,
27 we can follow the red arrows back to recover the MAP path (which is 1,2,2,3).

28 For more details on HMMs for automatic speech recognition (ASR) see e.g., [JM08].
29

30 8.3.6.4 Time and space complexity

32 The time complexity of Viterbi is clearly $O(K^2T)$ in general, and the space complexity is $O(KT)$, both
33 the same as forwards-backwards. If the transition matrix has the form $A(i, j) \propto \exp(-\sigma^2 \|\mathbf{z}_i - \mathbf{z}_j\|^2)$,
34 where \mathbf{z}_i is the continuous vector represented by state i , we can implement Viterbi in $O(TK)$ time,
35 instead of $O(TK \log K)$ needed by forwards-backwards. See [FHK03] for details.
36

37 8.3.6.5 N-best list

39 There are often multiple paths which have the same likelihood. The Viterbi algorithm returns one of
40 them, but can be extended to return the top N paths [SC90; NG01]. This is called the **N-best list**.
41 Computing such a list, \mathcal{L}_N , can provide a better summary of the posterior uncertainty.

42 In addition, we can perform **discriminative reranking** [CK05] of all the sequences in \mathcal{L}_N , based
43 on global features derived from $(\mathbf{y}_{1:T}, \mathbf{z}_{1:T})$. This technique is widely used in speech recognition. For
44 example, consider the sentence “recognize speech”. It is possible that the most probable interpretation
45 by the system of this acoustic signal is “wreck a nice speech”, or maybe “wreck a nice beach”. Maybe
46 the correct interpretation is much lower down on the list. However, by using a re-ranking system, we
47

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

$$\begin{aligned} \text{p}(\mathbf{z}_{1:T} | \mathbf{y}) &= p(z_T | \mathbf{y}) p(z_{T-1} | z_T, \mathbf{y}) p(z_{T-2} | z_{T-1}, \mathbf{y}) \cdots p(z_1 | z_2, \mathbf{y}) \end{aligned} \quad (8.80)$$

$$= p(z_T | \mathbf{y}) \prod_{t=T-1}^1 p(z_t | z_{t+1}, \mathbf{y}) \quad (8.81)$$

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

$$\begin{aligned} \text{p}(z_t = i | z_{t+1} = j, \mathbf{y}_{1:T}) &= p(z_t = i | z_{t+1} = j, \mathbf{y}_{1:t}, \mathbf{y}_{t+1:T}) \end{aligned} \quad (8.82)$$

$$= \frac{p(z_t = i, z_{t+1} = j | \mathbf{y}_{1:t})}{p(z_{t+1} = j | \mathbf{y}_{1:t})} \quad (8.83)$$

$$= \frac{p(z_{t+1} = j | z_t = i) p(z_t = i | \mathbf{y}_{1:t})}{p(z_{t+1} = j | \mathbf{y}_{1:t})} \quad (8.84)$$

$$= \frac{A(i, j) \alpha_t(i)}{\sum_{i'} A(i', j) \alpha_t(i')} \quad (8.85)$$

³⁴ The base case is $p(z_T = i | \mathbf{y}_{1:T}) = \alpha_T(i)$.

³⁶ 8.3.8 Application to discretized state spaces

³⁷ Exact inference in SSMs with continuous latent states is in general intractable, except for the special
³⁸ case of linear Gaussian models (which we discuss in Section 8.4.1). For low dimensional problems, we
³⁹ can discretize the latent variables into a grid, and then run the HMM filter and smoother. This can
⁴⁰ 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

1 In higher dimensions, this discretization strategy is usually intractable, so other approximations
2 are needed, as we discuss later on.

6 8.4 Inference for linear-Gaussian SSMs

8 In this section, we discuss inference in linear-Gaussian state space models, which we introduced in
9 Section 8.1.3. This corresponds to the following model:

$$p(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{u}_t) = \mathcal{N}(\mathbf{z}_t | \mathbf{A}_t \mathbf{z}_{t-1} + \mathbf{B}_t \mathbf{u}_t, \mathbf{Q}_t) \quad (8.86)$$

$$p(\mathbf{y}_t | \mathbf{z}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{y}_t | \mathbf{C}_t \mathbf{z}_t + \mathbf{D}_t \mathbf{u}_t, \mathbf{R}_t) \quad (8.87)$$

15 An LG-SSM is just a special case of a Gaussian Bayes net (Section 4.2.2.3), so the entire joint
16 distribution $p(\mathbf{y}_{1:T}, \mathbf{z}_{1:T} | \mathbf{u}_{1:T})$ is a large multivariate Gaussian with DKT dimensions. However, it
17 has special structure that makes it computationally tractable to use, as we show below. (To simplify
18 the notation, we drop the conditioning on the inputs \mathbf{u}_t , and we assume the parameters are known.)

22 8.4.1 The Kalman filter

24 The **Kalman filter** is an algorithm for exact Bayesian filtering for linear-Gaussian state space
25 models. The resulting algorithm is the Gaussian analog of the HMM filter in Section 8.3.1, except
26 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
27 can perform the prediction and update steps in closed form, as we explain below.⁴

30 8.4.1.1 Predict step

32 The one-step-ahead prediction for the hidden state, also called the **time update step**, is given by
33 the following:⁵

$$p(\mathbf{z}_t | \mathbf{y}_{1:t-1}) = \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}) \quad (8.88)$$

$$\boldsymbol{\mu}_{t|t-1} = \mathbf{A}_t \boldsymbol{\mu}_{t-1} \quad (8.89)$$

$$\boldsymbol{\Sigma}_{t|t-1} = \mathbf{A}_t \boldsymbol{\Sigma}_{t-1} \mathbf{A}_t^\top + \mathbf{Q}_t \quad (8.90)$$

42 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 $\boldsymbol{\Sigma}_t$,
43 to be consistent with the rest of the book.

44 5. Some authors write $\boldsymbol{\mu}_t^-$ and $\boldsymbol{\Sigma}_t^-$ to represent the one-step-ahead posterior predictive distribution, $p(\mathbf{z}_t | \mathbf{y}_{1:t-1})$.
45 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
46 notation will generalize nicely to the smoothing case. Note, however, that for shorthand, we write $\boldsymbol{\mu}_t$ and $\boldsymbol{\Sigma}_t$ rather
47 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:

$$p(\mathbf{z}_t | \mathbf{y}_{1:t}) = \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t) \quad (8.91)$$

$$\mathbf{e}_t = \mathbf{y}_t - \mathbf{C}_t \boldsymbol{\mu}_{t|t-1} \quad (8.92)$$

$$\mathbf{S}_t = \mathbf{C}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{C}_t^T + \mathbf{R}_t \quad (8.93)$$

$$\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1} \mathbf{C}_t^\top \mathbf{S}_t^{-1} \quad (8.94)$$

$$\mu_t = \mu_{t|t-1} + \mathbf{K}_t e_t \quad (8.95)$$

$$\Sigma_t = \Sigma_{t|t-1} - K_t C_t \Sigma_{t|t-1} \quad (8.96)$$

$$= \Sigma_{t|t-1} - \mathbf{K}_t \mathbf{S}_t \mathbf{K}_t^\top \quad (8.97)$$

¹⁰ See also Section 10.26 and Section 10.27 in *Supplementary Note 10*.

¹⁵ (Proving the equivalence of Equation (8.96) and Equation (8.97) is left as an exercise.)

16 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
23 predicted observation $\hat{y}_{t|t-1}$ and the actual observation y_t ; this is called the **residual** or **innovation**.
24 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
25 correction factor, which is K_t times the error signal e_t .

The amount of weight placed on the error signal depends on the **Kalman gain matrix**, \mathbf{K}_t . By using the matrix inversion lemma, the Kalman gain matrix can also be written as

$$K_t = \Sigma_{t|t-1}^{-1} C_t^\top (C_t \Sigma_{t|t-1} C_t^\top + R_t)^{-1} = (\Sigma_{t|t-1}^{-1} + C_t^\top R_t^{-1} C_t)^{-1} C_t^\top R_t^{-1} \quad (8.99)$$

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.

37 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

$$p(\mathbf{z}_{t-1}, \mathbf{z}_t | \mathbf{u}_{1:t-1}) = p(\mathbf{z}_t | \mathbf{z}_{t-1}) p(\mathbf{z}_{t-1} | \mathbf{u}_{1:t-1}) \quad (8.100)$$

$$\equiv \mathcal{N}(z_t | A_t z_{t-1}, Q_{t-1}) \mathcal{N}(z_{t-1} | \mu_{t-1}, \Sigma_{t-1}) \quad (8.101)$$

$$= \mathcal{N} \left(\begin{pmatrix} z_{t-1} \\ z_* \end{pmatrix} | \boldsymbol{\mu}', \boldsymbol{\Sigma}' \right) \quad (8.102)$$

1 where
 2

$$3 \quad \mu' = \begin{pmatrix} \mu_{t-1} \\ C_{t-1}\mu_{t-1} \end{pmatrix}, \Sigma' = \begin{pmatrix} \Sigma_{t-1} & \Sigma_{t-1}A_t^\top \\ A_t\Sigma_{t-1} & A_t\Sigma_{t-1}A_t^\top + Q_{t-1} \end{pmatrix} \quad (8.103)$$

4 From Equation (2.62), the marginal distribution is
 5

$$6 \quad p(z_t | y_{1:t-1}) = \mathcal{N}(z_t | \mu_{t|t-1}, \Sigma_{t|t-1}) \quad (8.104)$$

7 This gives us the predict step.
 8

9 Now for the measurement update step. From Equation (2.56), the joint distribution for state and
 10 observation is given by
 11

$$12 \quad p(z_t, y_t) = p(y_t | z_t)p(z_t | y_{1:t-1}) \quad (8.105)$$

$$13 \quad = \mathcal{N}(y_t | C_t z_t, R_t) \mathcal{N}(z_t | \mu_{t|t-1}, \Sigma_{t|t-1}) \quad (8.106)$$

$$14 \quad = \mathcal{N} \left(\begin{pmatrix} z_t \\ y_t \end{pmatrix} | \mu'', \Sigma'' \right) \quad (8.107)$$

15 where
 16

$$17 \quad \mu'' = \begin{pmatrix} \mu_{t|t-1} \\ C_t\mu_{t|t-1} \end{pmatrix}, \Sigma'' = \begin{pmatrix} \Sigma_{t|t-1} & \Sigma_{t|t-1}C_t^\top \\ C_t\Sigma_{t|t-1} & C_t\Sigma_{t|t-1}C_t^\top + R_t \end{pmatrix} \quad (8.108)$$

18 Finally, we convert this joint into a conditional using Equation (2.50) as follows:
 19

$$20 \quad p(z_t | y_t, y_{1:t-1}) = \mathcal{N}(z_t | \mu_t, \Sigma_t) \quad (8.109)$$

$$21 \quad \mu_t = \mu_{t|t-1} + \Sigma_{t|t-1}C_t^\top(C_t\Sigma_{t|t-1}C_t^\top + R_t)^{-1}[y_t - C_t\mu_{t|t-1}] \quad (8.110)$$

$$22 \quad \Sigma_t = \Sigma_{t|t-1} - \Sigma_{t|t-1}C_t^\top(C_t\Sigma_{t|t-1}C_t^\top + R_t)^{-1}C_t\Sigma_{t|t-1} \quad (8.111)$$

23 which can be shown to equal Equation (8.97).
 24

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 Σ_t for all t . The
 38 iterative equations for updating Σ_t are called the **Riccati equations**, and for time invariant systems,
 39 they converge to a fixed point, namely $\Sigma = A\Sigma A^\top + Q$. The corresponding steady state Kalman
 40 gain matrix is therefore
 41

$$42 \quad K \triangleq \Sigma C^\top (C\Sigma C^\top + R) \quad (8.112)$$

43 This is often precomputed, to save time. The corresponding update for the posterior mean becomes
 44

$$45 \quad \mu_t = (A - KCA)\mu_{t-1} + Ky_t \quad (8.113)$$

8.4.1.6 Posterior predictive

The one-step-ahead posterior predictive density for the observations can be computed as follows.
First we compute the one-step-ahead predictive density for latent states:

$$p(\mathbf{z}_t | \mathbf{y}_{1:t-1}) = \mathcal{N}(\mathbf{z}_t | \mathbf{A}_t \boldsymbol{\mu}_{t-1}, \mathbf{A}_t \boldsymbol{\Sigma}_{t-1} \mathbf{A}_t^\top + \mathbf{Q}_t) \quad (8.114)$$

Then we convert this to a prediction about observations by marginalizing out \mathbf{z}_t :

$$p(\mathbf{y}_t | \mathbf{y}_{1:t-1}) = \mathcal{N}(\mathbf{y}_t | \mathbf{C}\boldsymbol{\mu}_{t|t-1}, \mathbf{C}\boldsymbol{\Sigma}_{t|t-1}\mathbf{C}^\top + \mathbf{R}_t) \quad (8.115)$$

We can generalize this to predict observations K steps into the future by first forecasting K steps in latent space, and then “grounding” the final state into predicted observations. (This is in contrast to an RNN (Section 16.3.3), which requires generating observations at each step, in order to update future hidden states.) For notational simplicity, we just show how to do this for $K = 2$, i.e., we want 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 to know future inputs if the model is conditional.

We start with the prior, which is the previous posterior:

$$p(\mathbf{z}_{t-1} | \mathbf{y}_{1:t-1}) = \mathcal{N}(\mathbf{z}_{t-1} | \boldsymbol{\mu}_{t-1|t-1}, \boldsymbol{\Sigma}_{t-1|t-1}) \quad (8.116)$$

We predict the current hidden state as follows:

$$p(\mathbf{z}_t | \mathbf{y}_{1:t-1}) = \int \mathcal{N}(\mathbf{z}_t | \mathbf{A}\mathbf{z}_{t-1}, \mathbf{Q}) \mathcal{N}(\mathbf{z}_{t-1} | \boldsymbol{\mu}_{t-1|t-1}, \boldsymbol{\Sigma}_{t-1|t-1}) d\mathbf{z}_{t-1} \quad (8.117)$$

$$= \mathcal{N}(\mathbf{z}_t | \mathbf{A}\boldsymbol{\mu}_{t-1|t-1}, \mathbf{A}\boldsymbol{\Sigma}_{t-1|t-1}\mathbf{A}^\top + \mathbf{Q}) \quad (8.118)$$

$$\triangleq \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}) \quad (8.119)$$

We predict the next hidden state as follows:

$$p(\mathbf{z}_{t+1} | \mathbf{y}_{1:t-1}) = \int \mathcal{N}(\mathbf{z}_{t+1} | \mathbf{A}\mathbf{z}_t, \mathbf{Q}) \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}) d\mathbf{z}_t \quad (8.120)$$

$$= \mathcal{N}(\mathbf{z}_{t+1} | \mathbf{A}\boldsymbol{\mu}_{t|t-1}, \mathbf{A}\boldsymbol{\Sigma}_{t|t-1}\mathbf{A}^\top + \mathbf{Q}) \quad (8.121)$$

$$\triangleq \mathcal{N}(\mathbf{z}_{t+1} | \boldsymbol{\mu}_{t+1|t-1}, \boldsymbol{\Sigma}_{t+1|t-1}) \quad (8.122)$$

Finally we predict the next observation as follows:

$$p(\mathbf{y}_{t+1} | \mathbf{y}_{1:t-1}) = \int \mathcal{N}(\mathbf{y}_{t+1} | \mathbf{C}\mathbf{z}_{t+1}, \mathbf{R}) \mathcal{N}(\mathbf{z}_{t+1} | \boldsymbol{\mu}_{t+1|t-1}, \boldsymbol{\Sigma}_{t+1|t-1}) d\mathbf{z}_{t+1} \quad (8.123)$$

$$= \mathcal{N}(\mathbf{y}_{t+1} | \mathbf{C}\boldsymbol{\mu}_{t+1|t-1}, \mathbf{C}\boldsymbol{\Sigma}_{t+1|t-1}\mathbf{C}^\top + \mathbf{R}) \quad (8.124)$$

8.4.1.7 Numerical issues

In practice, the Kalman filter often encounters numerical issues, primarily related to the need to compute and invert the posterior covariance matrix. Recall from Equation (8.97) that the update step involves computing

$$\boldsymbol{\Sigma}_t = (\mathbf{I} - \mathbf{K}_t \mathbf{C}_t) \boldsymbol{\Sigma}_{t|t-1} \quad (8.125)$$

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:

$$\underline{\Sigma}_t = (\mathbf{I} - \mathbf{K}_t \mathbf{C}_t) \underline{\Sigma}_{t|t-1} (\mathbf{I} - \mathbf{K}_t \mathbf{C}_t)^T + \mathbf{K}_t \mathbf{R}_t \mathbf{K}_t^T \quad (8.126)$$

To see why this equation is true, first recall that the posterior mean estimate is given by

$$\underline{\mu}_{t|t} = \underline{\mu}_{t|t-1} + \mathbf{K}_t (\mathbf{y}_t - \mathbf{h}_t \underline{\mu}_{t|t-1}) \quad (8.127)$$

We have that $\mathbf{y}_t = \mathbf{C}_t \mathbf{z}_t + \boldsymbol{\epsilon}_t$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_t)$ is the observation noise. Hence we have the following, for any matrix \mathbf{K} :

$$\underline{\mathbf{z}}_t - \underline{\mu}_{t|t} = \mathbf{z}_t - (\underline{\mu}_{t|t-1} + \mathbf{K}(\mathbf{y}_t - \mathbf{C}_t \underline{\mu}_{t|t-1})) \quad (8.128)$$

$$= \mathbf{z}_t - (\underline{\mu}_{t|t-1} + \mathbf{K}(\mathbf{C}_t \mathbf{z}_t + \boldsymbol{\epsilon}_t - \mathbf{C}_t \underline{\mu}_{t|t-1})) \quad (8.129)$$

$$= (\mathbf{z}_t - \underline{\mu}_{t|t-1}) - \mathbf{K} \mathbf{C}_t (\mathbf{z}_t - \underline{\mu}_{t|t-1}) - \mathbf{K} \boldsymbol{\epsilon}_t \quad (8.130)$$

$$= (\mathbf{I} - \mathbf{K} \mathbf{C}_t) (\mathbf{z}_t - \underline{\mu}_{t|t-1}) - \mathbf{K} \boldsymbol{\epsilon}_t \quad (8.131)$$

Hence the posterior covariance of \mathbf{z}_t is given by

$$\underline{\Sigma}_{t|t} = \mathbb{E}[(\mathbf{z}_t - \underline{\mu}_{t|t})(\mathbf{z}_t - \underline{\mu}_{t|t})^T] \quad (8.132)$$

$$= \mathbb{E}[[(\mathbf{I} - \mathbf{K} \mathbf{C}_t)(\mathbf{z}_t - \underline{\mu}_{t|t-1}) - \mathbf{K} \boldsymbol{\epsilon}_t][(\mathbf{I} - \mathbf{K} \mathbf{C}_t)(\mathbf{z}_t - \underline{\mu}_{t|t-1}) - \mathbf{K} \boldsymbol{\epsilon}_t]^T] \quad (8.133)$$

$$= (\mathbf{I} - \mathbf{K} \mathbf{C}_t) \underline{\Sigma}_{t|t-1} (\mathbf{I} - \mathbf{K} \mathbf{C}_t)^T + \mathbf{K} \mathbf{R}_t \mathbf{K}^T \quad (8.134)$$

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:

$$p_0(\lambda) = \text{Ga}\left(\frac{\nu_0}{2}, \frac{\nu_0 \tau_0}{2}\right) \quad (8.137)$$

$$p_0(\mathbf{z} | \lambda) = \mathcal{N}(\mathbf{z}_0, V \Sigma_0^*) \quad (8.138)$$

¹ where τ_0 is the prior mean for σ^2 , and $\nu_0 > 0$ is the strength of this prior.

² We now discuss the belief updating step. We assume that the prior belief state at time $t - 1$ is

$$\mathcal{N}(\mathbf{z}_{t-1}, \lambda | \mathcal{D}_{1:t-1}) = \mathcal{N}(\mathbf{z}_{t-1} | \boldsymbol{\mu}_{t-1}, V \boldsymbol{\Sigma}_{t-1}^*) \text{Ga}(\lambda | \frac{\nu_{t-1}}{2}, \frac{\nu_{t-1} \tau_{t-1}}{2}) \quad (8.139)$$

³ The posterior is given by

$$\mathcal{N}(\mathbf{z}_t, \lambda | \mathcal{D}_{1:t}) = \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_t, V \boldsymbol{\Sigma}_t^*) \text{Ga}(\lambda | \frac{\nu_t}{2}, \frac{\nu_t \tau_t}{2}) \quad (8.140)$$

⁴ where

$$\boldsymbol{\mu}_{t|t-1} = \mathbf{A}_t \boldsymbol{\mu}_{t-1} \quad (8.141)$$

$$\boldsymbol{\Sigma}_{t|t-1}^* = \mathbf{A}_t \boldsymbol{\Sigma}_{t-1}^* \mathbf{A}_t + \mathbf{Q}_t^* \quad (8.142)$$

$$e_t = y_t - \mathbf{h}_t^\top \boldsymbol{\mu}_{t|t-1} \quad (8.143)$$

$$s_t^* = \mathbf{h}_t^\top \boldsymbol{\Sigma}_{t|t-1}^* \mathbf{h}_t + 1 \quad (8.144)$$

$$\mathbf{k}_t = \boldsymbol{\Sigma}_{t|t-1}^* \mathbf{h}_t / s_t^* \quad (8.145)$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + \mathbf{k}_t e_t \quad (8.146)$$

$$\boldsymbol{\Sigma}_t^* = \boldsymbol{\Sigma}_{t|t-1}^* - \mathbf{k}_t \mathbf{k}_t^\top s_t^* \quad (8.147)$$

$$\nu_t = \nu_{t-1} + 1 \quad (8.148)$$

$$\nu_t \tau_t = \nu_{t-1} \tau_{t-1} + e_t^2 / s_t^* \quad (8.149)$$

²⁴ If we marginalize out V , the marginal distribution for \mathbf{z}_t is a Student distribution:

$$p(\mathbf{z}_t | \mathcal{D}_{1:t}) = \mathcal{T}_{\nu_t}(\mathbf{z}_t | \boldsymbol{\mu}_t, \tau_t \boldsymbol{\Sigma}_t^*) \quad (8.150)$$

²⁵

²⁶ 8.4.2 Kalman filtering for linear regression (recursive least squares)

²⁷

²⁸ 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,
²⁹ 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
³⁰ this posterior online, by repeatedly performing the following update:

$$p(\mathbf{w} | \mathcal{D}_{1:t}) \propto p(\mathcal{D}_t | \mathbf{w}) p(\mathbf{w} | \mathcal{D}_{1:t-1}) \quad (8.151)$$

³¹

$$\propto p(\mathcal{D}_t | \mathbf{w}) p(\mathcal{D}_{t-1} | \mathbf{w}) \dots p(\mathcal{D}_1 | \mathbf{w}) p(\mathbf{w}) \quad (8.152)$$

³²

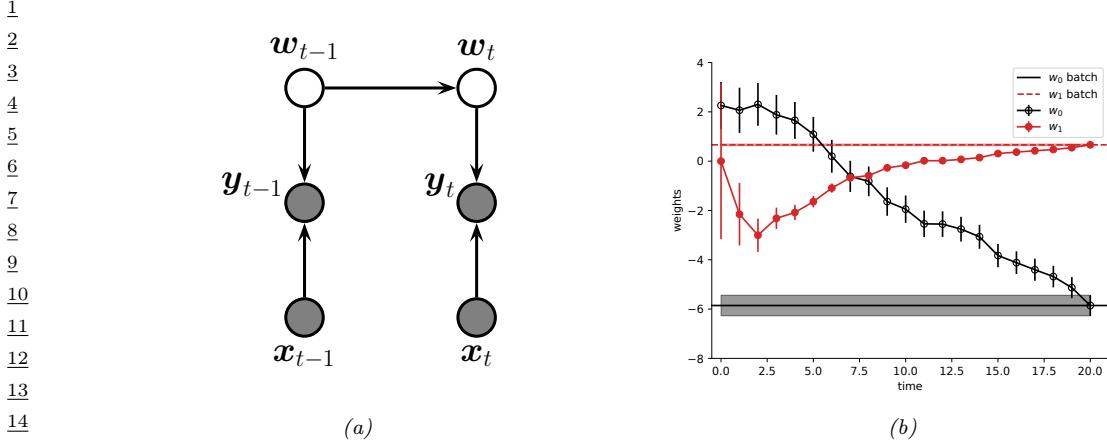
³³ where $\mathcal{D}_t = (\mathbf{x}_t, y_t)$ is the t 'th labeled example, and $\mathcal{D}_{1:t-1}$ are the first $t - 1$ examples. (For brevity,
³⁴ we drop the conditioning on σ^2 .) We see that the previous posterior, $p(\mathbf{w} | \mathcal{D}_{1:t-1})$, becomes the
³⁵ current prior, which gets updated by \mathcal{D}_t to become the new posterior, $p(\mathbf{w} | \mathcal{D}_{1:t})$. This is an example
³⁶ of sequential Bayesian updating or online Bayesian inference. In the case of linear regression, this
³⁷ process is known as the **recursive least squares** or **RLS** algorithm.

³⁸

³⁹ We can implement this method by using a linear Gaussian state space model (Section 31.2). The
⁴⁰ basic idea is to let the hidden state represent the regression parameters, and to let the (time-varying)
⁴¹ observation model represent the current data vector. If we assume the regression parameters do not
⁴² change, the dynamics model becomes

$$p(\mathbf{w}_t | \mathbf{w}_{t-1}) = \mathcal{N}(\mathbf{w}_t | \mathbf{w}_{t-1}, 0) = \delta(\mathbf{w}_t - \mathbf{w}_{t-1}) \quad (8.153)$$

⁴³



16 *Figure 8.9: (a) A dynamic generalization of linear regression. (b) Illustration of the recursive least squares*
17 *algorithm applied to the model $p(y|x, \mathbf{w}) = \mathcal{N}(y|w_0 + w_1x, \sigma^2)$. We plot the marginal posterior of w_0 and w_1*
18 *vs number of data points. (Error bars represent $\mathbb{E}[w_j|y_{1:t}, \mathbf{x}_{1:t}] \pm \sqrt{\text{Var}[w_j|y_{1:t}, \mathbf{x}_{1:t}]}$.) 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](#).*

21
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
24

$$25 \quad p(y_t|\mathbf{w}_t, \mathbf{x}_t) = \mathcal{N}(y_t|\mathbf{C}_t z_t, \mathbf{R}_t) = \mathcal{N}(y_t|\mathbf{x}_t^\top \mathbf{w}_t, \sigma^2) \quad (8.154)$$

26 See Figure 8.9a for the model.

27 Recall from Section 8.4.1 that the Kalman filter equations are as follows:

$$28 \quad \Sigma_{t|t-1} = \mathbf{A}_t \Sigma_{t-1} \mathbf{A}_t^\top + \mathbf{Q}_t \quad (8.155)$$

$$29 \quad \mathbf{S}_t = \mathbf{C}_t \Sigma_{t|t-1} \mathbf{C}_t^\top + \mathbf{R}_t \quad (8.156)$$

$$30 \quad \mathbf{K}_t = \Sigma_{t|t-1} \mathbf{C}_t^\top \mathbf{S}_t^{-1} \quad (8.157)$$

$$31 \quad \boldsymbol{\mu}_t = \mathbf{A}_{t-1} \boldsymbol{\mu}_{t-1} + \mathbf{K}_t (y_t - \mathbf{C}_t \mathbf{A}_{t-1} \boldsymbol{\mu}_{t-1}) \quad (8.158)$$

$$32 \quad \Sigma_t = (\mathbf{I} - \mathbf{K}_t \mathbf{C}_t) \Sigma_{t|t-1} \quad (8.159)$$

33 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
34 remaining equations simplify as follows:
35

$$36 \quad s_t = \mathbf{x}_t^\top \Sigma_{t-1} \mathbf{x}_t + \sigma^2 \quad (8.160)$$

$$37 \quad \mathbf{k}_t = \frac{1}{s_t} \Sigma_{t-1} \mathbf{x}_t \quad (8.161)$$

$$38 \quad \boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + \mathbf{k}_t (y_t - \mathbf{x}_t^\top \boldsymbol{\mu}_{t-1}) = \boldsymbol{\mu}_{t-1} + \frac{1}{s_t} \Sigma_{t-1} \mathbf{x}_t (y_t - \mathbf{x}_t^\top \boldsymbol{\mu}_{t-1}) \quad (8.162)$$

$$39 \quad \Sigma_t = (\mathbf{I} - \mathbf{k}_t \mathbf{x}_t^\top) \Sigma_{t-1} = \Sigma_{t-1} - \frac{1}{s_t} \Sigma_{t-1} \mathbf{x}_t \mathbf{x}_t^\top \Sigma_{t-1} \quad (8.163)$$

Note that from Equation (8.99), we can also write the Kalman gain as

$$k_t = \frac{1}{\sigma^2} (\Sigma_{t-1}^{-1} + \frac{1}{\sigma^2} x_t x_t^\top)^{-1} x_t \quad (8.164)$$

⁶ Also, from Equation (8.97), we can also write the posterior covariance as

$$\frac{1}{8} \Sigma_t = \Sigma_{t-1} - s_t k_t k_t^\top \quad (8.165)$$

If we let $\mathbf{V}_t = \Sigma_t / \sigma^2$, we can further simplify the equations, as follows [Bor16].

$$\mu_t = \mu_{t-1} + \frac{\sigma^2 \mathbf{V}_{t-1} x_t (\mathbf{y}_t - \mathbf{x}_t^\top \boldsymbol{\mu}_{t-1})}{\sigma^2 (\mathbf{x}_t^\top \mathbf{V}_{t-1} \mathbf{x}_t + 1)} = \mu_{t-1} + \frac{\mathbf{V}_{t-1} x_t (\mathbf{y}_t - \mathbf{x}_t^\top \boldsymbol{\mu}_{t-1})}{\mathbf{x}_t^\top \mathbf{V}_{t-1} \mathbf{x}_t + 1} \quad (8.166)$$

$$\mathbf{V}_t = \mathbf{V}_{t-1} - \frac{\mathbf{V}_{t-1} \mathbf{x}_t \mathbf{x}_t^\top \mathbf{V}_{t-1}}{\mathbf{x}_t^\top \mathbf{V}_{t-1} \mathbf{x}_t + 1} \quad (8.167)$$

¹⁶ We can initialize these recursions using a vague prior, $\mu_0 = \mathbf{0}$, $\Sigma_0 = \infty \mathbf{I}$. In this case, the posterior
¹⁷ mean will converge to the MLE, and the posterior standard deviations will converge to the standard
¹⁸ 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.

27 28 8.4.3 Predictive coding as Kalman filtering

²⁹ In the field of neuroscience, a popular theoretical model for how the brain works is known as
³⁰ **predictive coding** (see e.g., [Rao99; Fri03; Spr17; MSB21; Mar21]). This posits that the core
³¹ function of the brain is simply to minimize prediction error at each layer of a hierarchical model,
³² and at each moment in time (c.f., Figure 28.1). There is considerable biological evidence for this (see
³³ above references). Furthermore, it turns out that the predictive coding algorithm, when applied to a
³⁴ 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

$$\frac{41}{42} \quad \mathcal{F}_t(\psi_t) = \mathbb{E}_{q(\mathbf{z}_{t-1})} [D_{\text{KL}}(q(\mathbf{z}_t) \| p(\mathbf{z}_t, \mathbf{y}_t | \mathbf{z}_{t-1}))] \quad (8.168)$$

43 We will use a Gaussian approximation for q at each step. Furthermore, we will use the Laplace
44 approximation, which derives the covariance from the Hessian at the mean. Thus we have $q(\mathbf{z}_t) =$
45 $\mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_t, \boldsymbol{\Sigma}(\boldsymbol{\mu}_t))$, where $\boldsymbol{\psi}_t = \boldsymbol{\mu}_t$ is the variational parameter which we need to compute. (Once we
46 have computed $\boldsymbol{\mu}_t$, we can derive $\boldsymbol{\Sigma}$.)

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, $\boldsymbol{\epsilon}_z = \boldsymbol{\mu}_t - \mathbf{A}\boldsymbol{\mu}_{t-1} - \mathbf{B}\mathbf{u}_{t-1}$, and one from the current observation, $\boldsymbol{\epsilon}_y = \mathbf{y}_t - \mathbf{C}\boldsymbol{\mu}_t$:

$$\nabla \mathcal{F}_t(\boldsymbol{\mu}_t) = 2\mathbf{C}^\top\Sigma_y^{-1}\mathbf{y}_t - (\mathbf{C}^\top\Sigma_y^{-1}\mathbf{C} + \mathbf{C}^\top\Sigma_z^{-1}\mathbf{C})\boldsymbol{\mu}_t \quad (8.178)$$

$$+ (\Sigma_z^{-1} + \Sigma_z - \mathbf{T})\boldsymbol{\mu}_t - 2\Sigma_z^{-1}\mathbf{A}\boldsymbol{\mu}_{t-1} - 2\Sigma_z^{-1}\mathbf{B}\mathbf{u}_{t-1} \quad (8.179)$$

$$= 2\mathbf{C}^\top\Sigma_y^{-1}\mathbf{C}\boldsymbol{\mu}_t - 2\mathbf{C}^\top\Sigma_y^{-1}\mathbf{C}\boldsymbol{\mu}_t + 2\Sigma_z^{-1}\boldsymbol{\mu}_t - 2\Sigma_z^{-1}\mathbf{A}\boldsymbol{\mu}_{t-1} - 2\Sigma_z^{-1}\mathbf{B}\mathbf{u}_{t-1} \quad (8.180)$$

$$= -\mathbf{C}^\top\Sigma_y^{-1}[\mathbf{y}_t - \mathbf{C}\boldsymbol{\mu}_t] + \Sigma_z^{-1}[\boldsymbol{\mu}_t - \mathbf{A}\boldsymbol{\mu}_{t-1} - \mathbf{B}\mathbf{u}_{t-1}] \quad (8.181)$$

$$= -\mathbf{C}^\top\Sigma_y^{-1}\boldsymbol{\epsilon}_y + \Sigma_z^{-1}\boldsymbol{\epsilon}_z \quad (8.182)$$

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 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]).

¹ 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
² 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
³ $\nabla_{\mathbf{B}} \mathcal{F}_t = -\boldsymbol{\Sigma}_z \boldsymbol{\epsilon}_y \boldsymbol{u}_{t-1}^\top$, 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
⁴ expressions can be generalized to nonlinear models. Indeed, predictive coding can in fact approximate
⁵ backpropagation for many kinds of model [MTB20].

⁶ Gradient descent using these predicting coding takes the form of a **Hebbian update rule**, in
⁷ which we set the new parameter to the old one plus a term that is a multiplication of the two
⁸ quantities available at each end of the synaptic connection, namely the prediction error ϵ as input,
⁹ and the value μ (or θ) of the neuron as output. However, there are still several aspects of this
¹⁰ model that are biologically implausible, such as assuming symmetric weights (since both \mathbf{C} and \mathbf{C}^\top
¹¹ 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
¹² alignment of error signals and parameter values, and the need (in the nonlinear case) for computing
¹³ the derivative of the activation function. In [Mil+21] they develop an approximate, more biologically
¹⁴ plausible version of predictive coding that relaxes these requirements, and which does not seem to
¹⁵ hurt empirical performance too much.

¹⁶

¹⁷

¹⁸ 8.4.4 The Kalman (RTS) smoother

¹⁹ In Section 8.4.1, we described the Kalman filter, which sequentially computes $p(\mathbf{z}_t | \mathbf{y}_{1:t})$ for each t .
²⁰ This is useful for online inference problems, such as tracking. However, in an offline setting, we can
²¹ wait until all the data has arrived, and then compute $p(\mathbf{z}_t | \mathbf{y}_{1:T})$. By conditioning on past and future
²² data, our uncertainty will be significantly reduced. This is illustrated in Figure 8.5(c), where we see
²³ that the posterior covariance ellipsoids are smaller for the smoothed trajectory than for the filtered
²⁴ trajectory.

²⁵ We now explain how to compute the smoothed estimates, using an algorithm called the **RTS**
²⁶ **smoother**, named after its inventors, Rauch, Tung and Striebel [RTS65]. It is also known as the
²⁷ **Kalman smoothing** algorithm. The algorithm is the linear-Gaussian analog to the the forwards-
²⁸ filtering backwards-smoothing algorithm for HMMs.

²⁹

³⁰

³¹ 8.4.4.1 Algorithm

³² One can show (see below) that the backwards smoothing pass has the following recursive form:
³³

$$\mathbf{p}(\mathbf{z}_t | \mathbf{y}_{1:T}) = \mathcal{N}(\boldsymbol{\mu}_{t|T}, \boldsymbol{\Sigma}_{t|T}) \quad (8.183)$$

$$\boldsymbol{\mu}_{t|T} = \boldsymbol{\mu}_{t|t} + \mathbf{G}_t (\boldsymbol{\mu}_{t+1|T} - \boldsymbol{\mu}_{t+1|t}) \quad (8.184)$$

$$\boldsymbol{\Sigma}_{t|T} = \boldsymbol{\Sigma}_{t|t} + \mathbf{G}_t (\boldsymbol{\Sigma}_{t+1|T} - \boldsymbol{\Sigma}_{t+1|t}) \mathbf{G}_t^\top \quad (8.185)$$

$$\mathbf{G}_t = \boldsymbol{\Sigma}_{t|t} \mathbf{A}_{t+1}^\top \boldsymbol{\Sigma}_{t+1|t}^{-1} \quad (8.186)$$

³⁴

³⁵ 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
³⁶ the filtering step. The algorithm can be initialized from $\boldsymbol{\mu}_{T|T}$ and $\boldsymbol{\Sigma}_{T|T}$ from the Kalman filter. See
³⁷ 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 $\boldsymbol{\Sigma}_{t|T}$).
³⁸

³⁹

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 \mathbf{f}^{-1} evaluated at \mathbf{y} :
2

$$\underline{3} \quad [\mathbf{J}]_{jj'} = \frac{\partial f_j^{-1}(\mathbf{u})}{\partial u_{j'}}|_{\mathbf{u}=\mathbf{y}} \quad (8.190)$$

4

5 In general this is intractable to compute, so we seek an approximation.
6

7 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
8 function \mathbf{f} as follows:
9

$$\underline{10} \quad \mathbf{f}(\mathbf{z}) = \mathbf{f}(\boldsymbol{\mu} + \delta\mathbf{z}) \approx \mathbf{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\delta\mathbf{z} + \sum_i \frac{1}{2}\delta\mathbf{z}^\top \mathbf{F}^i(\boldsymbol{\mu}) \delta\mathbf{z} \mathbf{e}_i + \dots \quad (8.191)$$

11

12 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} ,
13

$$\underline{15} \quad [\mathbf{F}(\boldsymbol{\mu})]_{jj'} = \frac{\partial f_j(\mathbf{z})}{\partial x_{j'}}|_{\mathbf{z}=\boldsymbol{\mu}} \quad (8.192)$$

16

17 and $\mathbf{F}^i(\boldsymbol{\mu})$ is the Hessian of $f_i(\cdot)$ computed at $\boldsymbol{\mu}$:
18

$$\underline{19} \quad [\mathbf{F}^i(\boldsymbol{\mu})]_{jj'} = \frac{\partial^2 f_i(\mathbf{z})}{\partial x_j \partial x_{j'}}|_{\mathbf{z}=\boldsymbol{\mu}} \quad (8.193)$$

20

21 We can create a linear approximation by just using the first two terms:
22

$$\underline{24} \quad \hat{\mathbf{f}}(\mathbf{z}) = \mathbf{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\delta\mathbf{z} \quad (8.194)$$

25

26 We now derive the induced Gaussian approximation to $\mathbf{y} = \mathbf{f}(\mathbf{z})$. The mean is given by
27

$$\underline{28} \quad \mathbb{E}[\mathbf{y}] \approx \mathbb{E}[\mathbf{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\delta\mathbf{z}] \quad (8.195)$$

29

$$\underline{30} \quad = \mathbf{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\mathbb{E}[\delta\mathbf{z}] \quad (8.196)$$

31

$$\underline{32} \quad = \mathbf{f}(\boldsymbol{\mu}) \quad (8.197)$$

33

34 The covariance is given by
35

$$\underline{36} \quad \text{Cov}[\mathbf{y}] = \mathbb{E}[(\mathbf{f}(\mathbf{z}) - \mathbb{E}[\mathbf{f}(\mathbf{z})])(\mathbf{f}(\mathbf{z}) - \mathbb{E}[\mathbf{f}(\mathbf{z})])^\top] \quad (8.198)$$

37

$$\underline{38} \quad \approx \mathbb{E}[(\mathbf{f}(\mathbf{z}) - \mathbf{f}(\boldsymbol{\mu}))(\mathbf{f}(\mathbf{z}) - \mathbf{f}(\boldsymbol{\mu}))^\top] \quad (8.199)$$

39

$$\underline{40} \quad \approx \mathbb{E}[(\mathbf{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\delta\mathbf{z} - \mathbf{f}(\boldsymbol{\mu}))(\mathbf{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\delta\mathbf{z} - \mathbf{f}(\boldsymbol{\mu}))^\top] \quad (8.200)$$

41

$$\underline{42} \quad = \mathbb{E}[(\mathbf{F}(\boldsymbol{\mu})\delta\mathbf{z})(\mathbf{F}(\boldsymbol{\mu})\delta\mathbf{z})^\top] \quad (8.201)$$

43

$$\underline{44} \quad = \mathbf{F}(\boldsymbol{\mu})\mathbb{E}[\delta\mathbf{z}\delta\mathbf{z}^\top]\mathbf{F}(\boldsymbol{\mu})^\top \quad (8.202)$$

45

$$\underline{46} \quad = \mathbf{F}(\boldsymbol{\mu})\boldsymbol{\Sigma}\mathbf{F}(\boldsymbol{\mu})^\top \quad (8.203)$$

47

48 Let us consider a 1d example. In Figure 8.10a, we show what happens when we pass a Gaussian
49 distribution $p(x)$, shown on the bottom right, through a nonlinear function $y = f(x)$, shown on the
50 top right. The resulting distribution (approximated by Monte Carlo) is shown in the shaded gray
51 area in the top left corner. The best Gaussian approximation to this, computed from $\mathbb{E}[f(x)]$ and
52

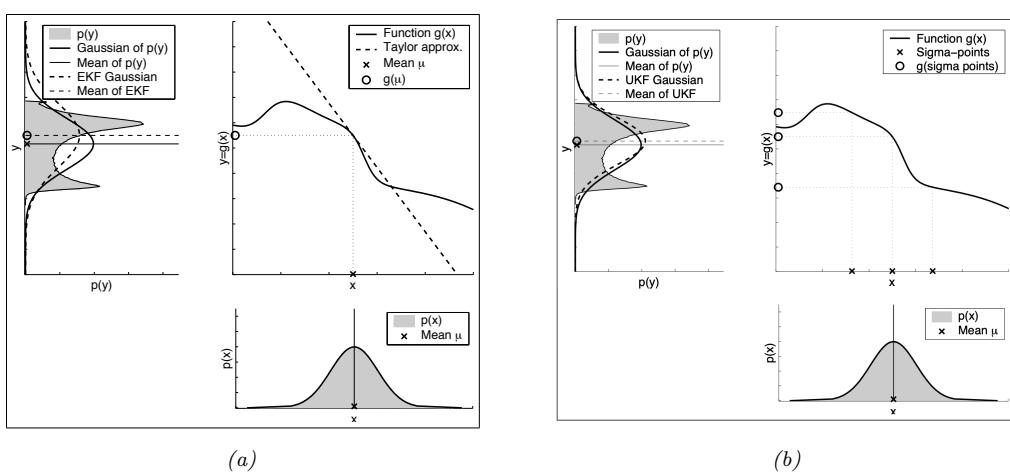


Figure 8.10: Nonlinear transformation of a Gaussian random variable. The prior $p(x)$ is shown on the bottom right. The function $y = f(x)$ is shown on the top right. The transformed distribution $p(y)$ is shown in the top left. A linear function induces a Gaussian distribution, but a non-linear function 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

$$\mathbb{E}[\tilde{f}(z)] \approx \begin{pmatrix} \mu \\ f(\mu) \end{pmatrix} \quad (8.204)$$

$$\text{Cov}[\tilde{f}(z)] \approx \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{F}(\mu) & \mathbf{I} \end{pmatrix} \Sigma \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{F}(\mu) & \mathbf{I} \end{pmatrix}^\top = \begin{pmatrix} \Sigma & \Sigma \mathbf{F}(\mu) \\ \mathbf{F}(\mu) \Sigma & \mathbf{F}(\mu) \Sigma \mathbf{F}(\mu)^\top \end{pmatrix} \quad (8.205)$$

When deriving the EKF, we need the following slightly more general version:

$$z \sim \mathcal{N}(\mu, \Sigma), \quad y = f(z) + q, \quad q \sim \mathcal{N}(\mathbf{0}, Q) \quad (8.206)$$

The resulting linear approximation to the joint is

$$\begin{pmatrix} z \\ y \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu \\ \mu_L \end{pmatrix}, \begin{pmatrix} \Sigma & \mathbf{C}_L \\ \mathbf{C}_L^\top & \mathbf{S}_L \end{pmatrix}\right) \quad (8.207)$$

$$\mu_L = f(\mu) \quad (8.208)$$

$$\mathbf{S}_L = \mathbf{F}(\mu) \Sigma \mathbf{F}(\mu)^\top + Q \quad (8.209)$$

$$\mathbf{C}_L = \Sigma \mathbf{F}(\mu)^\top \quad (8.210)$$

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

$$\mathbf{F}_t = \frac{\partial \mathbf{f}_t(\mathbf{z})}{\partial \mathbf{z}}|_{\boldsymbol{\mu}_{t-1}} \quad (8.211)$$

$$\mathbf{H}_t = \frac{\partial \mathbf{h}_t(\mathbf{z})}{\partial \mathbf{z}}|_{\boldsymbol{\mu}_{t|t-1}} \quad (8.212)$$

¹⁶ The updates then become
¹⁷

$$\boldsymbol{\mu}_{t|t-1} = \mathbf{f}(\boldsymbol{\mu}_{t-1}) \quad (8.213)$$

$$\boldsymbol{\Sigma}_{t|t-1} = \mathbf{F}_t \boldsymbol{\Sigma}_{t-1} \mathbf{F}_t + \mathbf{Q}_t \quad (8.214)$$

$$\mathbf{e}_t = \mathbf{y}_t - \mathbf{h}(\boldsymbol{\mu}_{t|t-1}) \quad (8.215)$$

$$\mathbf{S}_t = \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top + \mathbf{R}_t \quad (8.216)$$

$$\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t \mathbf{S}_t^{-1} \quad (8.217)$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t \mathbf{e}_t \quad (8.218)$$

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} = \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t \mathbf{S}_t \mathbf{K}_t^\top \quad (8.219)$$

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

³⁸ 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
³⁹

$$p(\mathbf{z}_{t-1}, \mathbf{z}_t | \mathbf{y}_{1:t-1}) \approx \mathcal{N}\left(\begin{pmatrix} \mathbf{z}_{t-1} \\ \mathbf{z}_t \end{pmatrix} | \mathbf{m}', \boldsymbol{\Sigma}'\right) \quad (8.220)$$

$$\mathbf{m}' = \begin{pmatrix} \boldsymbol{\mu}_{t-1} \\ \mathbf{f}(\boldsymbol{\mu}_{t-1}) \end{pmatrix} \quad (8.221)$$

$$\boldsymbol{\Sigma}' = \begin{pmatrix} \boldsymbol{\Sigma}_{t-1} & \boldsymbol{\Sigma}_{t-1} \mathbf{F}_t^\top \\ \mathbf{F}_t \boldsymbol{\Sigma}_{t-1} & \mathbf{F}_t \boldsymbol{\Sigma}_{t-1} \mathbf{F}_t^\top + \mathbf{Q}_{t-1} \end{pmatrix} \quad (8.222)$$

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:

$$p(\mathbf{z}_t, \mathbf{y}_t|\mathbf{y}_{1:t-1}) \approx \mathcal{N}\left(\begin{pmatrix} \mathbf{z}_t \\ \mathbf{y}_t \end{pmatrix} | \mathbf{m}'', \boldsymbol{\Sigma}''\right) \quad (8.223)$$

$$\mathbf{m}'' = \begin{pmatrix} \boldsymbol{\mu}_{t|t-1} \\ \mathbf{h}(\boldsymbol{\mu}_{t|t-1}) \end{pmatrix} \quad (8.224)$$

$$\boldsymbol{\Sigma}'' = \begin{pmatrix} \boldsymbol{\Sigma}_{t|t-1} & \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top \\ \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} & \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top + \mathbf{R}_{t-1} \end{pmatrix} \quad (8.225)$$

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 :

$$\mathbf{F}_t^i = \frac{\partial \mathbf{f}_t^i(\mathbf{z})}{\partial \mathbf{z}}|_{\boldsymbol{\mu}_{t-1}} \quad (8.229)$$

$$\mathbf{H}_t^i = \frac{\partial \mathbf{h}_t^i(\mathbf{z})}{\partial \mathbf{z}}|_{\boldsymbol{\mu}_{t|t-1}} \quad (8.230)$$

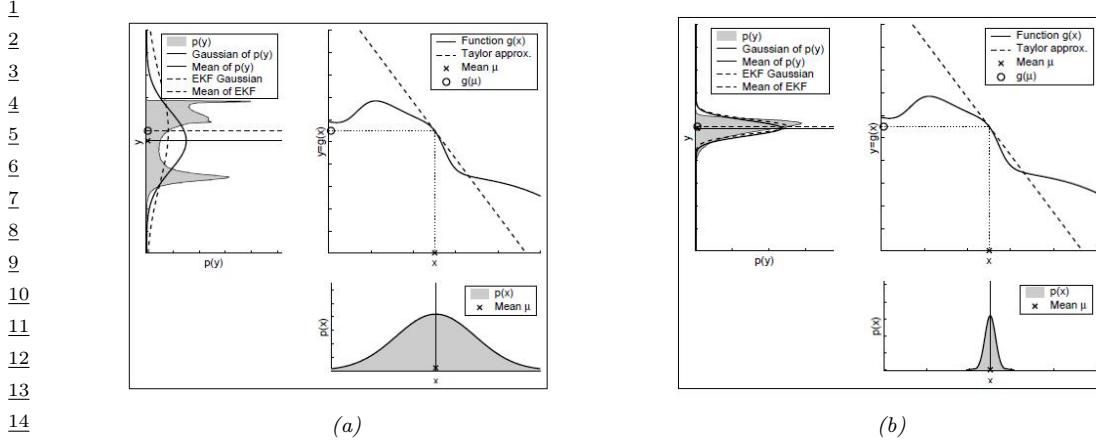


Figure 8.11: Illustration of the fact that a broad prior (a) may result in a more complex posterior than a narrow prior (b). Consequently, the EKF approximation may work poorly in situations of high uncertainty. From Figure 3.5 of [TBF06]. Used with kind permission of Dieter Fox.

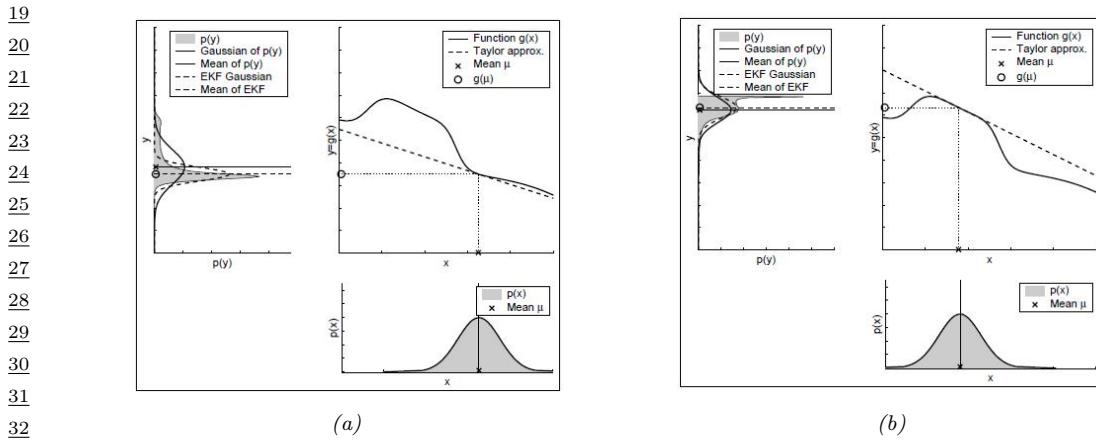


Figure 8.12: Illustration of the fact that if the function function is very nonlinear (a) at the current operating point, the posterior will be less well approximated by the EKF than if the function is locally linear (b). From Figure 3.6 of [TBF06]. Used with kind permission of Dieter Fox.

We then compute the following updates for each block:

$$\mu_{t|t-1}^i = f^i(\mu_{t-1}) \quad (8.231)$$

$$\Sigma_{t|t-1}^i = (\mathbf{F}_t^i)^T \Sigma_{t-1}^i \mathbf{F}_t^i + \mathbf{Q}_{t-1}^i \quad (8.232)$$

$$\mathbf{S}_t = \sum_i (\mathbf{H}_t^i)^T \Sigma_{t|t-1}^i \mathbf{H}_t^i + \mathbf{R}_t \quad (8.233)$$

$$\mathbf{K}_t^i = \Sigma_{t|t-1}^i \mathbf{H}_t^i \mathbf{S}_t^{-1} \quad (8.234)$$

$$\mu_t^i = \mu_{t|t-1} + \mathbf{K}_t^i e_t \quad (8.235)$$

$$\Sigma_t^i = \Sigma_{t|t-1}^i - \mathbf{K}_t^i \mathbf{H}_t^i \Sigma_{t|t-1}^i \quad (8.236)$$

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:

$$\boldsymbol{\mu}_{t|t-1} = f(\boldsymbol{\mu}_{t-1}, \mathbf{u}_t) \quad (8.244)$$

$$\mathbf{F}_t = \frac{\partial f}{\partial \mathbf{z}}|_{(\boldsymbol{\mu}_{t-1}, \mathbf{u}_t)} \quad (8.245)$$

$$\boldsymbol{\Sigma}_{t|t-1} = \mathbf{F}_t \boldsymbol{\Sigma}_{t-1} \mathbf{F}_t^\top + \mathbf{Q}_t \quad (8.246)$$

$$\hat{\mathbf{y}}_t = h(\boldsymbol{\mu}_{t|t-1}, \mathbf{u}_t) \quad (8.247)$$

1 Second, after seeing observation \mathbf{y}_t , we compute the following:
2

$$\underline{3} \quad \mathbf{e}_t = \mathcal{T}(\mathbf{y}_t) - \hat{\mathbf{y}}_t \quad (8.248)$$

$$\underline{4} \quad \mathbf{R}_t = \text{Cov} [\mathcal{T}(\mathbf{y}) | \hat{\mathbf{y}}_t] \quad (8.249)$$

5 where $\mathcal{T}(\mathbf{y})$ is the vector of sufficient statistics, and \mathbf{e}_t is the error or innovation term. (For a Gaussian
6 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.)
7

8 Finally we perform the update:
9

$$\underline{10} \quad \mathbf{H}_t = \frac{\partial h}{\partial \mathbf{z}}|_{(\boldsymbol{\mu}_{t|t-1}, \mathbf{u}_t)} \quad (8.250)$$

$$\underline{11} \quad \mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top (\mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top + \mathbf{R}_t)^{-1} \quad (8.251)$$

$$\underline{12} \quad \boldsymbol{\Sigma}_t = (\mathbf{I} - \mathbf{K}_t \mathbf{H}_t) \boldsymbol{\Sigma}_{t|t-1} \quad (8.252)$$

$$\underline{13} \quad \boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t \mathbf{e}_t \quad (8.253)$$

14 In [Oll18], they show that this is equivalent to an online version of natural gradient descent
15 (Section 6.4).
16

17 8.5.4.3 EEKF for training logistic regression

18 For example, consider the case where y is a class label with C possible values. (We drop the time
19 index for brevity.) Following Section 2.5.2.2, Let

$$\underline{20} \quad \mathcal{T}(\mathbf{y}) = [\mathbb{I}(y=1), \dots, \mathbb{I}(y=C-1)] \quad (8.254)$$

21 be the $(C-1)$ -dimensional vector of sufficient statistics, and let $\hat{\mathbf{y}} = [p_1, \dots, p_{C-1}]$ be the corre-
22 sponding predicted probabilities of each class label. The probability of the C 'th class is given by
23 $p_C = 1 - \sum_{c=1}^{C-1} \hat{y}_c$; we avoid including this to ensure that \mathbf{R} is not singular. The $(C-1) \times (C-1)$
24 covariance matrix \mathbf{R} is given by
25

$$\underline{26} \quad R_{ij} = \text{diag}(p_i) - p_i p_j \quad (8.255)$$

27 Now consider the simpler case where we have two class labels, so $C = 2$. In this case, $\mathcal{T}(\mathbf{y}) =$
28 $\mathbb{I}(y=1)$, and $\hat{\mathbf{y}} = p(y=1) = p$. The covariance matrix of the observation noise becomes the scalar
29 $r = p(1-p)$. Of course, we can make the output probabilities depend on the input covariates, as
30 follows:
31

$$\underline{32} \quad p(y_t | \mathbf{z}_t, \mathbf{u}_t) = \text{Ber}(y_t | \boldsymbol{\sigma}(\mathbf{z}_t^\top \mathbf{u}_t)) \quad (8.256)$$

33 We can use the exponential family representation of the output discussed in Section 8.5.4.3.
34

35 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
36 sequentially compute the posterior using the EEKF, and compare to the offline estimate computed
37 using a Laplace approximation (where the MAP estimate is computed using BFGS) and an MCMC
38 approximation, which we take as “ground truth”. In Figure 8.13c, we see that the resulting posterior
39 predictive distributions are similar. Finally, in Figure 8.14, we visualize how the posterior marginals
40 converge over time. (See also Section 8.8.4, where we solve this same problem using ADF.)
41

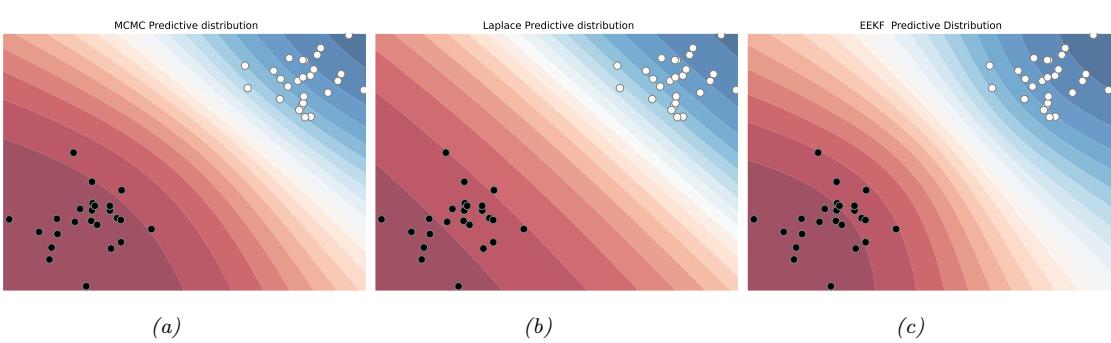


Figure 8.13: Bayesian inference applied to a 2d binary logistic regression problem, $p(y = 1|\mathbf{x}) = \sigma(w_0 + w_1x_1 + w_2x_2)$. We show the training data and the posterior predictive produced by different methods. (a) Offline MCMC approximation. (b) Offline Laplace approximation. (c) Online EEFK approximation at the final step of inference. Generated by `eekf_logistic_regression_demo.py`.

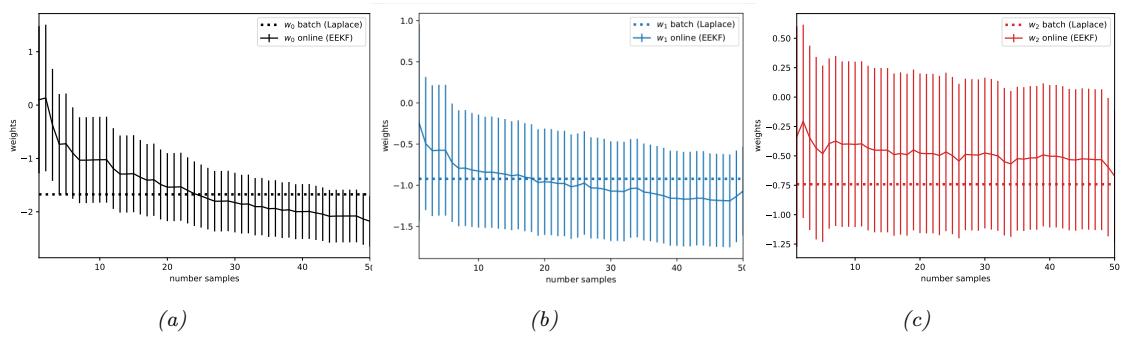


Figure 8.14: Marginal posteriors over time for the EEFK method. The horizontal line is the offline MAP estimate. Generated by `eekf_logistic_regression_demo.py`.

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

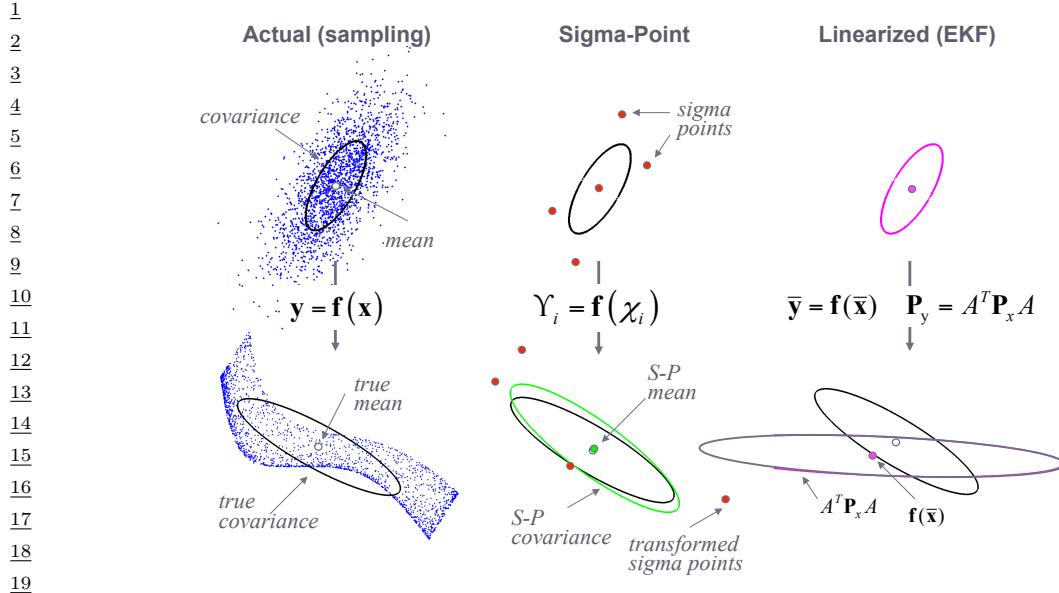


Figure 8.15: An example of the unscented transform in two dimensions. From [WM01]. Used with kind permission of Eric Wan.

[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 3. Estimate the mean and covariance of the resulting bag of points:

2

$$\mathbb{E}[\mathbf{f}(\mathbf{z})] \approx \boldsymbol{\mu}_U = \sum_{i=0}^{2d} w_i^m \mathbf{y}_i \quad (8.262)$$

3

$$\text{Cov}[\mathbf{f}(\mathbf{z})] \approx \mathbf{S}_U = \sum_{i=0}^{2d} w_i^c (\mathbf{y}_i - \boldsymbol{\mu}_U)(\mathbf{y}_i - \boldsymbol{\mu}_U)^T \quad (8.263)$$

4 where the w 's are weighting terms, given by the following:

5

$$w_0^m = \frac{\lambda}{d + \lambda} \quad (8.264)$$

6

$$w_0^c = \frac{\lambda}{d + \lambda} + (1 - \alpha^2 + \beta) \quad (8.265)$$

7

$$w_i^m = w_i^c = \frac{1}{2(d + \lambda)} \quad (8.266)$$

8 In general, the optimal values of α , β and κ are problem dependent. A typical recommendation
9 is $\alpha = 10^{-3}$, $\kappa = 1$, $\beta = 2$ [Bit16]. See Figure 8.10b for a 1d illustration and Figure 8.15 for a 2d
10 illustration.

11 Now suppose we want to approximate the joint distribution $p(\mathbf{z}, \mathbf{y})$, where $\mathbf{y} = \mathbf{f}(\mathbf{z}) + \mathbf{q}$, and
12 $\mathbf{q} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$. We have

13

$$\begin{pmatrix} \mathbf{z} \\ \mathbf{y} \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_U \end{pmatrix}, \begin{pmatrix} \Sigma & \mathbf{C}_U \\ \mathbf{C}_U^T & \mathbf{S}_u \end{pmatrix}\right) \quad (8.267)$$

14 where

15

$$\boldsymbol{\mu}_U = \sum_{i=0}^{2d} w_i^m \mathbf{y}_i \quad (8.268)$$

16

$$\mathbf{S}_U = \sum_{i=0}^{2d} w_i^c (\mathbf{y}_i - \boldsymbol{\mu}_U)(\mathbf{y}_i - \boldsymbol{\mu}_U)^T + \mathbf{Q} \quad (8.269)$$

17

$$\mathbf{C}_U = \sum_{i=0}^{2d} w_i^c (\mathbf{y}_i - \boldsymbol{\mu})(\mathbf{y}_i - \boldsymbol{\mu}_U)^T \quad (8.270)$$

18 The unscented transform is a third-order method in the sense that the mean of \mathbf{y} is exact for
19 polynomials up to order 3. However the covariance is only exact for linear functions.

20 8.6.2 The unscented Kalman filter (UKF)

21 The UKF uses the unscented transform twice, once to approximate passing through the system model
22 \mathbf{f} , and once to approximate passing through the measurement model \mathbf{h} . The derivation is analogous
23 to that of the EKF. The resulting algorithm is as follows.

8.6.2.1 Prediction step

³ We perform these steps.

5 1. Form the sigma points

$$z_{t-1,0} = \mu_{t-1} \quad (8.271)$$

$$z_{t-1,i} = \mu_{t-1} + \sqrt{d+\lambda} [\sqrt{\Sigma_{t-1}}]_{:i} \quad (8.272)$$

$$z_{t-1,i+n} = \mu_{t-1} - \sqrt{d+\lambda} [\sqrt{\Sigma_{t-1}}]_{:i} \quad (8.273)$$

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

$$\boldsymbol{\mu}_{t|t-1} = \sum_{i=1}^{2d} w_i^m \hat{\boldsymbol{z}}_{t,i} \quad (8.275)$$

$$\Sigma_{t|t-1} = \sum_{i=0}^{2d} w_i^c (\hat{z}_{t,i} - \mu_{t|t-1})(\hat{z}_{t,i} - \mu_{t|t-1})^\top + \mathbf{Q}_t \quad (8.276)$$

23 8.6.2.2 Update step

We perform these steps.

26 1. Form the sigma points

(8.277)

$$z_{t,i} \equiv \mu_{t|t-1} + \sqrt{d+\lambda} [\sqrt{\Sigma_{t|t-1}}]_{:,i} \quad (8.278)$$

$$z_{t,i+n} = \mu_{t|t-1} - \sqrt{d + \lambda} [\sqrt{\Sigma_{t|t-1}}]_{:i} \quad (8.279)$$

³³ 2. Propagate these points through the measurement model:

$$(8.280)$$

37 3. Compute the predicted mean and covariance of $p(\mathbf{y}_t, \mathbf{z}_t | \mathbf{y}_{t-1}, \mathbf{z}_{t-1})$:

$$\mathbf{m}_t = \sum^{2d} w_i^m \hat{\mathbf{z}}_{t,i} \quad (8.281)$$

$$\mathbf{S}_t = \sum_{i=0}^{2d} w_i^c (\hat{\mathbf{z}}_{t,i} - \mathbf{m}_t)(\hat{\mathbf{z}}_{t,i} - \mathbf{m}_t)^\top + \mathbf{R}_t \quad (8.282)$$

$$\mathbf{C}_t = \sum_{i=0}^{2d} w_i^c (\hat{\mathbf{z}}_{t,i} - \boldsymbol{\mu}_{t|t-1}) (\hat{\mathbf{z}}_{t,i} - \boldsymbol{m}_t)^T + \mathbf{R}_t \quad (8.283)$$

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

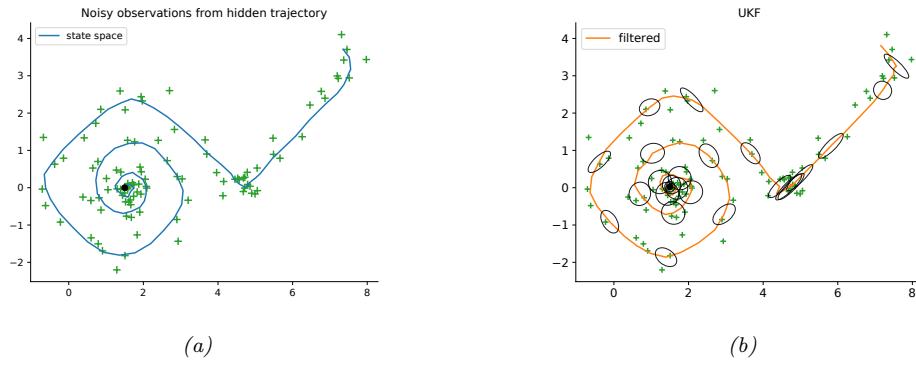


Figure 8.16: Illustration of UKF applied to a 2d nonlinear dynamical system. (a) True underlying state and observed data. (b) UKF estimate. Generated by `ekf_vs_ukf_demo.py`.

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].

⁶ **8.7.1 Ensemble Kalman filter**

⁸ The **ensemble Kalman filter (EnKF)** is a technique developed in the geoscience (meteorology)
⁹ community to perform approximate online inference in large nonlinear systems. The canonical
¹⁰ reference is [Eve09], but a more accessible tutorial (using the same Bayesian signal processing
¹¹ approach we adopt in this chapter) is in [Rot+17].

¹² EnKF is mostly used for problems where the hidden state represents an unknown physical quantity
¹³ (e.g., temperature and pressure) at each point on a spatial grid, and the measurements are sparse and
¹⁴ spatially localized. Combining this information over space and time is called **data assimilation**.
¹⁵ However, the technique can be applied to other nonlinear state estimation problems. For example, it
¹⁶ was recently used in [Li+20b] to model the spread of the SARS-CoV2 virus, using an ODE dynamics
¹⁷ model, based on the EnKF method described in [And01]. We briefly explain the method below,
¹⁸ following [Rot+17].

¹⁹ 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
²⁰ updated in a manner that closely resembles the Kalman filter, so there is no importance sampling or
²¹ resampling step. The downside is that the posterior does not converge to the true Bayesian posterior
²² even as $S \rightarrow \infty$ [LGMT11], except in the linear-Gaussian case. However, sometimes the performance
²³ of EnKF can be better for small number of samples (although this depends of course on the PF
²⁴ proposal distribution).

²⁵ The posterior mean and covariance can be derived from the ensemble of samples as follows:

$$\tilde{\mu}_{t|t} = \frac{1}{S} \sum_{s=1}^S \mathbf{z}_{t|t}^s = \frac{1}{S} \mathbf{z}_{t|t} \mathbf{1} \quad (8.292)$$

$$\tilde{\Sigma}_{t|t} = \frac{1}{S-1} \sum_{s=1}^S (\mathbf{z}_t^s - \tilde{\mu}_{t|t})(\mathbf{z}_t^s - \tilde{\mu}_{t|t})^\top = \frac{1}{S-1} \tilde{\mathbf{Z}}_{t|t} \tilde{\mathbf{Z}}_{t|t}^\top \quad (8.293)$$

³⁴ where $\tilde{\mathbf{Z}}_{t|t} = \mathbf{z}_{t|t} - \tilde{\mu}_{t|t} \mathbf{1}^\top$.

³⁵ We update the samples as follows. For the time update, we first draw S system noise variables
³⁶ $\mathbf{v}_t^s \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$, and then we pass these, and the previous state estimate, through the dynamics
³⁷ 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
³⁸ Equation (8.114).

³⁹ 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-
⁴⁰ 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).

⁴¹ Finally we compute the measurement update using

$$\mathbf{z}_{t|t} = \mathbf{z}_{t|t-1} + \tilde{\mathbf{K}}_t (\mathbf{y}_t \mathbf{1}^\top - \mathbf{y}_{t|t-1}) \quad (8.294)$$

⁴² which is the analog of Equation (8.95).

⁴³

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].

8.7.3.1 The Gaussian approximation

To explain the approach, we temporarily drop the time indices, and the conditioning on past information, and consider a single time step of inference. Furthermore, we will use the shorthand

$$\int_x^{\infty} f(x) = \int_{-\infty}^{\infty} f(x) dx \quad (8.299)$$

9 Let $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z)$ and $p(\mathbf{y}|\mathbf{z}) = \mathcal{N}(\mathbf{y}|g(\mathbf{z}), \mathbf{Q})$ for some function 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.,

$$q(z, y) = \mathcal{N} \left(\begin{pmatrix} z \\ y \end{pmatrix} \mid \begin{pmatrix} \mu_z \\ \mu_y \end{pmatrix}, \begin{pmatrix} \Sigma_z & \Sigma_{zy} \\ \Sigma_{yz}^\top & \Sigma_y \end{pmatrix} \right) \quad (8.300)$$

15
a. where

$$\underline{\frac{17}{18}} \quad \mu_y = \int g(z) \mathcal{N}(z|m, \Sigma) \quad (8.301)$$

$$\underline{\frac{19}{20}} \quad \Sigma_y = \int_z (g(z) - \mu_y)(g(z) - \mu_y)^\top \mathcal{N}(z|\mu_z, \Sigma_z) + Q \quad (8.302)$$

$$\Sigma_{zy} = \int_z^y (z - \mu_z)(g(z) - \mu_y)^\top \mathcal{N}(z|\mu_z, \Sigma_z) \quad (8.303)$$

We can either compute these integrals by linearizing \mathbf{g} and using closed form expressions, or by using numerical integration, as we discuss in Section 8.7.3.3.

Once we have computed the joint $q(\mathbf{z}, \mathbf{y})$, we can compute the posterior conditional $q(\mathbf{z}|\mathbf{y})$ using the usual rules for conditioning a Gaussian:

$$\underline{29} \quad q(\mathbf{z}|\mathbf{y}) = \mathcal{N} \left(\mathbf{z} | \underbrace{\boldsymbol{\mu}_z + \boldsymbol{\Sigma}_{zy} \boldsymbol{\Sigma}_y^{-1} (\mathbf{y} - \boldsymbol{\mu}_y)}_{\boldsymbol{\mu}_{z|y}}, \underbrace{\boldsymbol{\Sigma}_z - \boldsymbol{\Sigma}_{zy} \boldsymbol{\Sigma}_y^{-1} \boldsymbol{\Sigma}_{yz}^T}_{\boldsymbol{\Sigma}_{z|y}} \right) \quad (8.304)$$

33
34 In practice, Σ_y may be rank deficient, so we should avoid computing Σ_y^{-1} . Fortunately we can
35 compute $\mu_{z|x}$ and $\Sigma_{zz|x}$ in a numerically stable way by solving a linear system. In particular, the
36 posterior mean is the solution to

$$\frac{37}{36} \left(\mu_1 - \mu_2 + \sum_{i=1}^n \mu_i \right) = 0 \quad (8.305)$$

$$38 \quad \mu_{zy} = \mu_z + \Sigma_{zy} \alpha \quad (8.306) \\ 39 \quad \Sigma_z \alpha = \alpha_1 \quad (8.306)$$

⁴⁰ and the posterior covariance is the solution to

$$\frac{42}{\Sigma_{\alpha} \Sigma_{\beta} \Sigma_{\gamma} \Delta} \quad (8.207)$$

$$\Sigma_{zz|y} = \Sigma_z - \Sigma_{zy}\mathbf{A} \quad (8.301)$$

⁴⁵ See Tlali, 1995; if $\Sigma \geq 1$, then $t = 1$; if $\Sigma < 1$, then $t = 1 - \frac{1}{\Sigma}$. [Wu et al., 1996, A.1, A.2]

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.

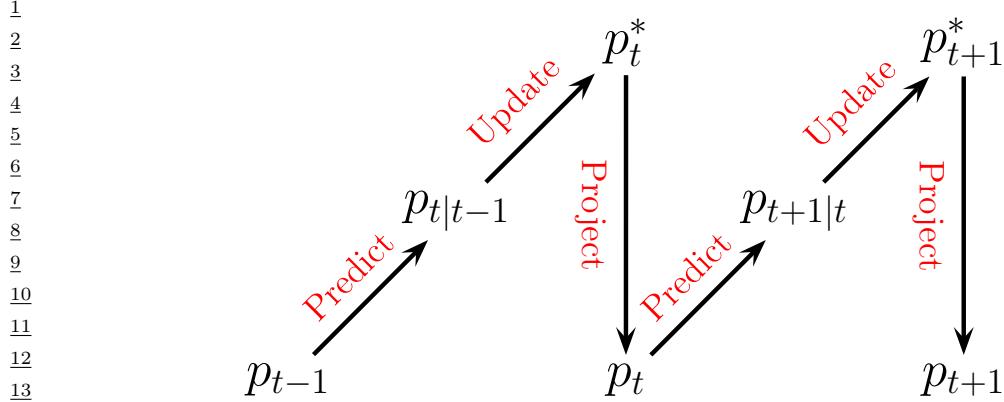


Figure 8.17: Illustration of the predict-update-project cycle of assumed density filtering.

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

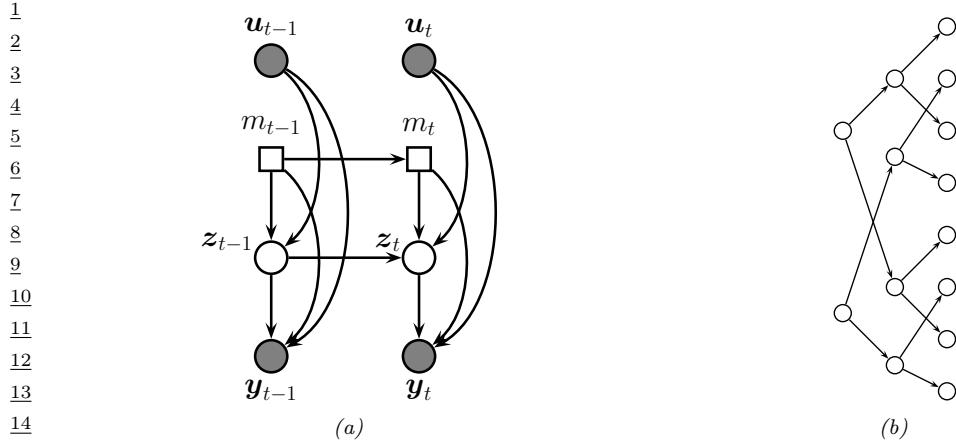


Figure 8.18: (a) A switching SSM. Squares represent discrete random variables, circles represent continuous random variables. (b) Illustration of how the number of modes in the belief state of a switching SSM grows exponentially over time. We assume there are two binary states.

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 .

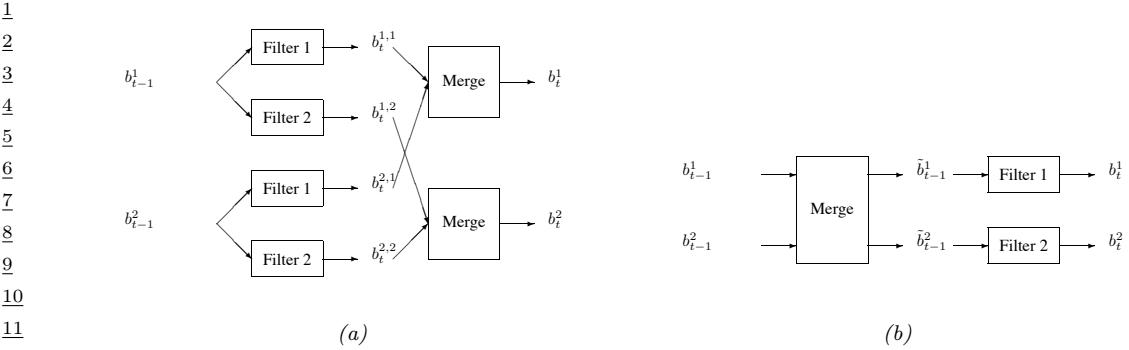


Figure 8.19: ADF for a switching linear dynamical system with 2 discrete states. (a) GPB2 method. (b) IMM method.

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

$$b_t^{ij} \triangleq p(\mathbf{z}_t, c_{t-1} = i, c_t = j | \mathbf{y}_{1:t}) = \pi_{tij} \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{t,ij}, \boldsymbol{\Sigma}_{t,ij}) \quad (8.331)$$

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 mixtures down to a single mixture to give

$$b_t^j \triangleq p(\mathbf{z}_t, c_t = j | \mathbf{y}_{1:t}) = \pi_{tj} \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{t,j}, \boldsymbol{\Sigma}_{t,j}) \quad (8.332)$$

See Figure 8.19a for a sketch.

The optimal way to approximate a mixture of Gaussians with a single Gaussian is given by $q = \arg \min_q D_{\text{KL}}(q \| p)$, 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 by moment matching, that is,

$$\boldsymbol{\mu} = \mathbb{E}[\mathbf{z}] = \sum_k \pi_k \boldsymbol{\mu}_k \quad (8.333)$$

$$\boldsymbol{\Sigma} = \text{Cov}[\mathbf{z}] = \sum_k \pi_k (\boldsymbol{\Sigma}_k + (\boldsymbol{\mu}_k - \boldsymbol{\mu})(\boldsymbol{\mu}_k - \boldsymbol{\mu})^\top) \quad (8.334)$$

In the graphical model literature, this is called **weak marginalization** [Lau92], since it preserves the first two moments. Applying these equations to our model, we can go from b_t^{ij} to b_t^j as follows (where we drop the t subscript for brevity):

$$\pi_j = \sum_i \pi_{ij} \quad (8.335)$$

$$\pi_{j|i} = \frac{\pi_{ij}}{\sum_{j'} \pi_{ij'}} \quad (8.336)$$

$$\boldsymbol{\mu}_j = \sum_i \pi_{j|i} \boldsymbol{\mu}_{ij} \quad (8.337)$$

$$\boldsymbol{\Sigma}_j = \sum_i \pi_{j|i} (\boldsymbol{\Sigma}_{ij} + (\boldsymbol{\mu}_{ij} - \boldsymbol{\mu}_j)(\boldsymbol{\mu}_{ij} - \boldsymbol{\mu}_j)^\top) \quad (8.338)$$

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 Gaussian (by moment matching), and then updating it using K different Kalman filters, one per value of c_t . See Figure 8.19b for a sketch.

8.8.4 ADF for training logistic regression

In this section, we discuss how to use the assumed density filtering algorithm of Section 8.8 to recursively compute (i.e., in an online fashion) the (approximate) posterior $p(\mathbf{w}_t | \mathcal{D}_{1:t})$ for a logistic 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 seen so far. This is particularly useful in cases where the data is arriving in a continual stream, such as online advertising (see e.g., [Gra+10]) and recommender systems (see e.g., [Aga+14]). We follow the presentation of [Zoe07]. (See also Section 17.6.2 where we extend this to MLPs.)

We assume our model has the following form:

$$p(y_t | \mathbf{y}_t, \mathbf{w}_t) = \text{Ber}(y_t | \sigma(\mathbf{y}_t^\top \mathbf{w}_t)) \quad (8.339)$$

$$p(\mathbf{w}_t | \mathbf{w}_{t-1}) = \mathcal{N}(\mathbf{w}_t | \mathbf{w}_{t-1}, \mathbf{Q}) \quad (8.340)$$

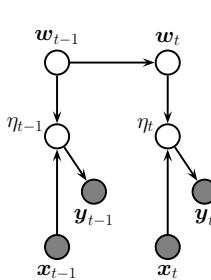


Figure 8.20: A dynamic logistic regression model. \mathbf{w}_t are the regression weights at time t , and $\eta_t = \mathbf{w}_t^\top \mathbf{y}_t$. Compare to Figure 8.9a.

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
² The posterior for η_t is given by

$$\frac{3}{4} p(\eta_t | \mathcal{D}_{1:t}) \approx p_t(\eta_t) = \mathcal{N}(\eta_t | m_t, v_t) \quad (8.345)$$

$$\frac{5}{6} m_t = \int \eta_t \frac{1}{Z_t} p(y_t | \eta_t) p_{t|t-1}(\eta_t) d\eta_t \quad (8.346)$$

$$\frac{7}{8} v_t = \int \eta_t^2 \frac{1}{Z_t} p(y_t | \eta_t) p_{t|t-1}(\eta_t) d\eta_t - m_t^2 \quad (8.347)$$

$$\frac{9}{10} Z_t = \int p(y_t | \eta_t) p_{t|t-1}(\eta_t) d\eta_t \quad (8.348)$$

¹¹
¹² where $p(y_t | \eta_t) = \text{Ber}(y_t | \eta_t)$. These integrals are one dimensional, and so can be efficiently computed
¹³ using Gaussian quadrature, as explained in [Zoe07; KB00].

¹⁴ 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
¹⁵ 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
¹⁶ smoothing posterior (leaving the current observation out), as opposed to coming from the filtered
¹⁷ posterior based just on past data. This is called **quadrature EP** [ZH05].

²⁰ Having inferred $p_t(\eta_t)$, whether using one-step EKF or EP, we need to compute $p_t(w | \eta_t)$. This
²¹ can be done as follows. Define δ_m as the change in the mean and δ_v as the change in the variance:

$$\frac{23}{24} m_t = m_{t|t-1} + \delta_m, \quad v_t = v_{t|t-1} + \delta_v \quad (8.349)$$

²⁵ Then one can show that the new factored posterior over the model parameters is given by

$$\frac{26}{27} p_t(w_{t,j}) = \mathcal{N}(w_{t,j} | \mu_{t,j}, \tau_{t,j}) \quad (8.350)$$

$$\frac{28}{29} \mu_{t,j} = \mu_{t|t-1,j} + a_j \delta_m \quad (8.351)$$

$$\frac{29}{30} \tau_{t,j} = \tau_{t|t-1,j} + a_j^2 \delta_v \quad (8.352)$$

$$\frac{30}{31} a_j \triangleq \frac{x_{t,j} \tau_{t|t-1,j}}{\sum_{j'} x_{t,j'}^2 \tau_{t|t-1,j}^2} \quad (8.353)$$

³³ Thus we see that the parameters which correspond to inputs with larger magnitude (big $|x_{t,j}|$) or
³⁴ larger uncertainty (big $\tau_{t|t-1,j}$) get updated most, which makes intuitive sense.

³⁵ As an example, consider again the 2d binary classification problem in Section 8.5.4.3. We
³⁶ sequentially compute the posterior using the ADF, and compare to the offline estimate computed
³⁷ using a Laplace approximation (where the MAP estimate is computed using BFGS) and an MCMC
³⁸ approximation, which we take as “ground truth”. In Figure 8.21, we see that the resulting posterior
³⁹ predictive distributions are similar. Finally, in Figure 8.22, we visualize how the posterior marginals
⁴⁰ converge over time.

⁴¹ Note that the whole algorithm only takes $O(D)$ time and space per step, the same as SGD. However,
⁴² unlike SGD, there are no step-size parameters, since the diagonal covariance implicitly specifies the
⁴³ size of the update for each dimension. Furthermore, we get a posterior approximation, not just a
⁴⁴ point estimate. And since it is an online algorithm, it can also handle massive datasets. [ZGH10]
⁴⁵ extend this approach to the multi-label setting, and use it for online ranking problems.

⁴⁶
⁴⁷

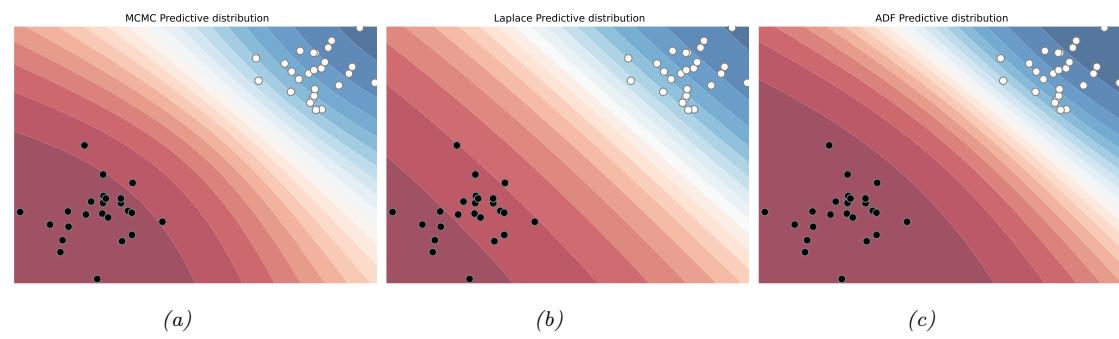


Figure 8.21: Bayesian inference applied to a 2d binary logistic regression problem, $p(y = 1|\mathbf{x}) = \sigma(w_0 + w_1x_1 + w_2x_2)$. We show the training data and the posterior predictive produced by different methods. (a) Offline MCMC approximation. (b) Offline Laplace approximation. (c) Online ADF approximation at the final step of inference. Generated by [adf_logistic_regression_demo.py](#).

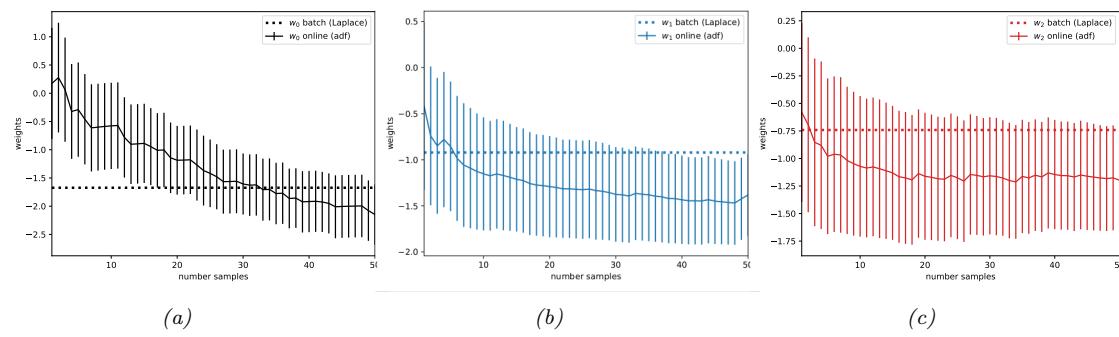


Figure 8.22: Marginal posteriors over time for the ADF method. The horizontal line is the offline MAP estimate. Generated by `adf_logistic_regression_demo.py`.

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

⁸ 9.2 Belief propagation on trees

⁹ The forwards-backwards algorithm for HMMs (Section 8.3.3) and the Kalman smoother algorithm
¹⁰ for LDS (Section 8.4.4) can both be interpreted as **message passing** algorithms, that pass messages
¹¹ along edges in the graph, in order to compute posterior marginals at each node, as illustrated in
¹² 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
¹³ algorithms are also called **belief propagation (BP)** algorithms.

¹⁴ To implement such methods, we have to specify how to multiply messages together to compute a joint
¹⁵ distribution (product operation), and how to marginalize out some variables from a joint distribution
¹⁶ (sum operation). For discrete distributions, this just requires manipulating multidimensional tables
¹⁷ (tensors). For Gaussians, we can use the rules defined in Section 2.3.7. However, we can also generalize
¹⁸ these operations to any commutative semi-ring, as we explain in Section 9.5.3.

¹⁹ In addition to specifying the local update rules, we need to specify a **message passing schedule**
²⁰ such that every node gets to “see” information from the entire rest of the graph exactly once (to
²¹ avoid overcounting of evidence). For chains the obvious schedule is left-to-right and then right-to-left,
²² as we illustrated above. For trees, we can go up to the root and then back down to the leaves, as we
²³ discuss in Section 9.2.1. General graphs may have cycles or loops; we discuss this case in Section 9.3.
²⁴

²⁵ 9.2.1 BP for polytrees

²⁶ In this section, we generalize the forwards-backwards algorithm for chains to work on a **polytree**,
²⁷ which is a directed graph whose undirected “backbone” is a tree, i.e., a graph with no loops. (That is,
²⁸ a polytree is a directed tree with multiple root nodes, in which a node may have multiple parents,
²⁹ whereas in a singly rooted tree, each node has a single parent.) This algorithm is called **belief**
³⁰ **propagation** and is due to [Pea88].

³¹ We consider the case of a general discrete node X with parents U_i and children Y_j . We partition
³² the evidence in the graph, e , into the evidence upstream of node X , e_X^+ , and all the rest, e_X^- . Thus
³³ e_X^+ contains all the evidence separated from X if its incoming arcs were deleted, and e_X^- contains the
³⁴ evidence below X and the evidence in X itself, if any. The posterior on node X can be computed as
³⁵ follows.

$$\text{bel}_X(x) \triangleq \Pr(X = x|e) = c' \lambda_X(x) \pi_X(x) \quad (9.3)$$

$$\lambda_X(x) \triangleq P(e_X^-|X = x) \quad (9.4)$$

$$\pi_X(x) \triangleq \Pr(X = x|e_X^+) \quad (9.5)$$

³⁶ where c' is a normalizing constant.

³⁷ Consider the graph shown in Figure 9.1. We will use the notation $e_{U_1 \rightarrow X}^+$ to denote the evidence
³⁸ 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
³⁹

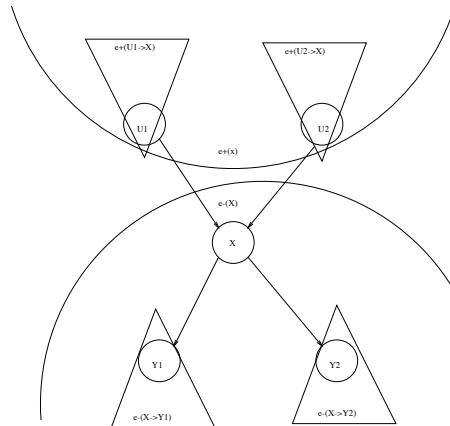


Figure 9.1: Message passing in a polytree.

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 .

$$\Pr(X = x|e_X^+) = \sum_{u1, u2} \Pr(X = x, U1 = u1, U2 = u2|e_X^+) \quad (9.9)$$

$$= \sum_{u1, u2} \Pr(X = x|u1, u2) \Pr(u1, u2|e_{U1 \rightarrow X}^+, e_{U2 \rightarrow X}^+) \quad (9.10)$$

$$= \sum_{u1, u2} \Pr(X = x|u1, u2) \Pr(u1|e_{U1 \rightarrow X}^+) \Pr(u2|e_{U2 \rightarrow X}^+) \quad (9.11)$$

10 If we define the π “message” that a node X sends to its children Y_j as

$$\Pi_{X \rightarrow Y_j}(x) \triangleq \Pr(X = x|e_{X \rightarrow Y_j}^+) \quad (9.12)$$

14 we can write in general that

$$\pi_X(x) = \sum_u P(X = x|u) \prod_i \Pi_{U_i \rightarrow X}(u_i) \quad (9.13)$$

18 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.

$$\lambda_{X \rightarrow U1}(u1) = \Pr(e_X^-, e_{U2 \rightarrow X}^+|u1) \quad (9.14)$$

24 all the ev. except in the U1 triangle

$$= \sum_x \sum_{u2} \Pr(e_X^-, e_{U2 \rightarrow X}^+|u1, u2, x) \Pr(u2, x|u1) \quad (9.16)$$

$$= \sum_x \sum_{u2} \Pr(e_X^-|x) \Pr(e_{U2 \rightarrow X}^+|u2) \Pr(u2, x|u1) \quad (9.17)$$

30 since X separates the U2 triangle from e_X^- , and $U2$ separates the U2 triangle from U1
31

$$= c \sum_x \sum_{u2} \Pr(e_X^-|x) \frac{\Pr(u2|e_{U2 \rightarrow X}^+)}{\Pr(u2)} \Pr(x|u2, u1) \Pr(u2|u1) \quad (9.19)$$

35 using Bayes’ rule, where $c = \Pr(e_{U2 \rightarrow X}^+)$ is a constant

$$= c \sum_x \sum_{u2} \Pr(e_X^-|x) \Pr(u2|e_{U2 \rightarrow X}^+) \Pr(x|u2, u1) \quad (9.21)$$

38 since U1 and U2 are marginally independent

$$= c \sum_x \sum_{u2} \lambda_X(x) \Pi_{U2 \rightarrow X}(u2) \Pr(x|u2, u1) \quad (9.23)$$

42 In general, we have

$$\lambda_{X \rightarrow U_i}(u_i) = c \sum_x \lambda_X(x) \left[\sum_{u_k: k \neq i} P(X = x|u) \prod_{k \neq i} \Pi_{U_k \rightarrow X}(u_k) \right] \quad (9.24)$$

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
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

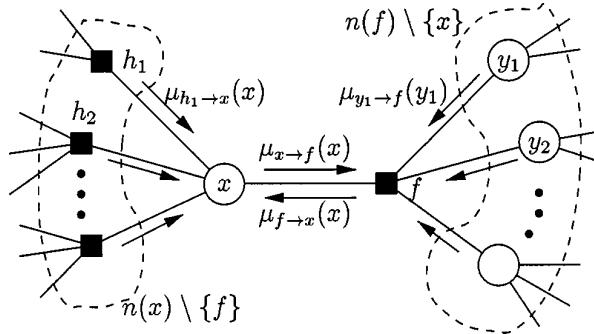


Figure 9.2: Message passing on a bipartite factor graph. Square nodes represent factors, and circles represent variables. The y_i nodes correspond to the neighbors x'_i of f other than x . From Figure 6 of [KFL01]. Used with kind permission of Brendan Frey.

More precisely, we initialize all messages to the all 1's vector. Then, in parallel, each node absorbs messages from all its neighbors using

$$\text{bel}_s(x_s) \propto \psi_s(x_s) \prod_{t \in \text{nbr}_s} m_{t \rightarrow s}(x_s) \quad (9.29)$$

Then, in parallel, each node sends messages to each of its neighbors:

$$m_{s \rightarrow t}(x_t) = \sum_{x_s} \left(\psi_s(x_s) \psi_{st}(x_s, x_t) \prod_{u \in \text{nbr}_s \setminus t} m_{u \rightarrow s}(x_s) \right) \quad (9.30)$$

The $m_{s \rightarrow t}$ message is computed by multiplying together all incoming messages, except the one sent by the recipient, and then passing through the ψ_{st} potential. (Note that we no longer need to distinguish λ messages from π messages, since all messages are just a product of factors that will be normalized at the end to get a proper probability.)

We continue this process until convergence. If the graph is a tree, the method is guaranteed to converge after $D(G)$ iterations, where $D(G)$ is the **diameter** of the graph, that is, the largest distance between any two nodes. But for loopy graphs, the method may not converge at all, and even if it does, it is not clear if the resulting belief states are accurate. We discuss these issues below.

9.2.3 BP for factor graphs

To handle models with higher-order clique potentials (beyond pairwise), it is useful to use a factor graph representation described in Section 4.4.5. In this section, we describe the BP equations for bipartite factor graphs, as proposed in [KFL01].¹ For a version that works for Forney factor graphs, see [Loe+07].

1. For an efficient JAX implementation of BP for factor graphs (including graphs with loops, as discussed in Section 9.3), see <https://github.com/vicariousinc/PGMax>.

1 In the case of bipartite factor graphs, we have two kinds of messages: variables to factors
2

$$\underline{4} \quad m_{x \rightarrow f}(x) = \prod_{h \in \text{nbr}(x) \setminus \{f\}} m_{h \rightarrow x}(x) \quad (9.31)$$

5 and factors to variables:
6

$$\underline{8} \quad m_{f \rightarrow x}(x) = \sum_{\mathbf{x}'} f(x, \mathbf{x}') \prod_{x' \in \text{nbr}(f) \setminus \{x\}} m_{x' \rightarrow f}(x') \quad (9.32)$$

11 Here $\text{nbr}(x)$ are all the factors that are connected to variable x , and $\text{nbr}(f)$ are all the variables that
12 are connected to factor f . These messages are illustrated in Figure 9.2. At convergence, we can
13 compute the final beliefs as a product of incoming messages:
14

$$\underline{15} \quad \text{bel}(x) \propto \prod_{f \in \text{nbr}(x)} m_{f \rightarrow x}(x) \quad (9.33)$$

18 9.2.4 Max product belief propagation

20 So far we have considered **sum-product belief propagation**. We can replace the sum operation
21 with the max operation to get **max-product belief propagation**. The result of this computation
22 (whether serial or parallel) is that we compute the **max marginals** for each node:
23

$$\underline{24} \quad \zeta_i(k) = \max_{\mathbf{x}_{-i}} p(x_i = k, \mathbf{x}_{-i} | \mathbf{v}) \quad (9.34)$$

26 (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
27 the same result.) By contrast, sum-product computes the posterior marginals:
28

$$\underline{29} \quad \gamma_i(k) = \sum_{\mathbf{x}_{-i}} p(x_i = k, \mathbf{x}_{-i} | \mathbf{v}) \quad (9.35)$$

32 We can derive two different kinds of “MAP” estimates from these local quantities. Let $\hat{x}_i = \arg\max_k \gamma_i(k)$; this is known as the **maximizer of the posterior marginal** or **MPM** estimate
33 (see e.g., [MMP87; SM12]); let $\hat{\mathbf{x}} = [\hat{x}_1, \dots, \hat{x}_V]$ be the sequence of such estimates.
34

35 Now consider $\tilde{x}_i = \arg\max_k \zeta_i(k)$; we call this the **maximizer of the max marginal** or **MMM**
36 estimate; let $\tilde{\mathbf{x}} = [\tilde{x}_1, \dots, \tilde{x}_V]$.

37 An interesting question is: what, if anything, do these estimates have to do with the “true” MAP
38 estimate, $\mathbf{x}^* = \arg\max_{\mathbf{x}} p(\mathbf{x} | \mathbf{v})$? We discuss this below.
39

40 9.2.4.1 Connection between MMM and MAP

42 In [YW04], they showed that, if the max marginals are unique and computed exactly (e.g., if the
43 graph is a tree), then $\tilde{\mathbf{x}} = \mathbf{x}^*$. This means we can recover the global MAP estimate by running max
44 product BP and then setting each node to its local max (i.e., using the MMM estimate).

45 However, if there are ties in the max marginals (corresponding to the case where there is more
46 than one globally optimal solution), this “local stitching” process may result in global inconsistencies.
47

¹ If we have a tree-structured model, we can use a **traceback** procedure, analogous to the Viterbi
² algorithm (Section 8.3.6), in which we clamp nodes to their optimal values while working backwards
³ from the root. For details, see e.g., [KF09a, p569].

⁴

⁵ Unfortunately, traceback does not work on general graphs. An alternative, iterative approach,
⁶ proposed in [YW04], is follows. First we run max product BP, and clamp all nodes which have unique
⁷ max marginals to their optimal values; we then clamp a single ambiguous node to an optimal value,
⁸ and condition on all these clamped values as extra evidence, and perform more rounds of message
⁹ passing, until all ties are broken. This may require many rounds of inference, although the number
¹⁰ of non-clamped (hidden) variables get reduced at each round.

¹¹

¹² 9.2.4.2 Connection between MPM and MAP

¹³

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].

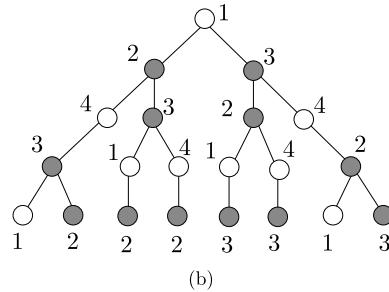
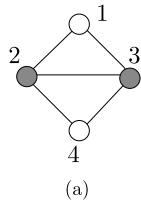
Another approach to is to use sampling methods to approximate the relevant integrals (cf. particle filtering in Section 13.2); this is called **non-parametric BP** or **particle BP** (see e.g., [Sud+03; Isa03; Sud+10; Pac+14]).

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]

1
2
3
4
5
6
7
8
9
10



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

24 Loopy BP may not converge, or may only converge slowly. In this section, we discuss some techniques
 25 that increase the chances of convergence, and the speed of convergence.

26

27 9.3.1.1 When will LBP converge?

28

29 The details of the analysis of when LBP will converge are beyond the scope of this chapter, but
 30 we briefly sketch the basic idea. The key analysis tool is the **computation tree**, which visualizes
 31 the messages that are passed as the algorithm proceeds. Figure 9.3 gives a simple example. In the
 32 first iteration, node 1 receives messages from nodes 2 and 3. In the second iteration, it receives one
 33 message from node 3 (via node 2), one from node 2 (via node 3), and two messages from node 4 (via
 34 nodes 2 and 3). And so on.

35 The key insight is that T iterations of LBP is equivalent to exact computation in a computation
 36 tree of height $T + 1$. If the strengths of the connections on the edges is sufficiently weak, then the
 37 influence of the leaves on the root will diminish over time, and convergence will occur. See [MK05;
 38 WJ08] and references therein for more information.

39

40 9.3.1.2 Making LBP converge

41 Although the theoretical convergence analysis is very interesting, in practice, when faced with a
 42 model where LBP is not converging, what should we do?

43 One simple way to increase the chance of convergence is to use **damping**. That is, at iteration k ,
 44 we use an update of the form

$$45 \quad m_{t \rightarrow s}^k(x_s) = \lambda m_{t \rightarrow s}(x_s) + (1 - \lambda) m_{t \rightarrow s}^{k-1}(x_s) \quad (9.37)$$

46

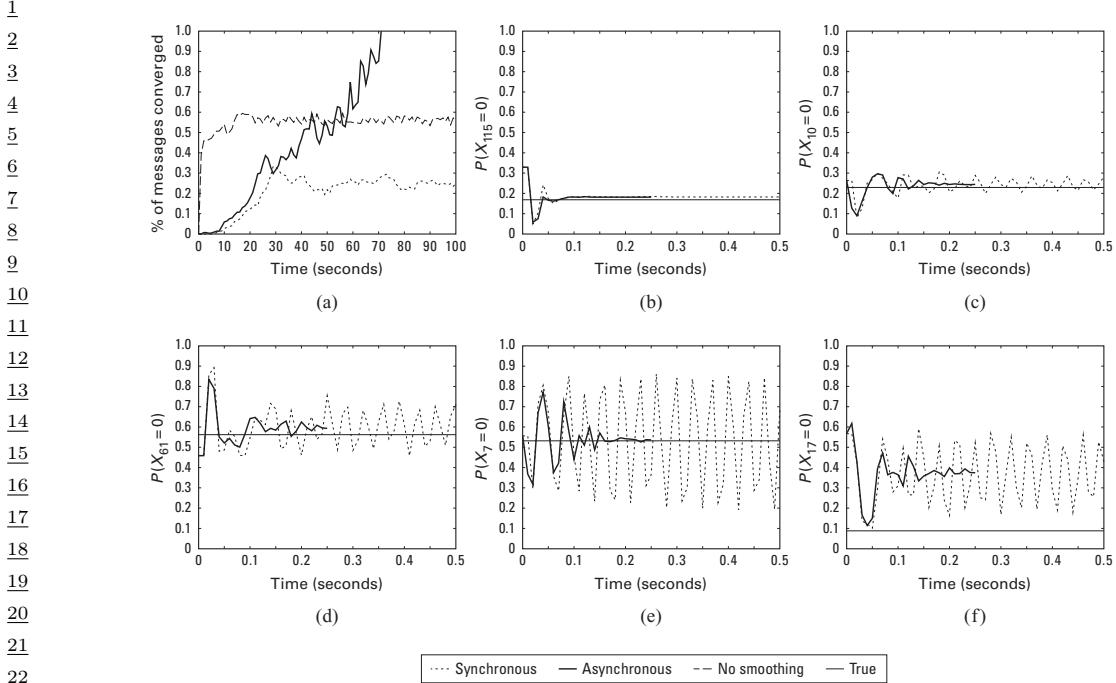


Figure 9.4: Illustration of the behavior of loopy belief propagation on an 11×11 Ising grid with random potentials, $w_{ij} \sim \text{Unif}(-C, C)$, where $C = 11$. For larger C , inference becomes harder. (a) Percentage of messages that have converged vs time for 3 different update schedules: Dotted = 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.

¹
² **9.3.1.3 Increasing the convergence rate with adaptive scheduling**
³ The standard approach when implementing LBP is to perform **synchronous updates**, where all
⁴ nodes absorb messages in parallel, and then send out messages in parallel. That is, the new messages
⁵ at iteration $k + 1$ are computed in parallel using
⁶

$$\underline{7} \quad \mathbf{m}^{k+1} = (f_1(\mathbf{m}^k), \dots, f_E(\mathbf{m}^k)) \quad (9.38)$$

⁸
⁹ where E is the number of edges, and $f_{st}(\mathbf{m})$ is the function that computes the message for edge
¹⁰ $s \rightarrow t$ given all the old messages. This is analogous to the Jacobi method for solving linear systems
¹¹ of equations.

¹² It is well known [Ber97] that the Gauss-Seidel method, which performs **asynchronous updates**
¹³ in a fixed round-robin fashion, converges faster when solving linear systems of equations. We can
¹⁴ apply the same idea to LBP, using updates of the form

$$\underline{15} \quad \mathbf{m}_i^{k+1} = f_i(\{\mathbf{m}_j^{k+1} : j < i\}, \{\mathbf{m}_j^k : j > i\}) \quad (9.39)$$

¹⁶
¹⁷ where the message for edge i is computed using new messages (iteration $k + 1$) from edges earlier in
¹⁸ the ordering, and using old messages (iteration k) from edges later in the ordering.

¹⁹ This raises the question of what order to update the messages in. One simple idea is to use a fixed
²⁰ or random order. The benefits of this approach are shown in Figure 9.4, where we see that (damped)
²¹ asynchronous updating results in convergence much more often than synchronous updating.

²² However, we can do even better by using an adaptive ordering. The intuition is that we should
²³ focus our computational efforts on those variables that are most uncertain. [EMK06] proposed
²⁴ a technique known as **residual belief propagation**, in which messages are scheduled to be sent
²⁵ according to the norm of the difference from their previous value. That is, we define the residual of
²⁶ new message $m_{s \rightarrow t}$ at iteration k to be

$$\underline{27} \quad r(s, t, k) = \|\log m_{s \rightarrow t} - \log m_{s \rightarrow t}^k\|_\infty = \max_i |\log \frac{m_{s \rightarrow t}(i)}{m_{s \rightarrow t}^k(i)}| \quad (9.40)$$

²⁸
²⁹ We can store messages in a priority queue, and always send the one with highest residual. When a
³⁰ message is sent from s to t , all of the other messages that depend on $m_{s \rightarrow t}$ (i.e., messages of the
³¹ form $m_{t \rightarrow u}$ where $u \in \text{nbr}(t) \setminus s$) need to be recomputed; their residual is recomputed, and they are
³² added back to the queue. In [EMK06], they showed (experimentally) that this method converges
³³ more often, and much faster, than using synchronous updating, asynchronous updating with a fixed
³⁴ order, and the TRP approach.

³⁵
³⁶ A refinement of residual BP was presented in [SM07]. In this paper, they use an upper bound on
³⁷ the residual of a message instead of the actual residual. This means that messages are only computed
³⁸ if they are going to be sent; they are not just computed for the purposes of evaluating the residual.
³⁹ This was observed to be about five times faster than residual BP, although the quality of the final
⁴⁰ results is similar.

⁴¹
⁴² **9.3.2 Accuracy**
⁴³
⁴⁴ For a graph with a single loop, one can show that the max-product version of LBP will find the
⁴⁵ correct MAP estimate, if it converges [Wei00]. For more general graphs, one can bound the error in
⁴⁶ the approximate marginals computed by LBP, as shown in [WJW03; IFW05; Vin+10].
⁴⁷

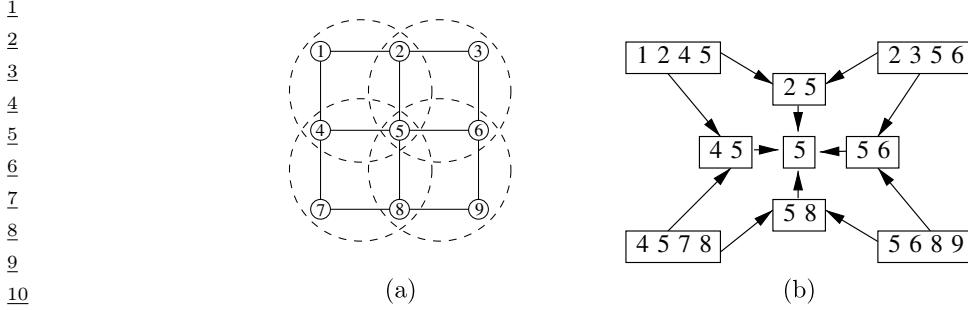


Figure 9.5: (a) Clusters superimposed on a 3×3 lattice graph. (b) Corresponding hyper-graph. Nodes represent clusters, and edges represent set containment. From Figure 4.5 of [WJ08]. Used with kind permission of Martin Wainwright.

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.

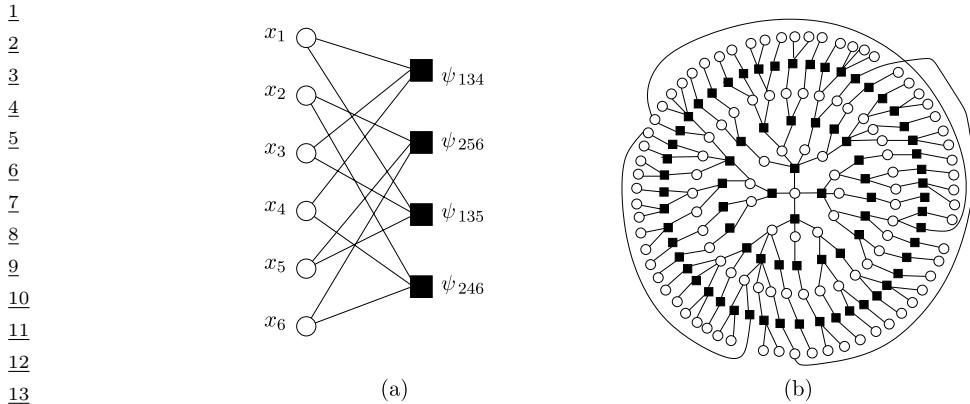


Figure 9.6: (a) A simple factor graph representation of a $(2,3)$ low-density parity check code. Each message bit (hollow round circle) is connected to two parity factors (solid black squares), and each parity factor is connected to three bits. Each parity factor has the form $\psi_{stu}(x_s, x_t, x_u) = \mathbb{I}(x_s \otimes x_t \otimes x_u = 1)$, 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.

²⁴ However, the main impetus behind the interest in LBP arose when McEliece, MacKay, and Cheng
²⁵ [MMC98] showed that a popular algorithm for error correcting codes, known as **turbocodes** [BGT93]
²⁶, 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

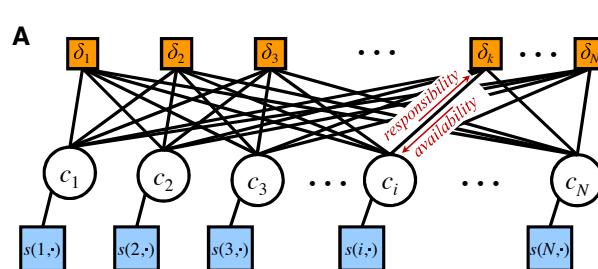


Figure 9.7: Factor graphs for affinity propagation. Circles are variables, squares are factors. Each c_i node has N possible states. From Figure S2 of [FD07a]. Used with kind permission of Brendan Frey.

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,

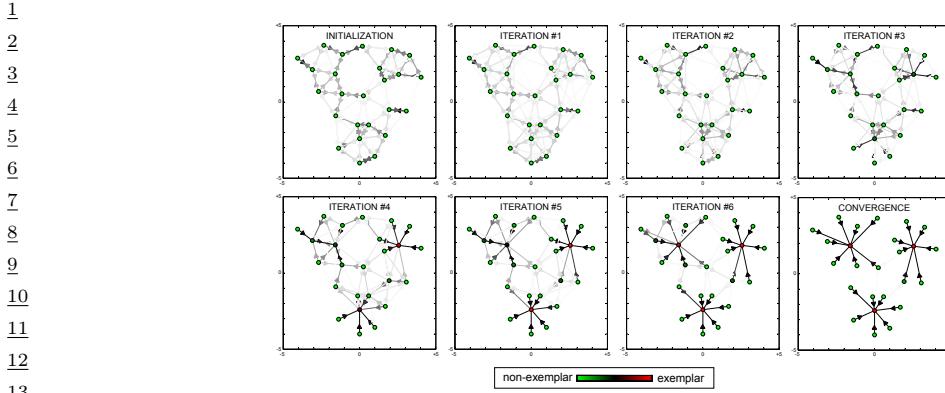


Figure 9.8: Example of affinity propagation. Each point is colored coded by how much it wants to be an exemplar (red is the most, green is the least). This can be computed by summing up all the incoming availability messages and the self-similarity term. The darkness of the $i \rightarrow k$ arrow reflects how much point i wants to belong to exemplar k . From Figure 1 of [FD07a]. Used with kind permission of Brendan Frey.

18

19

20

21 denoted $r_{i \rightarrow k}$, known as the responsibility. This is a measure of how much i thinks k would make a
22 good exemplar, compared to all the other exemplars i has looked at. In addition, each factor node
23 δ_k sends a message to each variable node c_i . Again this can be reduced to a scalar message, $a_{i \leftarrow k}$,
24 known as the availability. This is a measure of how strongly k believes it should be an exemplar for i ,
25 based on all the other data points k has looked at.

26 As usual with loopy BP, the method might oscillate, and convergence is not guaranteed. However,
27 by using damping, the method is very reliable in practice. If the graph is densely connected, message
28 passing takes $O(N^2)$ time, but with sparse similarity matrices, it only takes $O(E)$ time, where E is
29 the number of edges or non-zero entries in \mathbf{S} .

30 The number of clusters can be controlled by scaling the diagonal terms $S(i, i)$, which reflect how
31 much each data point wants to be an exemplar. Figure 9.8 gives a simple example of some 2d data,
32 where the negative Euclidean distance was used to measured similarity. The $S(i, i)$ values were set
33 to be the median of all the pairwise similarities. The result is 3 clusters. Many other results are
34 reported in [FD07a], who show that the method significantly outperforms K-medoids.

35

36 9.3.7 Emulating BP with graph neural nets

37

38 There is a close connection between message passing in PGMs and message passing in graph neural
39 networks (GNNs), which we discuss in Section 16.3.5. However, for PGMs, the message computations
40 are computing using (non-learned) update equations that work for any model; all that is needed
41 is the graph structure G , model parameters θ , and evidence v . By contrast, GNNs are trained to
42 emulate specific functions using labeled input-output pairs.

43 It is natural to wonder what happens if we train a GNN on the exact posterior marginals derived
44 from a small PGM, and then apply that trained GNN to a different test PGM. In [Yoo+18; Zha+19c],
45 they show this method can work quite well if the test PGM is similar in structure to the one used for
46 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

$$\begin{aligned} p(J) &= \sum_{L,S,G,H,I,D,C} p(C, D, I, G, S, L, J, H) \\ &= \sum_{L,S,G,H,I,D,C} \psi_C(C) \psi_D(D, C) \psi_I(I) \psi_G(G, I, D) \psi_S(S, I) \psi_L(L, G) \\ &\quad \times \psi_J(J, L, S) \psi_H(H, G, J) \\ &= \sum_{L,S} \psi_J(J, L, S) \sum_G \psi_L(L, G) \sum_H \psi_H(H, G, J) \sum_I \psi_S(S, I) \psi_I(I) \\ &\quad \times \sum_D \psi_G(G, I, D) \sum_C \psi_C(C) \psi_D(D, C) \end{aligned}$$

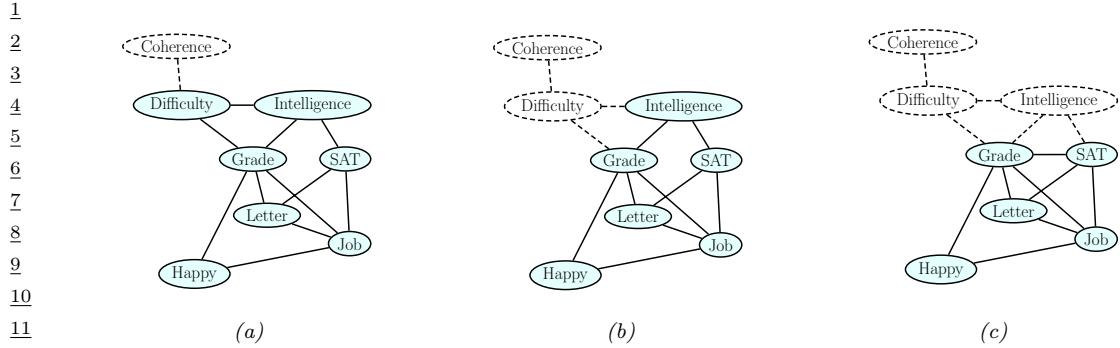


Figure 9.9: Example of the elimination process, in the order C, D, I, H, G, S, L . When we eliminate I (figure 14 c), we add a fill-in edge between G and S , since they are not connected. Adapted from Figure 9.10 of [KF09a].

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

$$\tau_1(D) = \sum_C \tau'_1(C, D) \quad (9.46)$$

Next we multiply together all the terms in the scope of the \sum_D operator and then marginalize out to create

$$\tau'_2(G, I, D) = \psi_G(G, I, D)\tau_1(D) \quad (9.47)$$

$$\tau_2(G, I) = \sum_D \tau'_2(G, I, D) \quad (9.48)$$

And so on.

The above technique can be used to compute any marginal of interest, such as $p(J)$ or $p(J, H)$. To compute a conditional, we can take a ratio of two marginals, where the visible variables have been clamped to their known values (and hence don't need to be summed over). For example,

$$p(J = j | I = 1, H = 0) = \frac{p(J = j, I = 1, H = 0)}{\sum_{j'} p(J = j', I = 1, H = 0)} \quad (9.49)$$

9.4.2 Computational complexity of VE

The running time of VE is clearly exponential in the size of the largest factor, since we have to sum over all of the corresponding variables. Some of the factors come from the original model (and are thus unavoidable), but new factors may also be created in the process of summing out. For example, in Table 9.1, we created a factor involving G, I and S ; but these nodes were not originally present together in any factor.

$$\begin{aligned}
 & \sum_L \sum_S \psi_J(J, L, S) \sum_G \psi_L(L, G) \sum_H \psi_H(H, G, J) \sum_I \psi_S(S, I) \psi_I(I) \sum_D \psi_G(G, I, D) \underbrace{\sum_C \psi_C(C) \psi_D(D, C)}_{\tau_1(D)} \\
 & \sum_L \sum_S \psi_J(J, L, S) \sum_G \psi_L(L, G) \sum_H \psi_H(H, G, J) \sum_I \psi_S(S, I) \psi_I(I) \underbrace{\sum_D \psi_G(G, I, D) \tau_1(D)}_{\tau_2(G, I)} \\
 & \sum_L \sum_S \psi_J(J, L, S) \sum_G \psi_L(L, G) \sum_H \psi_H(H, G, J) \underbrace{\sum_I \psi_S(S, I) \psi_I(I) \tau_2(G, I)}_{\tau_3(G, S)} \\
 & \sum_L \sum_S \psi_J(J, L, S) \sum_G \psi_L(L, G) \sum_H \psi_H(H, G, J) \underbrace{\tau_3(G, S)}_{\tau_4(G, J)} \\
 & \sum_L \sum_S \psi_J(J, L, S) \underbrace{\sum_G \psi_L(L, G) \tau_4(G, J) \tau_3(G, S)}_{\tau_5(J, L, S)} \\
 & \sum_L \sum_S \psi_J(J, L, S) \underbrace{\tau_5(J, L, S)}_{\tau_6(J, L)} \\
 & \sum_L \tau_6(J, L) \\
 & \underbrace{\tau_6(J, L)}_{\tau_7(J)}
 \end{aligned}$$

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

$$\tau'_3(G, S, I) = \psi_S(S, I) \psi_I(I) \tau_2(G, I) \quad (9.50)$$

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:

$$\{C, D\}, \{D, I, G\}, \{G, L, S, J\}, \{G, J, H\}, \{G, I, S\} \quad (9.51)$$

1 $\sum_D \sum_C \psi_D(D, C) \sum_H \sum_L \sum_S \psi_J(J, L, S) \sum_I \underbrace{\psi_I(I) \psi_S(S, I)}_{\tau_1(I, D, L, J, H)} \sum_G \psi_G(G, I, D) \psi_L(L, G) \psi_H(H, G, J)$
 2
 3
 4
 5 $\sum_D \sum_C \psi_D(D, C) \sum_H \sum_L \sum_S \psi_J(J, L, S) \underbrace{\sum_I \psi_I(I) \psi_S(S, I) \tau_1(I, D, L, J, H)}_{\tau_2(D, L, S, J, H)}$
 6
 7
 8 $\sum_D \sum_C \psi_D(D, C) \sum_H \sum_L \underbrace{\sum_S \psi_J(J, L, S) \tau_2(D, L, S, J, H)}_{\tau_3(D, L, J, H)}$
 9
 10
 11
 12 $\sum_D \sum_C \psi_D(D, C) \sum_H \underbrace{\sum_L \tau_3(D, L, J, H)}_{\tau_4(D, J, H)}$
 13
 14
 15 $\sum_D \sum_C \psi_D(D, C) \underbrace{\sum_H \tau_4(D, J, H)}_{\tau_5(D, J)}$
 16
 17
 18 $\sum_D \underbrace{\sum_C \psi_D(D, C) \tau_5(D, J)}_{\tau_6(D, J)}$
 19
 20 $\underbrace{\sum_D \tau_6(D, J)}_{\tau_7(J)}$
 21
 22
 23

Table 9.2: Eliminating variables from Figure 4.28 in the order G, I, S, L, H, C, D .

24 It is clear that the time complexity of VE is
 25
 26
 27

28 It is clear that the time complexity of VE is
 29

$$30 \quad \sum_{c \in \mathcal{C}(G_\prec)} K^{|c|} \tag{9.52}$$

31
 32 where $\mathcal{C}(G)$ are the (maximal) cliques in graph G , $|c|$ is the size of the clique c , and we assume for
 33 notational simplicity that all the variables have K states each.

34 Let us define the **induced width** of a graph given elimination ordering \prec , denoted w_\prec , as the
 35 size of the largest factor (i.e., the largest clique in the induced graph) minus 1. Then it is easy to
 36 see that the complexity of VE with ordering \prec is $O(K^{w_\prec+1})$. The smallest possible induced width
 37 for a graph is known at its **treewidth** (see Section 9.4.2). Unfortunately finding the corresponding
 38 optimal elimination order is an NP-complete problem [Yan81; ACP87]. See Section 9.5.1.4 for a
 39 discussion of some approximate methods for finding good elimination orders.
 40

41

42 9.4.3 Computational complexity of exact inference

43

44 We have seen that variable elimination takes $O(NK^{w+1})$ time to compute the marginals for a graph
 45 with N nodes, and treewidth w , where each variable has K states. If the graph is densely connected,
 46 then $w = O(N)$, and so inference will take time exponential in N .
 47

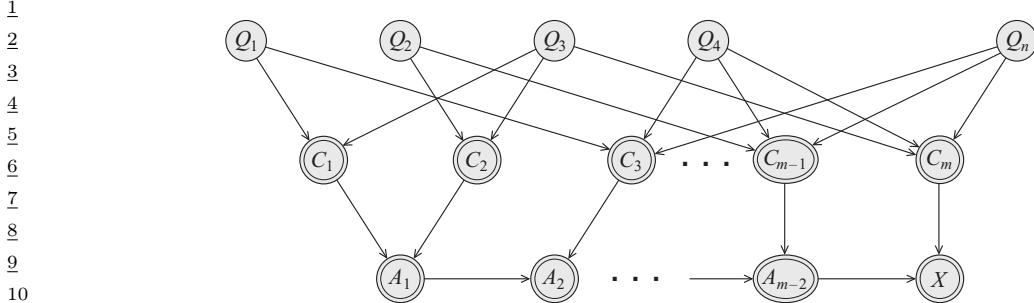


Figure 9.10: Encoding a 3-SAT problem on n variables and m clauses as a DGM. The Q_s variables are binary random variables. The C_t variables are deterministic functions of the Q_s 's, and compute the truth value of each clause. The A_t nodes are a chain of AND gates, to ensure that the CPT for the final 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

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

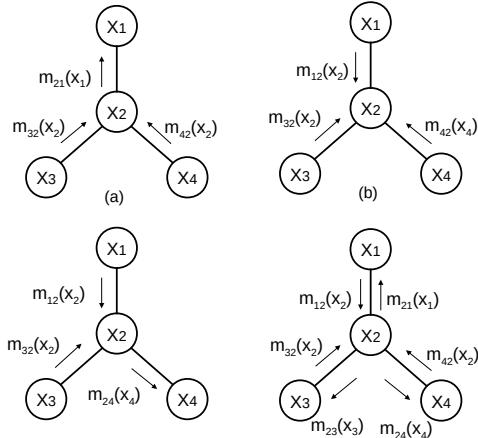


Figure 9.11: Sending multiple messages along a tree. (a) X_1 is root. (b) X_2 is root. (c) X_4 is root. (d) All of the messages needed to compute all singleton marginals. Adapted from Figure 4.3 of [Jor07].

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.

34

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].

42

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

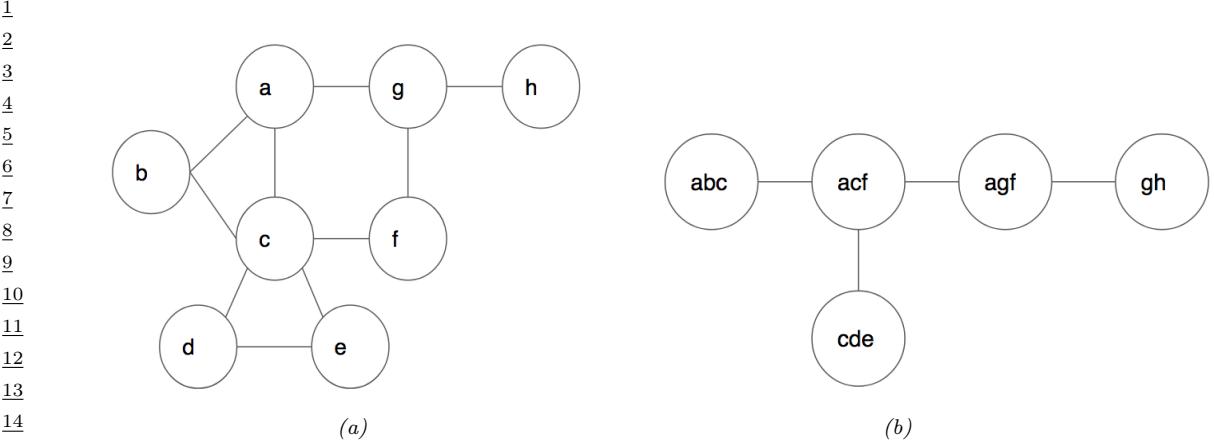


Figure 9.12: (a) An undirected graph. (b) Its corresponding junction tree.

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.

1 There are many possible tree compositions of a graph, as we discuss below. We therefore define
 2 the **treewidth** of a graph G as the minimum width of any tree decomposition for G minus 1:
 3

$$4 \quad \text{treewidth}(G) \triangleq \left(\min_{T \in \mathcal{T}(G)} \text{width}(T) \right) - 1 \quad (9.54)$$

5 We see that the treewidth of a tree is 1, and the treewidth of the graph in Figure 9.12(a) is 2.
 6

7 9.5.1.2 Why create a tree decomposition?

8 Before we discuss how to compute a tree decomposition, we pause and explain why we want to do
 9 this. The reason is that trees have a number of properties that make them useful for computational
 10 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
 11 the path from u to v that is a **separator**, i.e., that partitions the graph into two subgraphs, one
 12 containing u and the other containing v . This is conducive to using algorithms based on dynamic
 13 programming, where we recursively solve the subproblems defined on the two subtrees, and then
 14 combine their solutions via the separator node s . This is useful for graphical model inference (see
 15 Section 9.5), solving sparse systems of linear equations (see e.g., [PL03]), etc.
 16

17 9.5.1.3 Computing a tree decomposition

18 We now describe an algorithm known as **triangulation** or **elimination** for constructing a junction
 19 tree from an undirected graph. We first choose an ordering of the nodes, π . We then work backwards
 20 in this ordering, eliminating the nodes one at a time. We initially let $\mathcal{U} = \{1, \dots, N\}$ be the set
 21 of all uneliminated nodes, and set the counter to $i = N$. At each step i , we pick node $v_i = \pi_i$, we
 22 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
 23 between all nodes in C_i to make it a clique, we eliminate v_i by removing it from \mathcal{U} , and we decrement
 24 i by 1, until all nodes are eliminated.

25 We illustrate this method by applying it to the graph in Figure 9.13, using the ordering $\pi =$
 26 (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
 27 Figure 9.13(a). We create the set $C_8 = \{g, h\}$ from node v_i and all its uneliminated neighbors.
 28 Then we add fill-in edges between them, if necessary. (In this case all the nodes in C_8 are already
 29 connected.) In the next step, we eliminate $v_i = \pi(7) = g$, and create the clique $C_7 = \{a, f, g\}$,
 30 adding the fill-in edge $a - f$, as shown in Figure 9.13(b). We continue in this way until all nodes are
 31 eliminated, as shown in Figure 9.13(c).

32 If we add the fill-in edges back to the original graph, the resulting graph will be **chordal**, which
 33 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
 34 in a chordal graph is length 3. Consequently chordal graphs are sometimes called **triangulated**.

35 Figure 9.13(d) illustrates the maximal cliques of the resulting chordal graph. In general, computing
 36 the maximal cliques of a graph is NP-hard, but in the case of a chordal graph, the process is easy:
 37 at step i of the elimination algorithm, we create clique C_i by connecting v_i to all its uneliminated
 38 neighbors; if this clique is contained in an already created clique, we simply discard it, otherwise we
 39 add it to our list of cliques. For example, when triangulating the graph in Figure 9.13, we drop clique
 40 $C_4 = \{c, d\}$ since it is already contained in $C_5 = \{c, d, e\}$. Similarly we drop cliques $C_2 = \{a, b\}$ and
 41 $C_1 = \{a\}$.

42 There are several ways to create a jtree from this set of cliques. One approach is as follows: create
 43 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
 44

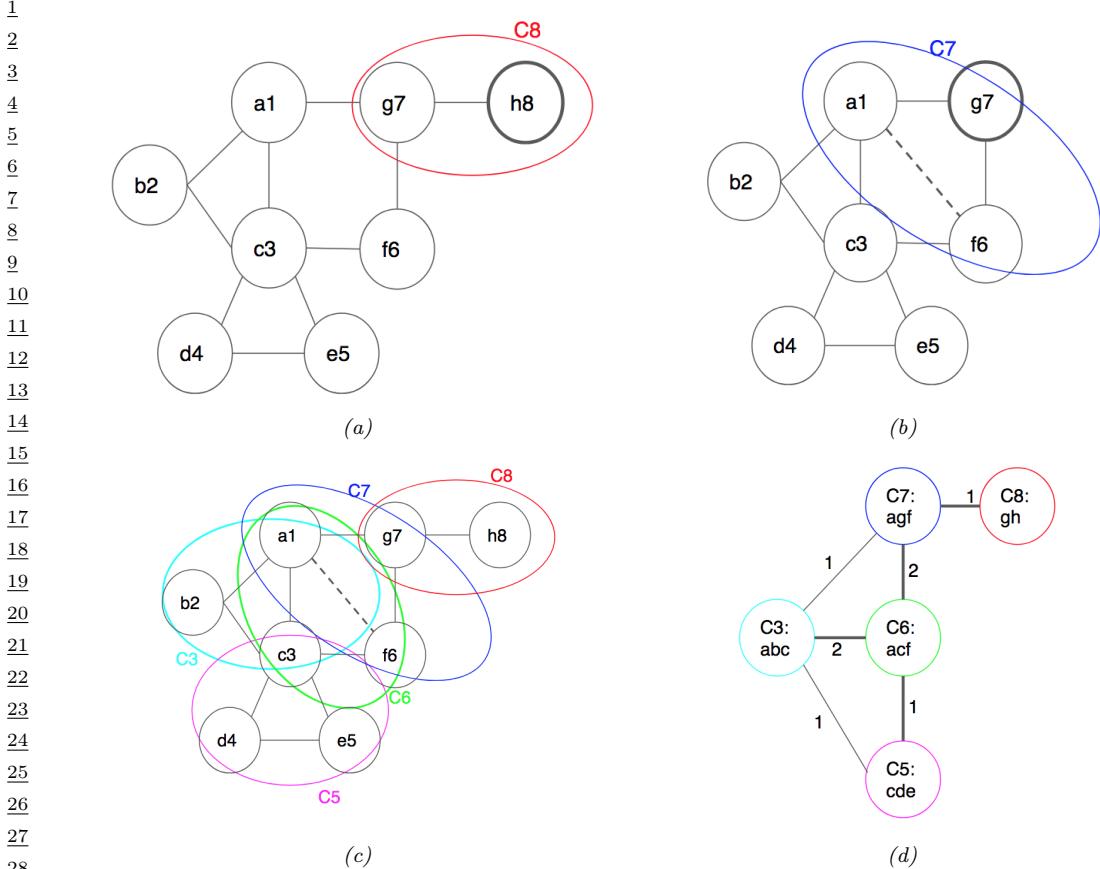


Figure 9.13: (a-b) Illustration of two steps of graph triangulation using the elimination order (a,b,c,d,e,f,g,h) applied to the graph in Figure 9.12a. The node being eliminated is shown with a darker border. Cliques are numbered by the vertex that created them. The dotted a-f line is a fill-in edge 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

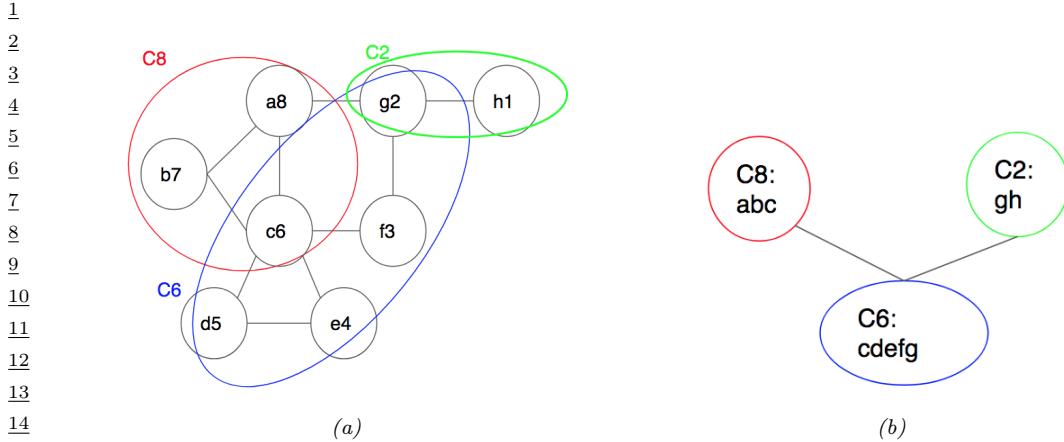


Figure 9.14: (a) Maximal cliques induced by using the elimination order (h, g, f, e, d, c, b, a) . The fill-in edges for clique C_6 are not shown, to reduce clutter. (b) Corresponding junction graph/tree. The largest clique has size 5, which is much larger than in Figure 9.13d.

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 appoximation algorithms. [Dav+04] present the approximate minimum degree ordering algorithm, which is implemented in Matlab.³

38

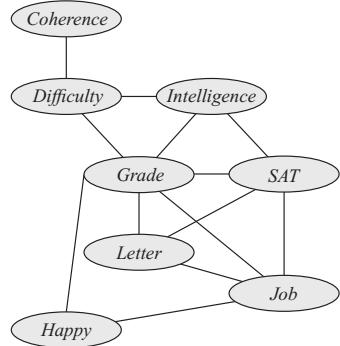
39 9.5.1.5 Application to graphical models

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

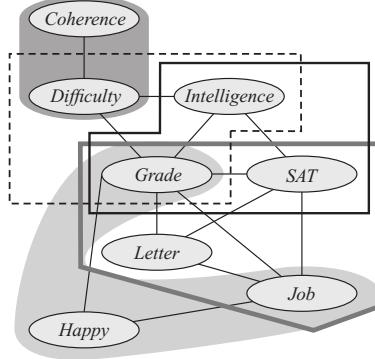
44 3. See the description of the symamd command at <https://bit.ly/31N6E2b>. (“sym” stands for symbolic, “amd” stands
45 approximate minimum degree.) This method is used to find good elimination orderings for Cholesky decompositions of
46 sparse matrices.

47

1
2
3
4
5
6
7
8
9
10
11
12



(a)



(b)

13
14
15
16
17
18
19
20
21
22
23
24
25
26
27



(c)

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
1	$[0, \infty)$	$(+, 0)$	$(\times, 1)$	sum-product
2	$[0, \infty)$	$(\max, 0)$	$(\times, 1)$	max-product
3	$(-\infty, \infty]$	(\min, ∞)	$(+, 0)$	min-sum
4	$\{T, F\}$	(\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:

$$p(\mathbf{x}) = \frac{1}{Z} \prod_c \psi_c(\mathbf{x}_c) = \exp\left(\sum_c \boldsymbol{\eta}_c^\top \mathcal{T}(\mathbf{x}_c) - \log A(\boldsymbol{\eta})\right) = \exp(\boldsymbol{\eta}^\top \mathcal{T}(\mathbf{x}) - \log A(\boldsymbol{\eta})) \quad (9.59)$$

1 where ψ_c is the potential function for clique c , η_c are the natural parameters for clique c , $\mathcal{T}(\mathbf{x}_c)$ are
 2 the corresponding sufficient statistics, and $A = \log Z$ is the log partition function.
 3

4 We will consider pairwise models (with node and edge potentials), and discrete variables. The
 5 natural parameters are the node and edge log potentials, $\boldsymbol{\eta} = (\{\eta_{s;j}\}, \{\eta_{s,t;j,k}\})$, and the sufficient
 6 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:
 7 we use $s, t \in \mathcal{V}$ to index nodes and $j, k \in \mathcal{X}$ to index states.)

8 The mean of the sufficient statistics are given by

$$9 \quad \boldsymbol{\mu} = \mathbb{E}[\mathcal{T}(\mathbf{x})] = (\{p(x_s = j)\}_s, \{p(x_s = j, x_t = k)\}_{s \neq t}) = (\{\mu_{s;j}\}_s, \{\mu_{s,t;j,k}\}_{s \neq t}) \quad (9.60)$$

11 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
 12 computes $A(\boldsymbol{\eta}) = \log Z(\boldsymbol{\eta})$, we can use automatic differentiation (Section 6.2) to compute gradients,
 13 and then we can extract the corresponding node marginals from the gradient vector. (If we have
 14 evidence (known values) on some of the variables, we simply “clamp” the corresponding entries to 0
 15 or 1 in the node potentials.)

16 As a concrete example, consider a chain structured model $x_1 - x_2 - x_3$, where each node has K
 17 states. We can represent the node potentials as $K \times 1$ tensors (table of numbers), and the edge
 18 potentials by $K \times K$ tensors. The partition function is given by

$$19 \quad Z(\boldsymbol{\psi}) = \sum_{x_1, x_2, x_3} \psi_1(x_1) \psi_2(x_2) \psi_3(x_3) \psi_{12}(x_1, x_2) \psi_{23}(x_2, x_3) \quad (9.61)$$

22 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
 23 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
 24 marginals $\boldsymbol{\mu}_{s,t}(j, k) = p(x_s = j, x_t = k)$ using $\boldsymbol{\mu}_{s,t} = \nabla_{\boldsymbol{\eta}_{s,t}} A(\boldsymbol{\eta})$.

25 We can compute the partition function Z efficiently use numpy’s **einsum** function, which imple-
 26 ments tensor contraction using Einstein summation notation. We label each dimension of the tensors
 27 by A, B and C, so einsum knows how to match things up. We then compute gradients using an
 28 auto-diff library.⁴ The result is that inference can be done in two lines of Python code:

```
29
30 logZ_fun = lambda logpots: np.log(jnp.einsum('A,B,C,AB,BC',
31           *[np.exp(lp) for lp in logpots]))
32 probs = grad(logZ_fun)(logpots)
```

33 To perform conditional inference, such as $p(x_s = k | x_t = e)$, we multiply in one-hot indicator
 34 vectors to clamp x_t to the value e so that the unnormalized joint only assigns non-zero probability to
 35 state combinations that are valid. We then sum over all values of the unclamped variables to get the
 36 constrained partition function Z_e . The gradients will now give us the marginals conditioned on the
 37 evidence [Dar03].

38 To handle real-valued observations, we can just write down the unnormalized log joint where we
 39 include factors for each piece of local evidence. For example, consider an HMM (Chapter 30) defining
 40 the following log joint:

$$41 \quad \log p(\mathbf{z}, \mathbf{y}) = \log \text{Cat}(z_1 | \boldsymbol{\pi}) + \sum_{t=1}^{T-1} \log \text{Cat}(z_{t+1} | \mathbf{P}_{z_t,:}) + \sum_{t=1}^T \log p(\mathbf{y}_t | z_t) \quad (9.62)$$

45 4. We use JAX. See [ugm_inf_autodiff.py](#) for the full code, and see <https://github.com/srush/ProbTalk> for a PyTorch
 46 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:

$$\boldsymbol{\psi}^* = \underset{\boldsymbol{\psi}}{\operatorname{argmin}} D_{\text{KL}}(q(\mathbf{z}|\boldsymbol{\psi}) \| p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})) \quad (10.2)$$

$$= \underset{\boldsymbol{\psi}}{\operatorname{argmin}} \mathbb{E}_{q(\mathbf{z}|\boldsymbol{\psi})} \left[\log q(\mathbf{z}|\boldsymbol{\psi}) - \log \left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{x})} \right) \right] \quad (10.3)$$

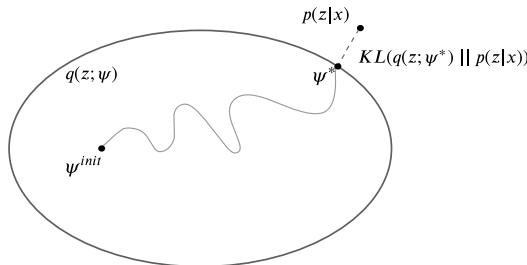
$$= \underset{\boldsymbol{\psi}}{\operatorname{argmin}} \underbrace{\mathbb{E}_{q(\mathbf{z}|\boldsymbol{\psi})} [\log q(\mathbf{z}|\boldsymbol{\psi}) - \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) - \log p_{\boldsymbol{\theta}}(\mathbf{z})]}_{\mathcal{L}(\boldsymbol{\psi}|\boldsymbol{\theta}, \mathbf{x})} + \log p_{\boldsymbol{\theta}}(\mathbf{x}) \quad (10.4)$$

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.

1
2
3
4
5
6
7
8
9
10



11
12 *Figure 10.1: Illustration of variational inference. The large oval represents the set of variational distributions*
13 $Q = \{q(z; \psi) : \psi \in \mathbb{R}^K\}$, where K is the number of variational parameters. The true distribution is the point
14 $p(z|x)$, which we assume lies outside the set. Our goal is to find the best approximation to p within our
15 variational family; this is the point ψ^* which is closest in KL divergence. We find this point by starting an
16 optimization procedure from the random initial point ψ^{init} . Adapted from a figure by David Blei.

16
17

18 If we define $\mathcal{E}(z) = -\log p_{\theta}(z, x)$ as the energy, then we can write

$$19 \quad \mathcal{L}(\psi|\theta, x) = \mathbb{E}_{q(z|\psi)} [\mathcal{E}(z)] - \mathbb{H}(q) \quad (10.6)$$

20 where $\mathbb{H}(q)$ is the entropy. In physics, this is known as the **variational free energy**. We can
21 interpret this as the expected energy minus the entropy:
22

$$23 \quad \text{VFE} = \text{expected energy} - \text{entropy} \quad (10.7)$$

24 This is an upper bound on the **free energy**, $-\log p_{\theta}(x)$, which follows from the fact that
25

$$26 \quad D_{\text{KL}}(q||p) = \text{VFE}(\psi|\theta, x) + \log p_{\theta}(x) \geq 0 \quad (10.8)$$

27 Our goal is to minimize the VFE. See Figure 10.1 for an illustration.
28

31 10.1.2 Evidence lower bound (ELBO)

32 The negative of the VFE is known as the **evidence lower bound** or **ELBO** function [BKM16]:
33

$$34 \quad \mathbb{L}(\psi|\theta, x) \triangleq \mathbb{E}_{q(z|\psi)} [\log p_{\theta}(x, z) - \log q(z|\psi)] \quad (10.9)$$

35 The name “ELBO” arises because
36

$$37 \quad \mathbb{L}(\psi|\theta, x) \leq \log p_{\theta}(x) \quad (10.10)$$

38 where $\log p_{\theta}(x)$ is called the “evidence”. The inequality follows from Equation (10.8). Therefore
39 maximizing the ELBO wrt ψ will decrease the original KL, since $\log p_{\theta}(x)$ is a constant wrt ψ .
40 (Note: we use the symbol \mathbb{L} for the ELBO, rather than \mathcal{L} , since the latter denotes a loss we want to
41 minimize.)

42 We can rewrite the ELBO as follows:
43

$$44 \quad \mathbb{L}(\psi|\theta, x) = \mathbb{E}_{q(z|\psi)} [\log p_{\theta}(x, z)] + \mathbb{H}(q(z|\psi)) \quad (10.11)$$

1 We can interpret this
2

$$\frac{3}{4} \text{ELBO} = \text{expected log joint} + \text{entropy of posterior} \quad (10.12)$$

5 The second term encourages the posterior to be maximum entropy, while the first term encourages it
6 to be a joint MAP configuration,

7 We can also rewrite the ELBO in the following equivalent way:
8

$$\frac{9}{10} \mathcal{L}(\psi|\theta, x) = \mathbb{E}_{q(z|\psi)} [\log p_\theta(x|z)] - D_{\text{KL}}(q(z|\psi)\|p_\theta(z)) \quad (10.13)$$

11 We can interpret this as follows:

$$\frac{12}{13} \text{ELBO} = \text{expected log likelihood} - \text{KL from posterior to prior} \quad (10.14)$$

14 The KL term acts like a regularizer, preventing the posterior from diverging too much from the prior.
15

16 10.2 Mean field VI

18 A common approximation in variational inference is to assume that all the latent variables are
19 independent of each other, i.e.,
20

$$\frac{21}{22} q(z|\psi) = \prod_{j=1}^J q_j(z_j) \quad (10.15)$$

24 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
25 variational parameters for the j 'th distribution. This is called the **mean field** approximation.

26 From Equation (10.11), the ELBO becomes
27

$$\frac{28}{29} \mathcal{L}(\psi) = \int q(z|\psi) \log p_\theta(x, z) dz + \sum_{j=1}^J \mathbb{H}(q_j) \quad (10.16)$$

31 since the entropy of a product distribution is the sum of entropies of each component in the product.
32 We can either directly optimize this (see e.g., [Baq+16]), or use a coordinate-wise optimization
33 scheme, as we discuss in Section 10.2.1.
34

35 10.2.1 Coordinate ascent variational inference (CAVI)

37 In this section, we discuss a coordinate ascent method for optimizing the mean field objective, which
38 we call **coordinate ascent variational inference** or **CAVI**.

39 To derive the update equations, we initially assume there are just 3 discrete latent variables, so
40 the ELBO is given by

$$\frac{41}{42} \mathcal{L}(q_1, q_2, q_3) = \sum_{z_1} \sum_{z_2} \sum_{z_3} q_1(z_1) q_2(z_2) q_3(z_3) \log \tilde{p}(z_1, z_2, z_3) + \sum_{j=1}^3 \mathbb{H}(q_j) \quad (10.17)$$

45 where $\tilde{p}(z) = p_\theta(z, x)$ is the unnormalized joint. (We omit θ and x from the notation, since in this
46 section, we assume both are fixed, i.e., we are just performing inference for the latent variables given
47

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

$$\mathbb{L}_2(q_2) = \sum_{z_2} q_2(z_2) \left[\sum_{z_1} \sum_{z_3} q_1(z_1) q_3(z_3) \log \tilde{p}(z_1, z_2, z_3) \right] + \mathbb{H}(q_2) + \text{const} \quad (10.18)$$

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

$$\mathbb{L}_2(q_2) = -D_{\text{KL}}(q_2 \| f_2) + \text{const} \quad (10.21)$$

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

$$\mathbb{L}(\mathbf{q}) = \sum_{\mathbf{z}_1} \cdots \sum_{\mathbf{z}_J} q_1(\mathbf{z}_1) \cdots q_J(\mathbf{z}_J) \log \tilde{p}(\mathbf{z}_1, \dots, \mathbf{z}_J) + \sum_{j=1}^J \mathbb{H}(q_j) \quad (10.22)$$

13 The optimal estimate of the marginal posterior for each variable is given by

14

$$q_i(\mathbf{z}_i) \propto \exp (\mathbb{E}_{q_{-i}} [\log \tilde{p}(\mathbf{z})]) \quad (10.23)$$

15 where $\mathbb{E}_{q_{-i}} [\log \tilde{p}(\mathbf{z})]$ 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

$$p(\mathbf{x}|\mathbf{z}) = \prod_i p(x_i|z_i) = \exp \left(\sum_i L_i(z_i) \right) \quad (10.24)$$

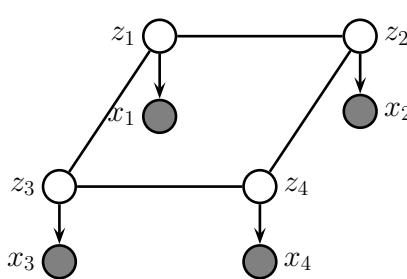


Figure 10.2: A grid-structured MRF with hidden nodes z_i and local evidence nodes x_i . The prior $p(\mathbf{z})$ is an undirected Ising model, and the likelihood $p(\mathbf{x}|\mathbf{z}) = \prod_i p(x_i|z_i)$ is a directed fully factored model.

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.

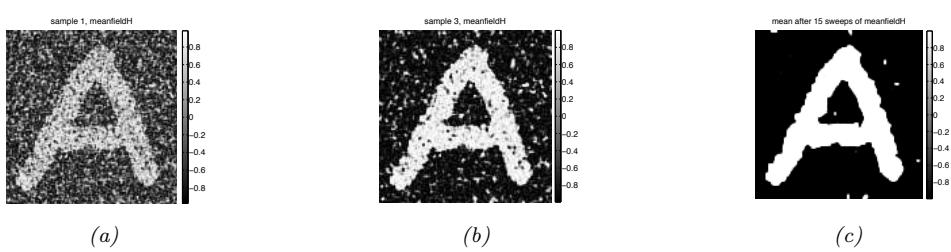


Figure 10.3: Example of image denoising using mean field (with parallel updates and a damping factor of 0.5). We use an Ising prior with $W_{ij} = 1$ and a Gaussian noise model with $\sigma = 2$. We show the results after 1, 3 and 15 iterations across the image. Compare to Figure 12.6, which shows the results of using Gibbs sampling. Generated by `ising_image_denoise_demo.py`.

¹⁵ 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 .
¹⁶ Also, let $L_i^+ \triangleq L_i(+1)$ and $L_i^- \triangleq L_i(-1)$. The approximate marginal posterior is given by
¹⁷

$$\frac{18}{q_i(z_i = 1)} = \frac{e^{m_i + L_i^+}}{e^{m_i + L_i^+} + e^{-m_i + L_i^-}} = \frac{1}{1 + e^{-2m_i + L_i^- - L_i^+}} = \sigma(2a_i) \quad (10.32)$$

$$a_i \triangleq m_i + 0.5(L_i^+ - L_i^-) \quad (10.33)$$

²² Similarly, we have $q_i(z_i = -1) = \sigma(-2a_i)$. From this we can compute the new mean for site i :

$$\mu_i = \mathbb{E}_{\sigma^*}[z_i] = a_i(z_i = +1) \cdot (+1) + a_i(z_i = -1) \cdot (-1) \quad (10.34)$$

$$= \frac{1}{1+e^{-2a_i}} - \frac{1}{1+e^{2a_i}} = \frac{e^{a_i}}{e^{a_i} + e^{-a_i}} - \frac{e^{-a_i}}{e^{-a_i} + e^{a_i}} = \tanh(a_i) \quad (10.35)$$

²⁷ We can turn the above equations into a fixed point algorithm by writing

$$\underline{28} \quad \underline{29} \quad \underline{30} \quad \mu_i^t = \tanh \left(\sum_{j \in \text{nbr}_i} W_{ij} \mu_j^{t-1} + 0.5(L_i^+ - L_i^-) \right) \quad \underline{31} \quad (10.36)$$

³² It is usually better to use **damped updates** of the form

$$\underline{33} \quad \underline{34} \quad \mu_i^t = (1 - \lambda)\mu_i^{t-1} + \lambda \tanh \left(\sum_{j \in \text{nbr}_i} W_{ij} \mu_j^{t-1} + 0.5(L_i^+ - L_i^-) \right) \quad (10.37)$$

$$\underline{35}$$

$$\underline{36}$$

³⁷ for $0 < \lambda < 1$. We can update all the nodes in parallel, or update them asynchronously.

³⁸ Figure 10.3 shows the method in action, applied to a 2d Ising model with homogeneous attractive
³⁹ potentials, $W_{ij} = 1$. We use parallel updates with a damping factor of $\lambda = 0.5$. (If we don't use
⁴⁰ damping, we tend to get "checkerboard" artefacts.)

10.2.3 Variational Bayes

⁴⁴ In Bayesian modeling, we treat the parameters θ as latent variables. Thus our goal is to approximate the parameter posterior $p(\theta|D) \propto p(\theta)p(D|\theta)$. Applying VI to this problem is called **variational Bayes** [Att00].

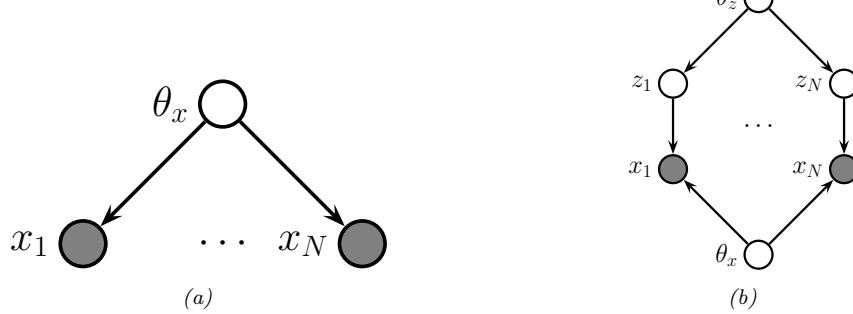


Figure 10.4: Graphical models with (a) Global hidden variable θ_x and observed variables $x_{1:N}$. (b) Local hidden variables $z_{1:N}$, global hidden variables θ_x, θ_z , and observed variables $x_{1:N}$.

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 **10.2.4.1 Target distribution**

2 The unnormalized log posterior has the form

$$\log \tilde{p}(\mu, \lambda) = \log p(\mu, \lambda, \mathcal{D}) = \log p(\mathcal{D}|\mu, \lambda) + \log p(\mu|\lambda) + \log p(\lambda) \quad (10.43)$$

$$\begin{aligned} &= \frac{N}{2} \log \lambda - \frac{\lambda}{2} \sum_{n=1}^N (x_n - \mu)^2 - \frac{\kappa_0 \lambda}{2} (\mu - \mu_0)^2 \\ &\quad + \frac{1}{2} \log(\kappa_0 \lambda) + (a_0 - 1) \log \lambda - b_0 \lambda + \text{const} \end{aligned} \quad (10.44)$$

12 **10.2.4.2 Updating $q(\mu|\psi_\mu)$**

14 The optimal form for $q(\mu|\psi_\mu)$ is obtained by averaging over λ :

$$\log q(\mu|\psi_\mu) = \mathbb{E}_{q(\lambda|\psi_\lambda)} [\log p(\mathcal{D}|\mu, \lambda) + \log p(\mu|\lambda)] + \text{const} \quad (10.45)$$

$$= -\frac{\mathbb{E}_{q(\lambda|\psi_\lambda)} [\lambda]}{2} \left\{ \kappa_0(\mu - \mu_0)^2 + \sum_{n=1}^N (x_n - \mu)^2 \right\} + \text{const} \quad (10.46)$$

21 By completing the square one can show that $q(\mu|\psi_\mu) = \mathcal{N}(\mu|\mu_N, \kappa_N^{-1})$, where

$$\mu_N = \frac{\kappa_0 \mu_0 + N \bar{x}}{\kappa_0 + N}, \quad \kappa_N = (\kappa_0 + N) \mathbb{E}_{q(\lambda|\psi_\lambda)} [\lambda] \quad (10.47)$$

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.

29 **10.2.4.3 Updating $q(\lambda|\psi_\lambda)$**

31 The optimal form for $q(\lambda|\psi_\lambda)$ is given by

$$\log q(\lambda|\psi_\lambda) = \mathbb{E}_{q(\mu|\psi_\mu)} [\log p(\mathcal{D}|\mu, \lambda) + \log p(\mu|\lambda) + \log p(\lambda)] + \text{const} \quad (10.48)$$

$$\begin{aligned} &= (a_0 - 1) \log \lambda - b_0 \lambda + \frac{1}{2} \log \lambda + \frac{N}{2} \log \lambda \\ &\quad - \frac{\lambda}{2} \mathbb{E}_{q(\mu|\psi_\mu)} \left[\kappa_0(\mu - \mu_0)^2 + \sum_{n=1}^N (x_n - \mu)^2 \right] + \text{const} \end{aligned} \quad (10.49)$$

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

$$a_N = a_0 + \frac{N+1}{2} \quad (10.50)$$

$$b_N = b_0 + \frac{1}{2} \mathbb{E}_{q(\mu|\psi_\mu)} \left[\kappa_0(\mu - \mu_0)^2 + \sum_{n=1}^N (x_n - \mu)^2 \right] \quad (10.51)$$

1 **10.2.4.4 Computing the expectations**

3 To implement the updates, we have to specify how to compute the various expectations. Since
4 $q(\mu) = \mathcal{N}(\mu|\mu_N, \kappa_N^{-1})$, we have
5

$$\mathbb{E}_{q(\mu)} [\mu] = \mu_N \quad (10.52)$$

$$\mathbb{E}_{q(\mu)} [\mu^2] = \frac{1}{\kappa_N} + \mu_N^2 \quad (10.53)$$

10 Since $q(\lambda) = \text{Ga}(\lambda|a_N, b_N)$, we have

$$\mathbb{E}_{q(\lambda)} [\lambda] = \frac{a_N}{b_N} \quad (10.54)$$

14 We can now give explicit forms for the update equations. For $q(\mu)$ we have

$$\mu_N = \frac{\kappa_0 \mu_0 + N \bar{x}}{\kappa_0 + N} \quad (10.55)$$

$$\kappa_N = (\kappa_0 + N) \frac{a_N}{b_N} \quad (10.56)$$

20 and for $q(\lambda)$ we have

$$a_N = a_0 + \frac{N+1}{2} \quad (10.57)$$

$$b_N = b_0 + \frac{1}{2} \kappa_0 (\mathbb{E} [\mu^2] + \mu_0^2 - 2\mathbb{E} [\mu] \mu_0) + \frac{1}{2} \sum_{n=1}^N (x_n^2 + \mathbb{E} [\mu^2] - 2\mathbb{E} [\mu] x_n) \quad (10.58)$$

27 We see that μ_N and a_N are in fact fixed constants, and only κ_N and b_N need to be updated
28 iteratively. (In fact, one can solve for the fixed points of κ_N and b_N analytically, but we don't do
29 this here in order to illustrate the iterative updating scheme.)
30

31 **10.2.4.5 Illustration**

33 Figure 10.5 gives an example of this method in action. The green contours represent the exact
34 posterior, which is Gaussian-Gamma. The dotted red contours represent the variational approximation
35 over several iterations. We see that the final approximation is reasonably close to the exact solution.
36 However, it is more "compact" than the true distribution. It is often the case that mean field inference
37 underestimates the posterior uncertainty, for reasons explained in Section 5.1.3.2.
38

39 **10.2.4.6 Lower bound**

41 In VB, we maximize a lower bound on the log marginal likelihood:

$$\mathcal{L}(\psi_\theta | \mathcal{D}) \leq \log p(\mathcal{D}) = \log \int \int p(\mathcal{D} | \mu, \lambda) p(\mu, \lambda) d\mu d\lambda \quad (10.59)$$

45 It is very useful to compute the lower bound itself, for three reasons. First, it can be used to assess
46 convergence of the algorithm. Second, it can be used to assess the correctness of one's code: as with
47

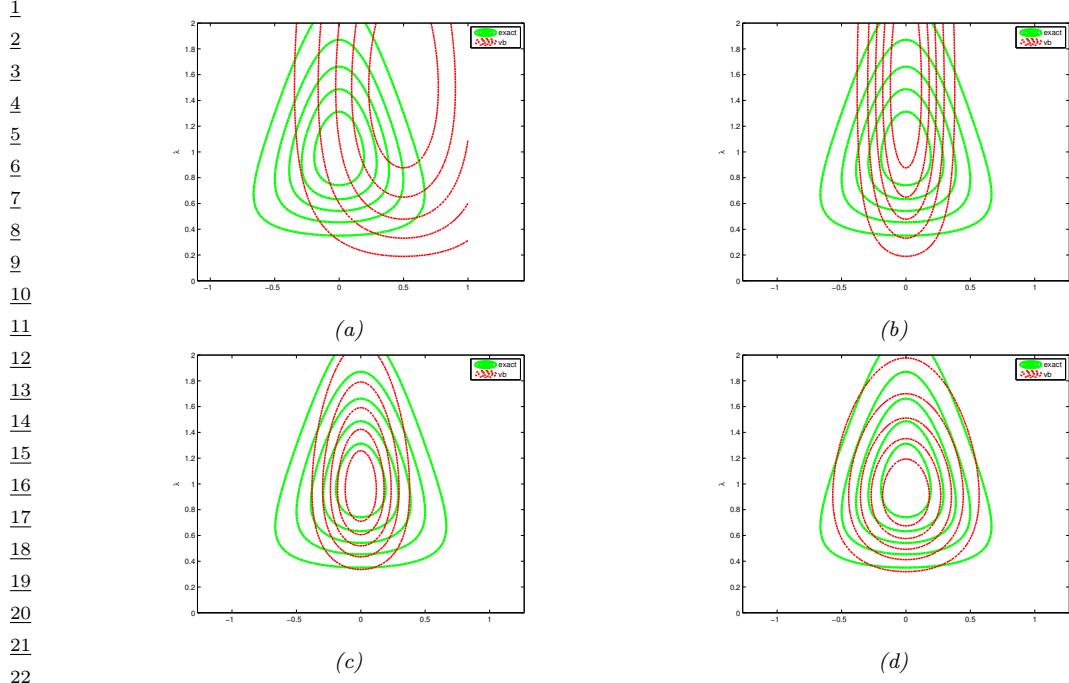


Figure 10.5: Factored variational approximation (red) to the Gaussian-Gamma distribution (green). (a) Initial guess. (b) After updating $q(\mu|\psi_\mu)$. (c) After updating $q(\lambda|\psi_\lambda)$. (d) At convergence (after 5 iterations). Adapted from Fig. 10.4 of [Bis06]. Generated by `unigauss_vb_demo.py`.

26

27

28 EM, if we use CAVI to optimize the objective, the bound should increase monotonically at each
 29 iteration, otherwise there must be a bug. Third, the bound can be used as an approximation to the
 30 marginal likelihood, which can be used for Bayesian model selection. One can show that the lower
 31 bound has the following form:

32

$$33 \quad \mathcal{L} = \text{const} + \frac{1}{2} \ln \frac{1}{\kappa_N} + \ln \Gamma(a_N) - a_N \ln b_N \quad (10.60)$$

35

36 10.2.5 Variational Bayes EM

37

38 In Bayesian latent variable models, we have two forms of hidden variables: local (or per example)
 39 hidden variables \mathbf{z}_n , and global (shared) hidden variables $\boldsymbol{\theta}$, which represent the parameters of the
 40 model. See Figure 10.4b for an illustration. (Note that the parameters, which are fixed in number,
 41 are sometimes called **intrinsic variables**, whereas the local hidden variables are called **extrinsic**
 42 **variables**.) If $\mathbf{h} = (\boldsymbol{\theta}, \mathbf{z}_{1:N})$ represents all the hidden variables, then the joint distribution is given
 43 by

44

$$45 \quad p(\mathbf{h}, \mathcal{D}) = p(\boldsymbol{\theta}, \mathbf{z}_{1:N}, \mathcal{D}) = p(\boldsymbol{\theta}) \prod_{n=1}^N p(\mathbf{z}_n | \boldsymbol{\theta}) p(\mathbf{x}_n | \mathbf{z}_n, \boldsymbol{\theta}) \quad (10.61)$$

46

47

1 We will make the following mean field assumption:

$$\frac{4}{5} q(\boldsymbol{\theta}, \mathbf{z}_{1:N}) = q(\boldsymbol{\theta} | \boldsymbol{\psi}_{\boldsymbol{\theta}}) \prod_{n=1}^N q(\mathbf{z}_n | \boldsymbol{\psi}_n) \quad (10.62)$$

7 We will use VI to maximize the ELBO:

$$\frac{9}{10} \mathcal{L}(\boldsymbol{\psi}_{1:N}, \boldsymbol{\theta} | \mathcal{D}) = \mathbb{E}_{q(\boldsymbol{\theta}, \mathbf{z}_{1:N} | \boldsymbol{\psi}_{1:N}, \boldsymbol{\theta})} [\log p(\mathbf{z}_{1:N}, \boldsymbol{\theta}, \mathcal{D}) - \log q(\boldsymbol{\theta}, \mathbf{z}_{1:N} | \boldsymbol{\psi}_{1:N}, \boldsymbol{\theta})] \quad (10.63)$$

11 If we use the mean field assumption, then we can apply the CAVI approach to optimize each set of
12 variational parameters. In particular, we can alternate between optimizing the $q_n(\mathbf{z}_n)$ in parallel,
13 independently of each other, with $q(\boldsymbol{\theta})$ held fixed, and then optimizing $q(\boldsymbol{\theta})$ with the q_n held fixed.
14 This is known as **variational Bayes EM** [BG06]. It is similar to regular EM, except in the E step,
15 we infer an approximate posterior for \mathbf{z}_n averaging out the parameters (instead of plugging in a point
16 estimate), and in the M step, we update the parameter posterior parameters using the expected
17 sufficient statistics.

18 Now suppose we approximate $q(\boldsymbol{\theta})$ by a delta function, $q(\boldsymbol{\theta}) = \delta(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})$. The Bayesian LVM ELBO
19 objective from Equation (10.63) simplifies to the the “LVM ELBO”:

$$\frac{21}{22} \mathcal{L}(\boldsymbol{\psi}_{1:N}, \boldsymbol{\theta} | \mathcal{D}) = \mathbb{E}_{q(\mathbf{z}_{1:N} | \boldsymbol{\psi}_{1:N})} [\log p(\boldsymbol{\theta}, \mathcal{D}, \mathbf{z}_{1:N}) - \log q(\mathbf{z}_{1:N} | \boldsymbol{\psi}_{1:N})] \quad (10.64)$$

23 We can optimize this using the **variational EM** algorithm, which is a CAVI algorithm which updates
24 the $\boldsymbol{\psi}_n$ in parallel in the variational E step, and then updates $\boldsymbol{\theta}$ in the M step.

25 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
26 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
27 over the parameters. For more details on VEM, see Section 6.7.6.1.

29 10.2.6 Example: VBEM for a GMM

31 Consider a standard Gaussian mixture model (GMM):

$$\frac{32}{33} p(\mathbf{z}, \mathbf{x} | \boldsymbol{\theta}) = \prod_n \prod_k \pi_k^{z_{nk}} \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k^{-1})^{z_{nk}} \quad (10.65)$$

35 where $z_{nk} = 1$ if data point n belongs to cluster k , and $z_{nk} = 0$ otherwise. Our goal is to approximate
36 the posterior $p(\mathbf{z}, \boldsymbol{\theta} | \mathbf{x})$ under the following conjugate prior

$$\frac{38}{39} p(\boldsymbol{\theta}) = \text{Dir}(\boldsymbol{\pi} | \boldsymbol{\alpha}) \prod_k \mathcal{N}(\boldsymbol{\mu}_k | \tilde{\mathbf{m}}, (\tilde{\kappa} \boldsymbol{\Lambda}_k)^{-1}) \text{Wi}(\boldsymbol{\Lambda}_k | \tilde{\mathbf{L}}, \tilde{\nu}) \quad (10.66)$$

41 where $\boldsymbol{\Lambda}_k$ is the precision matrix for cluster k . For the mixing weights, we usually use a symmetric
42 prior, $\boldsymbol{\alpha} = \alpha_0 \mathbf{1}$.

43 The exact posterior $p(\mathbf{z}, \boldsymbol{\theta} | \mathcal{D})$ is a mixture of K^N distributions, corresponding to all possible
44 labelings \mathbf{z} , which is intractable to compute. In this section, we derive a VBEM algorithm, which
45 will approximate the posterior around a local mode. We follow the presentation of [Bis06, Sec 10.2].
46 (See also Section 10.3.5, where we discuss a different variational approximation for this model.)

1 **10.2.6.1 The variational posterior**

2 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
3 stage we have not specified the forms of the q functions; these will be determined by the form of the
4 likelihood and prior. Below we will show that the optimal forms are as follows:
5

6

$$q_n(z_n) = \text{Cat}(z_n | \mathbf{r}_n) \tag{10.67}$$

7

$$q(\boldsymbol{\theta}) = \text{Dir}(\boldsymbol{\pi} | \hat{\boldsymbol{\alpha}}) \prod_k \mathcal{N}(\boldsymbol{\mu}_k | \hat{\mathbf{m}}_k, (\hat{\kappa}_k \boldsymbol{\Lambda}_k)^{-1}) \text{Wi}(\boldsymbol{\Lambda}_k | \hat{\mathbf{L}}_k, \hat{\nu}_k) \tag{10.68}$$

8 where \mathbf{r}_n are the posterior responsibilities, and the parameters with hats on them are the hyperpa-
9 rameters from the prior updated with data.
10

11 **10.2.6.2 Derivation of $q(\boldsymbol{\theta})$ (variational M step)**

12 Using the mean field recipe, we write down the log joint, and take expectations over \mathbf{z} :
13

14

$$\begin{aligned} \log q(\boldsymbol{\theta}) &= \log p(\boldsymbol{\pi}) + \sum_k \log p(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k) + \sum_n \mathbb{E}_{q(z_n)} [\log p(\mathbf{z}_n | \boldsymbol{\pi})] \\ &\quad + \sum_k \sum_n \mathbb{E}_{q(z_n)} [z_{nk}] \log \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k^{-1}) + \text{const} \end{aligned} \tag{10.69}$$

15 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
16 n .

17 We see that the variational posterior factorizes into the form
18

19

$$q(\boldsymbol{\theta}) = q(\boldsymbol{\pi}) \prod_k q(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k) \tag{10.70}$$

20 For the $\boldsymbol{\pi}$ term, we have
21

22

$$\log q(\boldsymbol{\pi}) = (\alpha_0 - 1) \sum_k \log \pi_k + \sum_k \sum_n r_{nk} \log \pi_k + \text{const} \tag{10.71}$$

23 Exponentiating, we recognize this as a Dirichlet distribution:
24

25

$$q(\boldsymbol{\pi}) = \text{Dir}(\boldsymbol{\pi} | \hat{\boldsymbol{\alpha}}) \tag{10.72}$$

26

$$\hat{\alpha}_k = \alpha_0 + N_k \tag{10.73}$$

27

$$N_k = \sum_n r_{nk} \tag{10.74}$$

28

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 $q(\boldsymbol{z})$ (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 $q(\boldsymbol{\pi}) = \text{Dir}(\boldsymbol{\pi} | \boldsymbol{\alpha})$, 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

$$\mathbb{E}_{q(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k)} [(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top \boldsymbol{\Lambda}_k (\boldsymbol{x}_n - \boldsymbol{\mu}_k)] = D \widehat{\kappa}_k^{-1} + \widehat{\nu}_k (\boldsymbol{x}_n - \widehat{\boldsymbol{m}}_k)^\top \widehat{\mathbf{L}}_k (\boldsymbol{x}_n - \widehat{\boldsymbol{m}}_k) \triangleq \tilde{\Lambda}_k \quad (10.86)$$

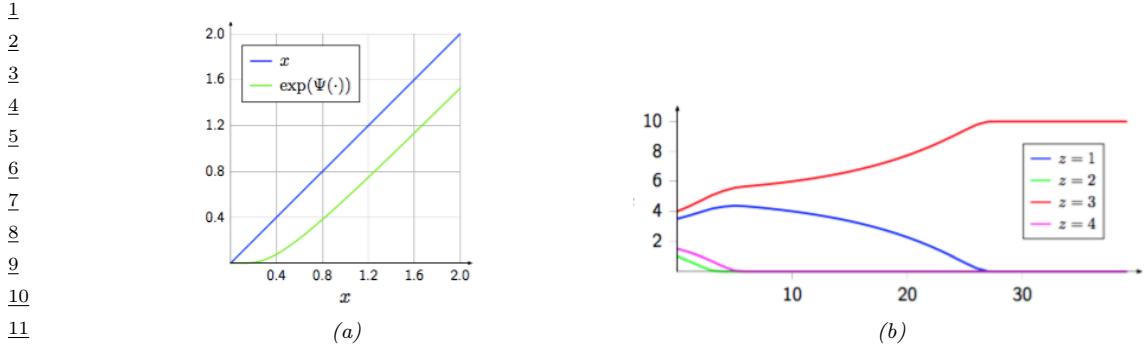


Figure 10.6: (a) We plot $\exp(\psi(x))$ vs x (green line). We see that it performs a form of shrinkage, so that small values get set to zero. (b) We plot N_k vs time for 4 different states (z values), starting from random initial values. We perform a series of VBEM updates, ignoring the likelihood term. 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.

17

18

19

20 Thus we get that the posterior responsibility of cluster k for datapoint n is

21

$$22 \quad r_{nk} \propto \tilde{\pi}_k \tilde{\Lambda}_k^{\frac{1}{2}} \exp \left(-\frac{D}{2 \tilde{\kappa}_k} - \frac{\tilde{\nu}_k}{2} (\mathbf{x}_n - \hat{\boldsymbol{\mu}}_k)^T \tilde{\Lambda}_k (\mathbf{x}_n - \hat{\boldsymbol{\mu}}_k) \right) \quad (10.87)$$

24

25 Compare this to the expression used in regular EM:

26

$$27 \quad r_{nk}^{EM} \propto \hat{\pi}_k |\hat{\Lambda}_k|^{\frac{1}{2}} \exp \left(-\frac{1}{2} (\mathbf{x}_n - \hat{\boldsymbol{\mu}}_k)^T \hat{\Lambda}_k (\mathbf{x}_n - \hat{\boldsymbol{\mu}}_k) \right) \quad (10.88)$$

29

30 where $\hat{\pi}_k$ is the MAP estimate for π_k . The significance of this difference is discussed in Section 10.2.6.4.

31

32 10.2.6.4 Automatic sparsity inducing effects of VBEM

33

34 In regular EM, the E step has the form given in Equation (10.88), whereas in VBEM, the E step has
35 the form given in Equation (10.87). Although they look similar, they differ in an important way. To
36 understand this, let us ignore the likelihood term, and just focus on the prior. Then we have

37

$$38 \quad r_{nk}^{VB} = \tilde{\pi}_k = \frac{\exp(\psi(\hat{\alpha}_k))}{\exp(\psi(\sum_{k'} \hat{\alpha}_{k'}))} \quad (10.89)$$

39

$$40 \quad r_{nk}^{EM} = \hat{\pi}_k = \frac{\hat{\alpha}_k - 1}{\sum_{k'} (\hat{\alpha}_{k'} - 1)} \quad (10.90)$$

42

43 where $\hat{\alpha}_k = \alpha_0 + N_k$, and $N_k = \sum_n r_{nk}$ is the expected number of assignments to cluster k .

44 We know from Figure 2.11 that using $\alpha_0 \ll 1$ causes π to be sparse, which will encourage r_n to be
45 sparse, which will “kill off” unnecessary mixture components (i.e., ones for which $N_k \ll N$, meaning
46 very few data points are assigned to cluster k). To encourage this sparsity promoting effect, let us

47

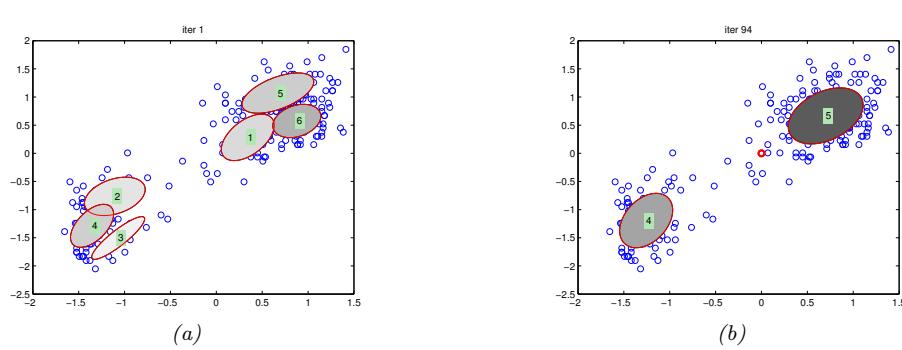


Figure 10.7: We visualize the posterior mean parameters at various stages of the VBEM algorithm applied to a mixture of Gaussians model on the Old Faithful data. Shading intensity is proportional to the mixing weight. We initialize with K-means and use $\alpha_0 = 0.001$ as the Dirichlet hyper-parameter. (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

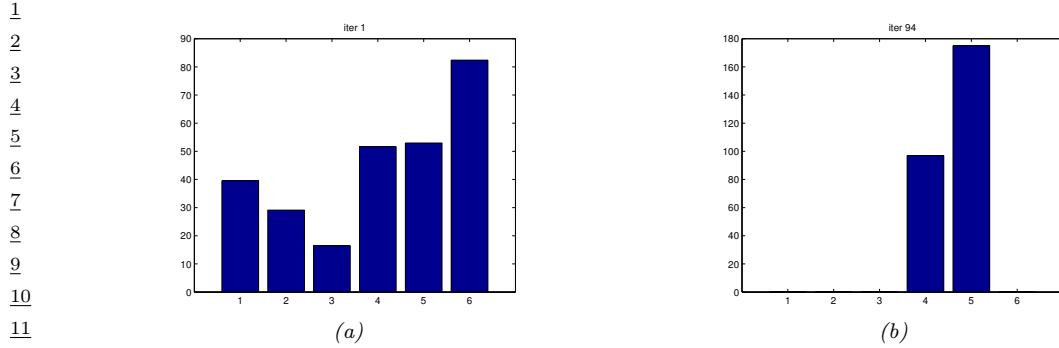


Figure 10.8: We visualize the posterior values of α_k for the model in Figure 10.7 after the first and last iteration of the algorithm. We see that unnecessary components get “killed off”. (Interestingly, the initially large cluster 6 gets “replaced” by cluster 5.) Generated by `variational_mixture_gaussians_demo.py`.

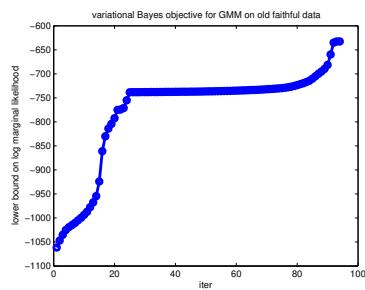


Figure 10.9: Lower bound vs iterations for the VB algorithm in Figure 10.7. The steep parts of the curve correspond to places where the algorithm figures out that it can increase the bound by “killing off” unnecessary mixture components, as described in Section 10.2.6.6. The plateaus correspond to slowly moving the clusters around. Generated by [variational_mixture_gaussians_demo.py](#).

³³ the number of clusters. Similar techniques can be used to choose the number of states in an HMM
³⁴ and other latent variable models. However, this **variational pruning effect** (also called **posterior**
³⁵ **collapse**), is not always desirable, since it can cause the model to “ignore” the latent variables \mathbf{z} if
³⁶ the likelihood function $p(\mathbf{x}|\mathbf{z})$ is sufficiently powerful. We discuss this more in Section 22.4.
³⁷

39 10.2.6.5 Lower bound on the marginal likelihood

⁴⁰ The VBEM algorithm is maximizing the following lower bound

$$\frac{42}{43} \quad \mathcal{L} = \sum_{\mathbf{z}} \int d\boldsymbol{\theta} q(\mathbf{z}, \boldsymbol{\theta}) \log \frac{p(\mathbf{x}, \mathbf{z}, \boldsymbol{\theta})}{q(\mathbf{z}, \boldsymbol{\theta})} \leq \log p(\mathbf{x}) \quad (10.93)$$

⁴⁶ 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 $p(K|\mathcal{D})$:

7
$$p(K|\mathcal{D}) = \frac{e^{\mathcal{L}(K)}}{\sum_{K'} e^{\mathcal{L}(K')}} \quad (10.94)$$

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 $\log p(\mathcal{D}|K) \approx \mathcal{L}(K) + \log(K!)$.

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.,

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
$$= \exp \left[\log h(\mathbf{z}_j) + \mathbb{E} [\boldsymbol{\eta}_j(\mathbf{z}_{-j}, \mathbf{x})]^\top \mathcal{T}(\mathbf{z}_j) - \mathbb{E} [A_j(\boldsymbol{\eta}_j(\mathbf{z}_{-j}, \mathbf{x}))] \right] \quad (10.97)$$

30
$$\propto h(\mathbf{z}_j) \exp \left[\mathbb{E} [\boldsymbol{\eta}_j(\mathbf{z}_{-j}, \mathbf{x})]^\top \mathcal{T}(\mathbf{z}_j) \right] \quad (10.98)$$

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 VMP can be extended to the case where each full conditional is conditionally conjugate using the
45 CVI framework in Section 10.3.8. See also [ABV21], where they use local Laplace approximations to
46 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

$$\underline{30} \quad \ell_{\boldsymbol{\psi}}(\mathbf{z}) = \log p_{\boldsymbol{\theta}}(\mathbf{z}, \mathcal{D}) - \log q_{\boldsymbol{\psi}}(\mathbf{z}) \quad (10.101)$$

31

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:

$$\nabla_{\psi} \hat{L}(\psi) = \nabla_{\psi} \int q_{\psi}(\mathbf{z}) \log \frac{p_{\theta}(\mathbf{z}, \mathcal{D})}{q_{\psi}(\mathbf{z})} d\mathbf{z} \quad (10.102)$$

$$= \int [\nabla_{\psi} q_{\psi}(\mathbf{z})] \left[\log \frac{p_{\theta}(\mathbf{z}, \mathcal{D})}{q_{\psi}(\mathbf{z})} \right] d\mathbf{z} + \int [q_{\psi}(\mathbf{z})] \left[\nabla_{\psi} \log \frac{p_{\theta}(\mathbf{z}, \mathcal{D})}{q_{\psi}(\mathbf{z})} \right] d\mathbf{z} \quad (10.103)$$

$$= \int [\nabla_{\psi} q_{\psi}(\mathbf{z})] \left[\log \frac{p_{\theta}(\mathbf{z}) p_{\theta}(\mathcal{D}|\mathbf{z})}{q_{\psi}(\mathbf{z})} \right] d\mathbf{z} - \int q_{\psi}(\mathbf{z}) \nabla_{\psi} \log q_{\psi}(\mathbf{z}) d\mathbf{z} \quad (10.104)$$

$$= \int [\nabla_{\psi} q_{\psi}(\mathbf{z})] \left[\log \frac{p_{\theta}(\mathbf{z}) p_{\theta}(\mathcal{D}|\mathbf{z})}{q_{\psi}(\mathbf{z})} \right] d\mathbf{z} - \int \nabla_{\psi} q_{\psi}(\mathbf{z}) d\mathbf{z} \quad (10.105)$$

$$= \int [\nabla_{\psi} q_{\psi}(\mathbf{z})] \left[\log \frac{p_{\theta}(\mathbf{z}) p_{\theta}(\mathcal{D}|\mathbf{z})}{q_{\psi}(\mathbf{z})} \right] d\mathbf{z} - \nabla_{\psi} \int q_{\psi}(\mathbf{z}) d\mathbf{z} \quad (10.106)$$

$$= \int [\nabla_{\psi} q_{\psi}(\mathbf{z})] \left[\log \frac{p_{\theta}(\mathbf{z}) p_{\theta}(\mathcal{D}|\mathbf{z})}{q_{\psi}(\mathbf{z})} \right] d\mathbf{z} \quad (10.107)$$

$$= \int q_{\psi}(\mathbf{z}) \nabla_{\psi} \log q_{\psi}(\mathbf{z}) \times \ell_{\psi}(\mathbf{z}) d\mathbf{z} \quad (10.108)$$

$$= \mathbb{E}_{q_{\psi}(\mathbf{z})} [\nabla_{\psi} \log q_{\psi}(\mathbf{z}) \times \ell_{\psi}(\mathbf{z})] \quad (10.109)$$

We can compute a stochastic approximation to this gradient by sampling $\mathbf{z}_s \sim q_{\psi}(\mathbf{z})$ and then computing

$$\widehat{\nabla_{\psi} L}(\psi_t) = \frac{1}{S} \sum_{s=1}^S \nabla_{\psi} \log q_{\psi}(\mathbf{z}_s) \times \ell_{\psi}(\mathbf{z}_s) |_{\psi=\psi_t} \quad (10.110)$$

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

$$\widehat{\nabla_{\psi_i} L}(\psi_t)^{\text{naive}} = \frac{1}{S} \sum_{s=1}^S \tilde{g}_i(\mathbf{z}_s) \quad (10.111)$$

$$\tilde{g}_i(\mathbf{z}_s) = g_i(\mathbf{z}_s) \times \ell_{\psi}(\mathbf{z}_s) \quad (10.112)$$

$$g_i(\mathbf{z}_s) = \nabla_{\psi_i} \log q_{\psi}(\mathbf{z}_s) \quad (10.113)$$

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

$$\widehat{\nabla_{\psi_i} \mathbb{L}(\psi_t)}^{\text{cv}} = \frac{1}{S} \sum_{s=1}^S g_i(\mathbf{z}_s) \times (\ell_{\psi}(\mathbf{z}_s) - c_i) \quad (10.116)$$

6 One can show that the optimal c_i that minimizes the variance of the CV estimator is
7

$$c_i = \frac{\text{Cov}[g_i(\mathbf{z})\ell_{\psi}(\mathbf{z}), g_i(\mathbf{z})]}{\mathbb{V}[g_i(\mathbf{z})]} \quad (10.117)$$

10 which can be estimated by sampling $\mathbf{z} \sim q_{\psi}(\mathbf{z})$. 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 $\mathbf{z}_s \sim q_{\psi_0}(\mathbf{z})$, $s = 1 : S$
15 3 $h_{\psi_0}(\mathbf{z}_s) = \log p_{\theta}(\mathbf{z}_s, \mathcal{D}) - \log q_{\psi_0}(\mathbf{z}_s)$, $s = 1 : S$
16 4 $\mathbf{g}_0 = \frac{1}{S} \sum_{s=1}^S [\nabla_{\psi} \log q_{\psi_0}(\mathbf{z}_s)] h_{\psi_0}(\mathbf{z}_s)$
17 5 Compute \mathbf{c}_1 using Equation (10.117) applied to \mathbf{z}_s
18 6 **for** $t = 1 : T$ **do**
19 7 $\mathbf{z}_s \sim q_{\psi_t}(\mathbf{z})$, $s = 1 : S$
20 8 $h_{\psi_t}(\mathbf{z}_s) = \log p_{\theta}(\mathbf{z}_s, \mathcal{D}) - \log q_{\psi_t}(\mathbf{z}_s)$, $s = 1 : S$
21 9 $\mathbf{g}_t = \frac{1}{S} \sum_{s=1}^S [\nabla_{\psi} \log q_{\psi_t}(\mathbf{z}_s)] \odot (h_{\psi_t}(\mathbf{z}_s) - \mathbf{c}_t)$
22 10 Compute \mathbf{c}_{t+1} using Equation (10.117) applied to \mathbf{z}_s
23 11 ψ_t = gradient-update(ψ_{t-1}, \mathbf{g}_t)

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

$$\hat{\mathbb{L}}(\psi) = \frac{1}{S} \sum_{s=1}^S \ell_{\psi}(\mathbf{z}_s) \quad (10.118)$$

32 where $\mathbf{z}_s \sim q_{\psi}(\mathbf{z})$. To smooth out the noise, we can use a running average over the last w observations
33 to get

$$\bar{\mathbb{L}}(\psi_t) = \frac{1}{w} \sum_{k=1}^w \hat{\mathbb{L}}(\psi_{t-k+1}) \quad (10.119)$$

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 \mathbf{z} are the latent variables, and the observed variables are a set of iid observations, $\mathcal{D} = \{\mathbf{x}_n : n = 1 : N\}$. In this case, we have

$$\ell_{\psi}(\mathbf{z}_s) = \sum_{n=1}^N \log p_{\theta}(\mathbf{x}_n | \mathbf{z}_s) + \log p_{\theta}(\mathbf{z}_s) - \log q_{\psi}(\mathbf{z}_s) \quad (10.120)$$

1 If N is large, we can compute an unbiased minibatch approximation to this expression as follows:

2

$$\hat{\ell}_{\psi}(\mathbf{z}_s) = \left[\frac{N}{B} \sum_{b=1}^B \log p_{\theta}(\mathbf{x}_b | \mathbf{z}_s) \right] + \log p_{\theta}(\mathbf{z}_s) - \log q_{\psi}(\mathbf{z}_s) \quad (10.121)$$

3

4 This is called **stochastic variational inference** or **SVI** [Hof+13].

5 We can combine this with the above stochastic gradient approximation of the ELBO to get the
6 following **doubly stochastic** approximation [TLG14]:

7

$$\widehat{\nabla_{\psi} L(\psi_t)} = \frac{1}{S} \sum_{s=1}^S \nabla_{\psi} \log q_{\psi}(\mathbf{z}_s) \times \hat{\ell}_{\psi}(\mathbf{z}_s) |_{\psi=\psi_t} \quad (10.122)$$

8

9 Of course, this can be combined with control variates.

10 10.3.3 Reparameterization VI

11 In this section, we exploit the **reparameterization trick** from Section 6.6.4 to get a lower variance
12 estimator for the gradient. This assumes that $\ell_{\psi}(\mathbf{z})$ is a differentiable function of \mathbf{z} . It also assumes
13 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
14 compute the latent random variables $\mathbf{z} = r(\psi, \epsilon)$. In this case, the ELBO becomes

15

$$L(\psi) = \mathbb{E}_{q_0(\epsilon)} [\ell_{\psi}(r(\psi, \epsilon))] = \mathbb{E}_{q_0(\epsilon)} [\log p_{\theta}(r(\psi, \epsilon), \mathcal{D}) - \log q_{\psi}(r(\psi, \epsilon))] \quad (10.123)$$

16

17 Since the sampling distribution $q_0(\epsilon)$ is independent of the variational parameters ψ , we can push
18 the gradient operator inside the expectation, and thus we can estimate the gradient using standard
19 automatic differentiation methods, as shown in Algorithm 7. This is called **reparameterized VI** or
20 **RVI**, and has provably lower variance than BBVI in certain cases [Xu+19].

21 Algorithm 7: Estimate of ELBO gradient

```
22 1 def elbo( $\psi$ ):  

23 2      $\epsilon \sim q_0(\epsilon)$   

24 3      $\mathbf{z} = r(\psi, \epsilon)$   

25 4     return  $\log p_{\theta}(\mathbf{z}, \mathcal{D}) - q_{\psi}(\mathbf{z} | \mathcal{D})$   

26 5 def elbo-grad( $\psi$ ):  

27 6     return grad(elbo( $\psi$ ))
```

28

29 10.3.3.1 “Sticking the landing” estimator

30 Applying the results from Section 6.6.4.2, we can derive the gradient estimate of the reparameterized
31 ELBO, for a single Monte Carlo sample, as follows:

32

$$\nabla_{\psi} L(\psi, \epsilon) = \nabla_{\psi} [\log p(\mathbf{z}, \mathcal{D}) - \log q_{\psi}(\mathbf{z} | \mathcal{D})] \quad (10.124)$$

33

$$= \underbrace{\nabla_{\mathbf{z}} [\log p(\mathbf{z} | \mathcal{D}) - \log q_{\psi}(\mathbf{z} | \mathcal{D})]}_{\text{path derivative}} \mathbf{J} - \underbrace{\nabla_{\psi} \log q_{\psi}(\mathbf{z} | \mathcal{D})}_{\text{score function}} \quad (10.125)$$

34

1 where $\mathbf{z} = r(\psi, \epsilon)$ and $\mathbf{J} = \nabla_{\psi} r(\psi, \epsilon)$ is the Jacobian matrix of the noise transformation.
 2 The first term is the indirect effect of ψ on the objective via the generated samples \mathbf{z} . The second
 3 term is the direct effect of ψ on the objective. The second term is zero in expectation since it is
 4 the score function (see Equation (2.223)), but it may be non-zero for a finite number of samples,
 5 even if $q_{\psi}(\mathbf{z}|\mathcal{D}) = p(\mathbf{z}|\mathcal{D})$ is the true posterior. In a paper called “**sticking the landing**”, [RWD17]
 6 propose to drop the second term to create a lower variance estimator.² In practice, this means we
 7 compute the gradient using Algorithm 8 instead of Algorithm 7.³
 8

9 **Algorithm 8:** “Sticking the landing” estimator of ELBO gradient

10 1 def elbo-pd(ψ):
 11 2 $\epsilon \sim q_0(\epsilon)$
 12 3 $\mathbf{z} = r(\psi, \epsilon)$
 13 4 $\psi' = \text{stop-gradient}(\psi)$
 14 5 return $\log p_{\theta}(\mathbf{z}, \mathcal{D}) - q_{\psi'}(\mathbf{z}|\mathcal{D})$
 15 6 def elbo-grad-pd(ψ):
 16 7 return grad(elbo-pd(ψ))

19 Note that the STL estimator is not always better than the “standard” estimator. In [GD20], they
 20 propose to use a weighted combination of estimators, where the weights are optimized so as to reduce
 21 variance for a fixed amount of compute.

22 **10.3.4 Gaussian VI**

23 The most widely used RVI approximation is when $q_{\psi}(\mathbf{z})$ is a Gaussian, where $\psi = (\mu, \Sigma)$. Following
 24 [TND21], we call this **Gaussian VI**. We give some examples of this below.

25 **10.3.4.1 Gaussian posterior: Cholesky decomposition**

26 In this section, we represent the covariance using its Cholesky decomposition, $\Sigma = \mathbf{L}\mathbf{L}^T$, where
 27 $\mathbf{L} = \text{tril}(\mathbf{l})$ is a lower triangular matrix, as in [TLG14; TN18]. The variational parameters are
 28 $\psi = (\mu, \mathbf{l})$. The noise transformation has the form

29
$$\mathbf{z} \sim \mathcal{N}(\mu, \Sigma) \iff \mathbf{z} = \mu + \mathbf{L}\epsilon \tag{10.126}$$

30 where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, so $r(\psi, \epsilon) = \mu + \mathbf{L}\epsilon$.

31 We have $\nabla_{\mu} r(\psi, \epsilon) = \mathbf{I}$, so the gradient of the ELBO wrt μ has the form

32
$$\nabla_{\mu} \mathcal{L}(\psi) = \mathbb{E}_{q_0(\epsilon)} [\nabla_{\mathbf{z}} \ell_{\psi}(\mathbf{z})] \tag{10.127}$$

33 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
 34 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

35 2. The expression “to stick a landing” means to land firmly on one’s feet after performing a gymnastics move. In the
 36 current context, the analogy is this: if the variational posterior is optimal, so $q_{\psi}(\mathbf{z}|\mathcal{D}) = p(\mathbf{z}|\mathcal{D})$, then we want our
 37 objective to be 0, and not to “wobble” with Monte Carlo noise.

38 3. The difference is that the path derivative version ignores the score function. This can be achieved by using
 39 $\log q_{\psi'}(\mathbf{z}|\mathcal{D})$, 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 $\mathbf{L}\boldsymbol{\epsilon} = \text{vec}(\mathbf{I} \mathbf{L} \boldsymbol{\epsilon}) = (\boldsymbol{\epsilon}^T \otimes \mathbf{I})\text{vec}(\mathbf{L})$ 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 $q(\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^T)$

```

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

$$\boldsymbol{\Sigma} = \mathbf{B}\mathbf{B}^T + \mathbf{C}^2 \quad (10.135)$$

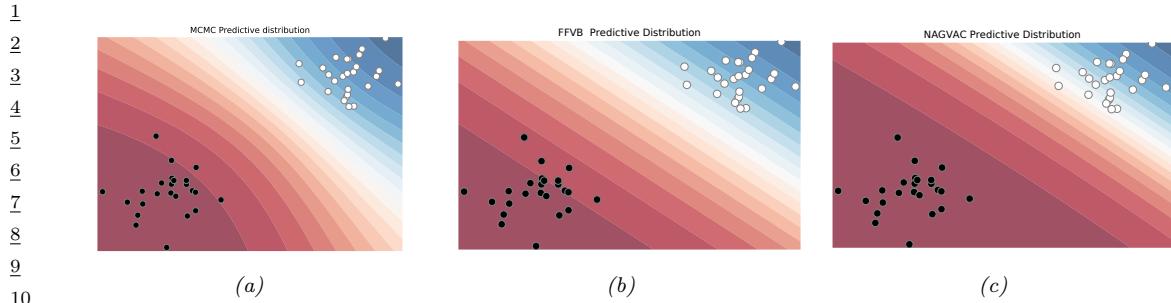


Figure 10.10: Bayesian inference applied to a 2d binary logistic regression problem, $p(y=1|\mathbf{x}) = \sigma(w_0 + w_1x_1 + w_2x_2)$. We show the training data and the posterior predictive produced by different methods. (a) MCMC approximation. (b) VB approximation using full covariance matrix (Cholesky decomposition). (c) VB using rank 1 approximation. Generated by [vb_gauss_biclusters_demo.py](#).

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.

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 possible to compute the natural gradient (Section 6.4) of the ELBO in closed form in $O(d)$ time, as shown in [Tra+20b], [Tra+20b], who call the approach **NAGVAC-1** (Natural Gradient Gaussian variational approximation). This can result in much faster convergence than following the normal gradient.

26 We give an example of this applied to a simple 2d binary logistic regression problem in Figure 10.10c.
 27 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:

$$u_i \sim \mathcal{N}(\mu_i, \sigma^2) \quad (10.136)$$

$$\mu_i = \alpha + \beta^T x_i \quad (10.137)$$

$$\alpha \approx N(0, 10)$$

$$\beta_i \sim \mathcal{N}(0, 1) \quad (10, 139)$$

$$\sigma \sim \text{Unif}(0, 10) \quad (10, 140)$$

⁴² where $x_i = (B_i, A_i, A_i \times B_i)$ are the features, and $y_i = G_i$ is the response

⁴³ We first use HMC, which is often considered the “gold standard” of posterior inference. The
⁴⁴ resulting model fit is shown in Figure 10.11. This shows that GDP increases as a function of
⁴⁵

⁴⁶ 4. We choose this example since it is used as the introductory example in the [Pyro tutorial](#).

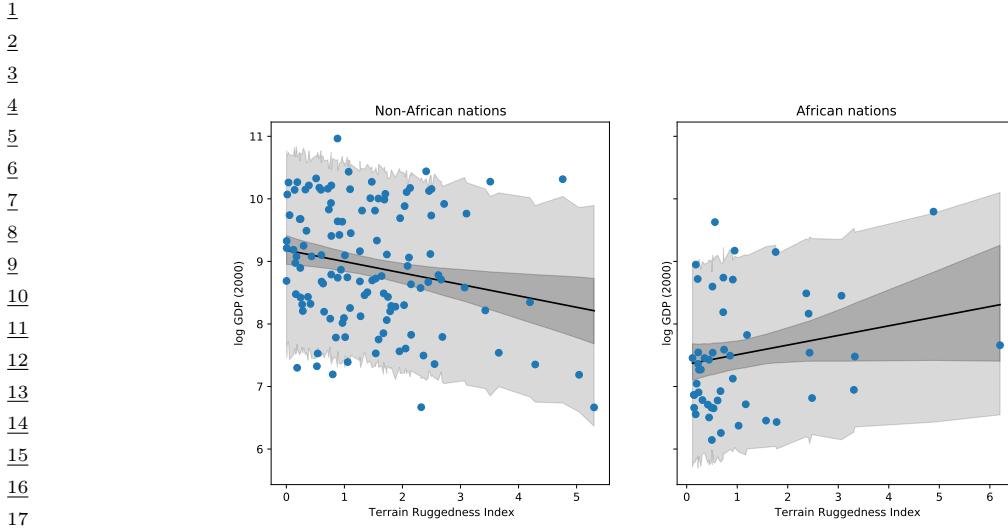


Figure 10.11: Posterior predictive distribution for the linear model applied to the Africa data. Dark shaded region is the 95% credible interval for μ_i . The light shaded region is the 95% credible interval for y_i . Adapted from Figure 8.5 of [McE20]. Generated by `linreg_bayes_svi_hmc_pyro.ipynb`.

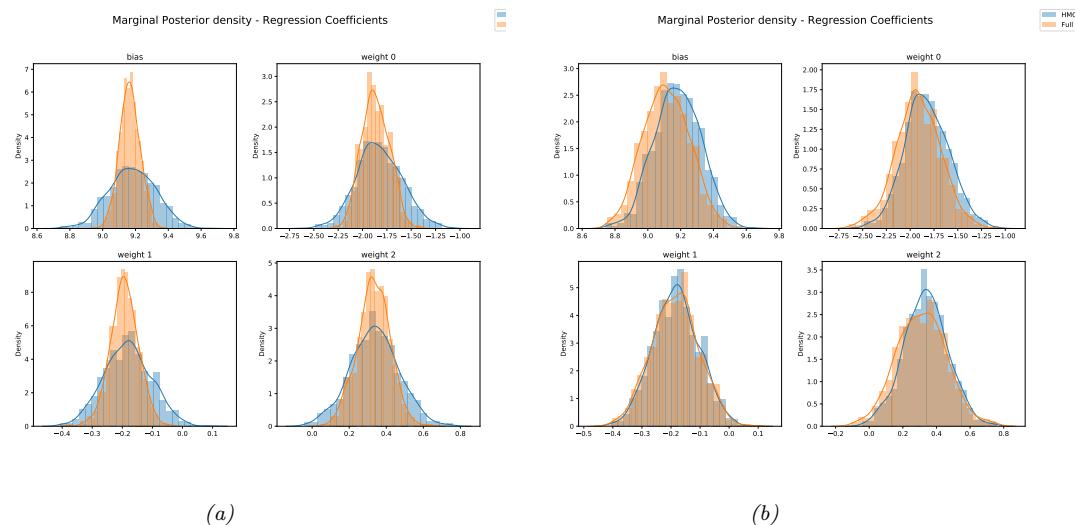


Figure 10.12: Posterior marginals for the linear model applied to the Africa data. (a) Blue is HMC, orange is Gaussian approximation with diagonal covariance. (b) Blue is HMC, orange is Gaussian approximation with full covariance. Generated by `linreg_bayes_svi_hmc_pyro.ipynb`.

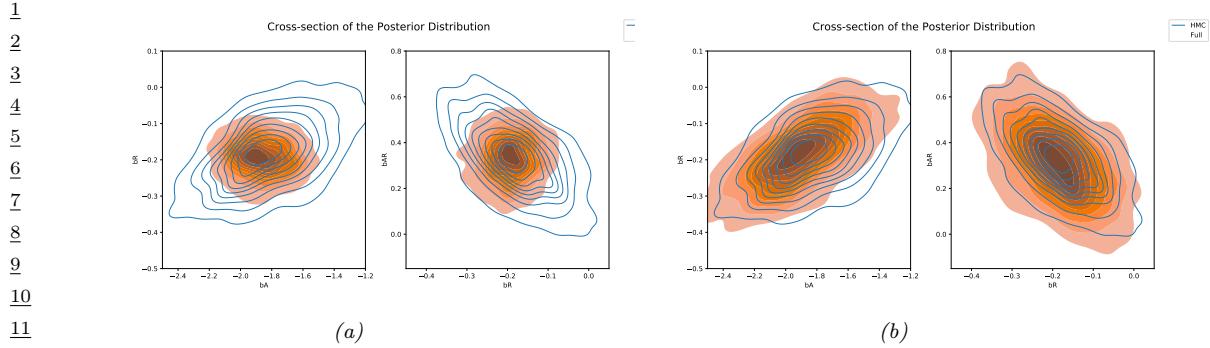


Figure 10.13: Joint posterior of pairs of variables for the linear model applied to the Africa data. (a) Blue is HMC, orange is Gaussian approximation with diagonal covariance. (b) Blue is HMC, orange is Gaussian approximation with full covariance. Generated by `linreg_bayes_svi_hmc_pyro.ipynb`.

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.

10.3.5 Automatic differentiation VI

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

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

$$p(\mathbf{x} | \boldsymbol{\theta}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \text{diag}(\boldsymbol{\lambda}_k)^{-1}) \quad (10.148)$$

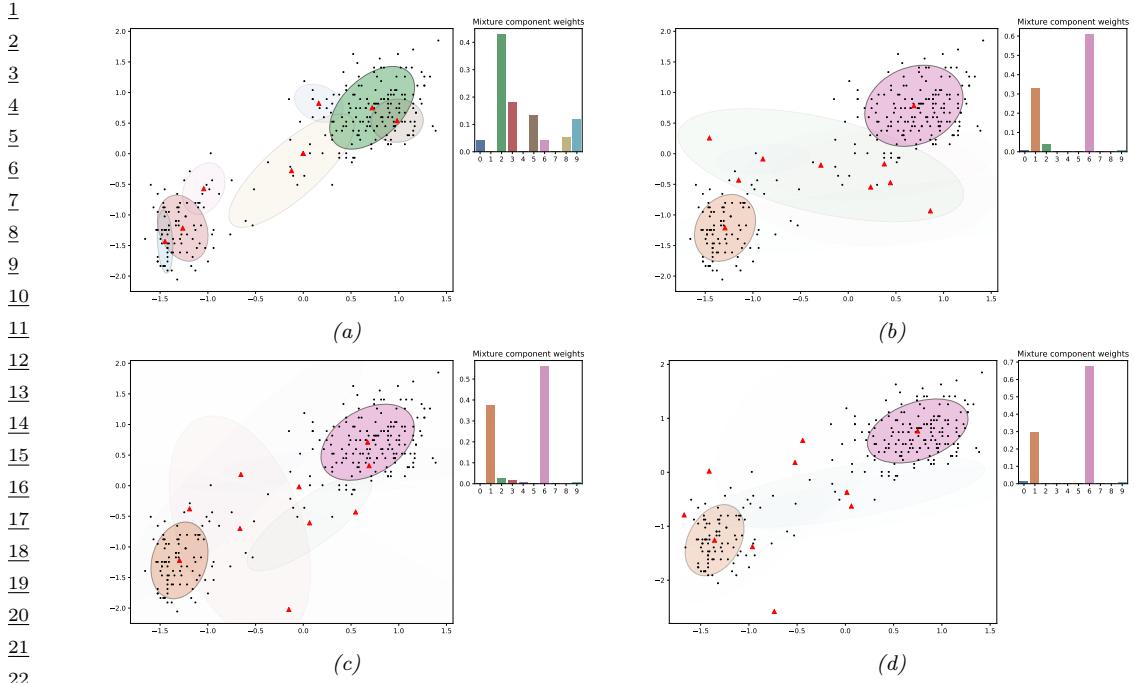


Figure 10.14: Posterior over the mixing weights (histogram) and the means and covariances of each Gaussian mixture component, using $K = 10$, when fitting the model to the Old Faithful dataset from Figure 10.7. (a) MAP approximation. (b-d) 3 samples from the Gaussian approximation. The intensity of the shading is 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

$$36 \quad q(\boldsymbol{\theta}|\psi_{\boldsymbol{\theta}}) = \left[\prod_{k=1}^K \prod_{d=1}^D \mathcal{N}(\mu_{kd}|m_{kd}, s_{kd}) \text{Ga}(\lambda_{kd}|\alpha_{kd}, \beta_{kd}) \right] \text{Dir}(\boldsymbol{\pi}|\mathbf{c}) \quad (10.150)$$

39 where $\psi_{\boldsymbol{\theta}} = (\mathbf{m}_{1:K, 1:D}, \mathbf{s}_{1:K, 1:D}, \boldsymbol{\alpha}_{1:K, 1:D}, \boldsymbol{\beta}_{1:K, 1:D}, \mathbf{c})$ are the variational parameters for $\boldsymbol{\theta}$.
40

We can compute the ELBO using
41

$$42 \quad \mathcal{L}(\psi_{\boldsymbol{\theta}}|\mathcal{D}) = \mathbb{E}_{q(\boldsymbol{\theta}|\psi_{\boldsymbol{\theta}})} \left[\sum_{n=1}^N \log p(\mathbf{x}_n|\boldsymbol{\theta}) \right] - D_{\text{KL}}(q(\boldsymbol{\theta}|\psi_{\boldsymbol{\theta}})\|p(\boldsymbol{\theta})) \quad (10.151)$$

45 We can approximate the first term using minibatching. Since $q(\boldsymbol{\theta}|\psi_{\boldsymbol{\theta}})$ 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

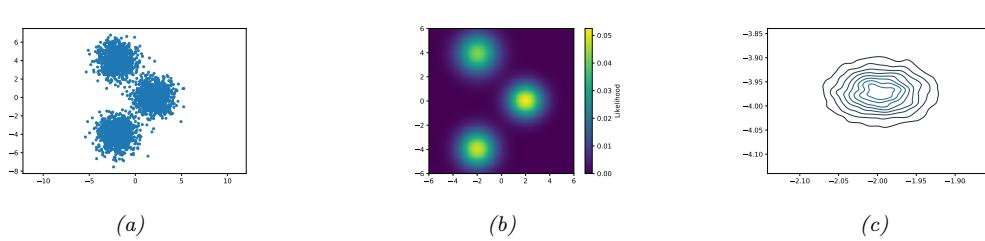


Figure 10.15: SVI for fitting a mixture of 3 Gaussians in 2d. (a) 3000 training points. (b) Fitted density, plugging in the posterior mean parameters. (c) Kernel density estimate fit to 10,000 samples from $q(\mu_1|\psi_\theta)$. Generated by `svi_gmm_demo_2d_tfp.py`.

per minibatch, $\theta^s \sim q(\theta|\psi_\theta)$, we get

$$\nabla_{\psi_\theta} \hat{L}(\psi_\theta | \mathcal{D}) \approx \frac{N}{B} \sum_{b=1}^B \nabla_{\psi_\theta} \log p(\mathbf{x}_b | \theta^s) - \nabla_{\psi_\theta} D_{\text{KL}}(q(\theta|\psi_\theta) \| p(\theta)) \quad (10.152)$$

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

$$q(z_n|\psi_n) = q(z_n|f_\phi^{\text{inf}}(\mathbf{x}_n)) = q_\phi(z_n|\mathbf{x}_n) \quad (10.154)$$

1 The “**amortized ELBO**”, for a model with local latents and fixed global parameters, becomes
2

3

$$\tilde{L}(\phi, \theta | \mathcal{D}) = \frac{1}{N} \sum_{n=1}^N [\mathbb{E}_{q_\phi(z_n | x_n)} [\log p_\theta(x_n, z_n) - \log q_\phi(z | x_n)]] \quad (10.155)$$

4

5 We can approximate this by sampling a single data point $x_n \sim p_{\mathcal{D}}$, and then sampling a single latent
6 $z_n \sim q_\phi(z_n | x_n)$, as in DSVI, to get

7

$$\tilde{L}(\phi, \theta | x_n, z_n) = \log p_\theta(x_n, z_n^s) - \log q_\phi(z_n) \quad (10.156)$$

8

9 (We call this the “**per-sample ELBO**”, although [Ble17] call it the **instantaneous ELBO**.) If the
10 posteriors are reparameterizable, we can push gradients inside and then apply SGD. See Algorithm 10
11 for the resulting pseudocode.

12 **Algorithm 10:** Amortized SVI

13 1 Initialize θ, ϕ
14 2 **repeat**
15 3 Sample $x_n \sim p_{\mathcal{D}}$
16 4 Sample $z_n \sim q_\phi(z | x_n)$
17 5 $\theta := \theta + \eta \nabla_\theta \tilde{L}(\phi, \theta | x_n, z_n)$
18 6 $\phi := \phi + \eta \nabla_\phi \tilde{L}(\phi, \theta | x_n, z_n)$
19 7 Update learning rate η
20 8 **until** converged;

21 This method is very widely used for fitting LVMs, e.g., for VAEs (see Section 22.2), for topic
22 models [SS17a], for probabilistic programming [RHG16a], for CRFs [TG18], etc. However, the use
23 of an inference network can result in a suboptimal setting of the local variational parameters ψ_n .
24 This is called the **amortization gap** [CLD18]. We can close this gap by using the inference network
25 to warm-start an optimizer for ψ_n ; this is known as **semi-amortized VI** [Kim+18c]. The key
26 insight is that the local SVI procedure is itself differentiable, so the inference network and generative
27 model can be trained end-to-end. (See also [MYM18], who propose a closely related method called
28 **iterative amortized inference**.)

29 An alternative approach is to use the inference network as a proposal distribution. If we combine
30 this with importance sampling, we get the IWAE bound of Section 10.5.1. If we use this with
31 Metropolis Hastings, we get a VI-MCMC hybrid (see Section 10.4.5).

32

33 10.3.8 Exploiting partial conjugacy

34 If the full conditionals of the joint model are conjugate distributions, we can use the VMP approach
35 of Section 10.2.7 to approximate the posterior one term at a time, similar to Gibbs sampling
36 (Section 12.3). However, in many models, some parts of the joint distribution are conjugate, and some
37 are non-conjugate. In [KL17a] they proposed the **conjugate-computation variational inference**
38 or **CVI** method to tackle models of this form. They exploit the partial conjugacy to perform some
39 updates in closed form, and perform the remaining updates using stochastic approximations.

40

1 To explain the method in more detail, let us assume the joint distribution has the form
2

$$\underline{3} \quad p(\mathbf{y}, \mathbf{z}) \propto \tilde{p}_{nc}(\mathbf{y}, \mathbf{z}) \tilde{p}_c(\mathbf{y}, \mathbf{z}) \quad (10.157)$$

4
5 where \mathbf{z} are all the latents (global or local), \mathbf{y} are all the observables (data)⁵, p_c is the conjugate
6 part, p_{nc} is the non-conjugate part, and the tilde symbols indicate that these distributions may not
7 be normalized wrt \mathbf{z} . More precisely, we assume the conjugate part is an exponential family model
8 of the following form:
9

$$\underline{10} \quad \tilde{p}_c(\mathbf{y}, \mathbf{z}) = h(\mathbf{z}) \exp[\mathcal{T}(\mathbf{z})^\top \boldsymbol{\eta} - A_c(\boldsymbol{\eta})] \quad (10.158)$$

11 where $\boldsymbol{\eta}$ is a *known* vector of natural parameters. (Any unknown model parameters should be
12 included in the latent state \mathbf{z} , as we illustrate below.) We also assume that the variational posterior
13 is an exponential family model with the same sufficient statistics, but different parameters:
14

$$\underline{15} \quad q(\mathbf{z}|\boldsymbol{\lambda}) = h(\mathbf{z}) \exp[\mathcal{T}(\mathbf{z})^\top \boldsymbol{\eta} - A(\boldsymbol{\lambda})] \quad (10.159)$$

16 The mean parameters are given by $\boldsymbol{\mu} = \mathbb{E}_q [\mathcal{T}(\mathbf{z})]$. We assume the sufficient statistics are minimal,
17 so that there is a unique 1:1 mapping between $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$: using
18

$$\underline{19} \quad \boldsymbol{\mu} = \nabla_{\boldsymbol{\lambda}} A(\boldsymbol{\lambda}) \quad (10.160)$$

$$\underline{20} \quad \boldsymbol{\lambda} = \nabla_{\boldsymbol{\mu}} A^*(\boldsymbol{\mu}) \quad (10.161)$$

21 where A^* is the conjugate of A (see Section 2.5.4). The ELBO is given by
22

$$\underline{23} \quad \underline{L}(\boldsymbol{\lambda}) = \mathbb{E}_q [\log p(\mathbf{y}, \mathbf{z}) - \log q(\mathbf{z}|\boldsymbol{\lambda})] \quad (10.162)$$

$$\underline{24} \quad \underline{L}(\boldsymbol{\mu}) = \underline{L}(\boldsymbol{\lambda}(\boldsymbol{\mu})) \quad (10.163)$$

25 The simplest way to fit this variational posterior is to perform SGD on the ELBO wrt the natural
26 parameters:
27

$$\underline{28} \quad \boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t + \eta_t \nabla_{\boldsymbol{\lambda}} \underline{L}(\boldsymbol{\lambda}_t) \quad (10.164)$$

29 (Note the + sign in front of the gradient, since we are maximizing the ELBO.) The above gradient
30 update is equivalent to solving the following optimization problem:
31

$$\underline{32} \quad \boldsymbol{\lambda}_{t+1} = \underset{\boldsymbol{\lambda} \in \Omega}{\operatorname{argmin}} \underbrace{(\nabla_{\boldsymbol{\lambda}} \underline{L}(\boldsymbol{\lambda}_t))^\top \boldsymbol{\lambda} - \frac{1}{2\eta_t} \|\boldsymbol{\lambda} - \boldsymbol{\lambda}_t\|_2^2}_{J(\boldsymbol{\lambda})} \quad (10.165)$$

33 where Ω is the space of valid natural parameters, and $\|\cdot\|_2$ is the Euclidean norm. To see this, note
34 that the first order optimality conditions satisfy
35

$$\underline{36} \quad \nabla_{\boldsymbol{\lambda}} J(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\lambda}} \underline{L}(\boldsymbol{\lambda}_t) - \frac{1}{2\eta_t} (2\boldsymbol{\lambda} - 2\boldsymbol{\lambda}_t) = \mathbf{0} \quad (10.166)$$

37 5. We denote observables by \mathbf{y} since the examples we consider later on are conditional models, where \mathbf{x} denote the
38 inputs.
39

¹ from which we get Equation (10.164).
²

³ We can replace the Euclidean distance with a more general proximity function, such as the Bregman
⁴ divergence between the distributions (see Section 6.5.1). This gives rise to the **mirror descent**
⁵ algorithm (Section 6.5). We can also perform updates in the mean parameter space. In [RM15a],
⁶ they show that this is equivalent to performing natural gradient updates in the natural parameter
⁷ space. Thus this method is sometimes called **natural gradient VI** or **NGVI**. Combining these two
⁸ steps gives the following update equation:

$$\underset{\boldsymbol{\mu} \in \mathcal{M}}{\operatorname{argmin}} (\nabla_{\boldsymbol{\mu}} \mathbb{L}(\boldsymbol{\mu}_t))^T \boldsymbol{\mu} - \frac{1}{\eta_t} B_{A^*}(\boldsymbol{\mu} || \boldsymbol{\mu}_t) \quad (10.167)$$

⁹
¹⁰ where \mathcal{M} is the space of valid mean parameters and $\eta_t > 0$ is a stepsize.
¹¹

¹² In [KL17a], they show that the above update is equivalent to performing exact Bayesian inference
¹³ in the following conjugate model:
¹⁴

$$q(\mathbf{z} | \boldsymbol{\lambda}_{t+1}) \propto e^{\mathcal{T}(\mathbf{z})^T \tilde{\boldsymbol{\lambda}}_t} \tilde{p}_c(\mathbf{y}, \mathbf{z}) \quad (10.168)$$

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 $\tilde{\boldsymbol{\lambda}}_t$. (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

$$\tilde{\boldsymbol{\lambda}}_t = (1 - \eta_t) \tilde{\boldsymbol{\lambda}}_{t-1} + \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q(\mathbf{z} | \boldsymbol{\mu}_t)} [\log \tilde{p}_{nc}(\mathbf{y}, \mathbf{z})] \quad (10.169)$$

²⁰ where $\tilde{\boldsymbol{\lambda}}_0 = \mathbf{0}$ and $\tilde{\boldsymbol{\lambda}}_1 = \boldsymbol{\eta}$. (Details on how to compute this derivative are given in Section 6.4.5.)
²¹ Once we can have “conjugated” the non-conjugate part, the natural parameter of the new variational
²² posterior is obtained by
²³

$$\boldsymbol{\lambda}_{t+1} = \tilde{\boldsymbol{\lambda}}_t + \boldsymbol{\eta} \quad (10.170)$$

²⁴ This corresponds to a multiplicative update of the form
²⁵

$$q_{t+1}(\mathbf{z}) \propto q_t(\mathbf{z})^{1-\eta_t} \left[\exp(\tilde{\boldsymbol{\lambda}}_t^T \mathcal{T}(\mathbf{z})) \right]^{\eta_t} \quad (10.171)$$

²⁶ We give some examples of this below.
²⁷

²⁸ 10.3.8.1 Example: Gaussian process with non-Gaussian likelihoods

²⁹ In Chapter 18, we discuss Gaussian processes, which are a popular model for non-parametric
³⁰ regression. Given a set of N inputs $\mathbf{x}_n \in \mathcal{X}$ and outputs $y_n \in \mathbb{R}$, we define the following joint
³¹ Gaussian distribution:
³²

$$p(\mathbf{y}_{1:N}, \mathbf{z}_{1:N} | \mathbf{X}) = \left[\prod_{n=1}^N \mathcal{N}(y_n | z_n, \sigma^2) \right] \mathcal{N}(\mathbf{z} | \mathbf{0}, \mathbf{K}) \quad (10.172)$$

³³ where \mathbf{K} is the kernel matrix computed using $K_{ij} = \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$, and $z_n = f(\mathbf{x}_n)$ is the unknown
³⁴ function value for input n . Since this model is jointly Gaussian, we can easily compute the exact
³⁵ posterior $p(\mathbf{z} | \mathbf{y})$ in $O(N^3)$ time. (Faster approximations are also possible, see Section 18.5.)
³⁶

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

$$\begin{aligned} \frac{12}{13} p(\mathbf{y}_{1:N}, \mathbf{w}|\mathbf{X}) &= \left[\prod_{n=1}^N p(y_n|z_n) \right] \mathcal{N}(\mathbf{w}|\mathbf{0}, \delta\mathbf{I}) \end{aligned} \tag{10.183}$$

¹⁴

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 Section 10.3.8.1. We then have the
¹⁷ following variational joint:

¹⁹

$$\begin{aligned} \frac{20}{21} q(\mathbf{w}|\boldsymbol{\lambda}_{t+1}) &\propto \left[\prod_{n=1}^N \mathcal{N}(\tilde{m}_{n,t}|\mathbf{w}^\top \mathbf{x}_n, \tilde{v}_{n,t}) \right] \mathcal{N}(\mathbf{w}|\mathbf{0}, \delta\mathbf{I}) \end{aligned} \tag{10.184}$$

²²

²³ This corresponds to a Bayesian linear regression problem with pseudo-observations $\tilde{m}_{n,t}$ and variational
²⁴ 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).

³² **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

$$\begin{aligned} \frac{39}{40} \psi_t &= \underset{\psi}{\operatorname{argmin}} \underbrace{\mathbb{E}_{q(\boldsymbol{\theta}|\psi)} [\ell_t(\boldsymbol{\theta})] + D_{\text{KL}}(q(\boldsymbol{\theta}|\psi)\|q(\boldsymbol{\theta}|\psi_{t-1}))}_{-\mathcal{L}_t(\psi)} \end{aligned} \tag{10.185}$$

⁴¹

$$\begin{aligned} \frac{42}{43} &= \underset{\psi}{\operatorname{argmin}} \mathbb{E}_{q(\boldsymbol{\theta}|\psi)} [\ell_t(\boldsymbol{\theta}) + \log q(\boldsymbol{\theta}|\psi) - \log q(\boldsymbol{\theta}|\psi_{t-1})] \end{aligned} \tag{10.186}$$

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.)

10.3.9.1 FOO-VB

In this section, we discuss a particular implementation of sequential VI called **FOO-VB**, which stands for “Fixed-point Operator for Online Variational Bayes” [Zen+21]. This assumes Gaussian priors and posteriors. In particular, let

$$q(\boldsymbol{\theta}|\psi_t) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma}), \quad q(\boldsymbol{\theta}|\psi_{t-1}) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{m}, \mathbf{V}) \quad (10.187)$$

In this case, we can write the ELBO as follows:

$$\mathbb{L}_t(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{2} \left[\log \frac{\det(\mathbf{V})}{\det(\boldsymbol{\Sigma})} - D + \text{tr}(\mathbf{V}^{-1}\boldsymbol{\Sigma}) + (\mathbf{m} - \boldsymbol{\mu})^\top \mathbf{V}^{-1}(\mathbf{m} - \boldsymbol{\mu}) \right] + \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})} [\ell_t(\boldsymbol{\theta})] \quad (10.188)$$

where D is the dimensionality of $\boldsymbol{\theta}$.

Let $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^\top$. We can compute the new variational parameters by solving the joint first order 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 use the identities

$$\frac{\partial \text{tr}(\mathbf{V}^{-1}\boldsymbol{\Sigma})}{\partial L_{ij}} = 2 \sum_n V_{in}^{-1} L_{nj} \quad (10.189)$$

$$\frac{\partial \log |\det(\mathbf{L})|}{\partial L_{ij}} = L_{ij}^{-T} \quad (10.190)$$

For the derivatives of the expected loss, we use the the reparameterization trick, $\boldsymbol{\theta} = \boldsymbol{\mu} + \mathbf{L}\boldsymbol{\epsilon}$, and following identities:

$$\mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\mu}, \mathbf{L})} [\ell_t(\boldsymbol{\theta})] = \mathbb{E}_{\boldsymbol{\epsilon}} [\ell_t(\boldsymbol{\theta}(\boldsymbol{\epsilon}))] \quad (10.191)$$

$$\frac{\partial \mathbb{E}_{\boldsymbol{\epsilon}} [\ell_t(\boldsymbol{\theta})]}{\partial L_{ij}} = \mathbb{E}_{\boldsymbol{\epsilon}} \left[\frac{\partial \ell_t(\boldsymbol{\theta})}{\partial \theta_i} \epsilon_j \right] \quad (10.192)$$

Note that the expectation depends on the unknown variational parameters for q_t , so we get a fixed point equation which we need to iterate. As a faster alternative, [Zen+18; Zen+21] propose to evaluate the expectations using the variational parameters from the previous step, which then gives the new parameters in a single step, similar to EM.

We now derive the update equations. From $\nabla_{\boldsymbol{\mu}} \mathbb{L}_t(\boldsymbol{\mu}, \mathbf{L}) = \mathbf{0}$ we get

$$\mathbf{0} = -\mathbf{V}^{-1}(\mathbf{m} - \boldsymbol{\mu}) + \mathbb{E}_{\boldsymbol{\epsilon}} [\nabla \ell_t(\boldsymbol{\theta})] \quad (10.193)$$

$$\boldsymbol{\mu} = \mathbf{m} - \mathbf{V} \mathbb{E}_{\boldsymbol{\epsilon}} [\nabla \ell_t(\boldsymbol{\theta})] \quad (10.194)$$

From $\nabla_{\mathbf{L}} \mathbb{L}_t(\boldsymbol{\mu}, \mathbf{L}) = \mathbf{0}$. we get

$$0 = -(L^{-T})_{ij} + \sum_n V_{in}^{-1} L_{nj} + \mathbb{E}_{\boldsymbol{\epsilon}} \left[\frac{\partial \ell_t(\boldsymbol{\theta})}{\partial \theta_i} \epsilon_j \right] \quad (10.195)$$

In matrix form, we have

$$\mathbf{0} = -\mathbf{L}^{-T} + \mathbf{V}^{-1}\mathbf{L} + \mathbb{E}_{\boldsymbol{\epsilon}} [\nabla \ell_t(\boldsymbol{\theta}) \boldsymbol{\epsilon}^T] \quad (10.196)$$

Explicitly solving for \mathbf{L} in the case of a general (or low rank) matrix $\boldsymbol{\Sigma}$ is somewhat complicated; for the details, see [Zen+21]. Fortunately, in the case of a diagonal appproximation, things simplify significantly, as we discuss in Section 10.3.9.2.

10.3.9.2 Bayesian gradient descent

Let $\mathbf{V} = \text{diag}(v_i^2)$, $\Sigma = \text{diag}(\sigma_i^2)$, so $\mathbf{L} = \text{diag}(\sigma_i)$. Also, let $g_i = \frac{\partial \ell_t(\theta)}{\partial \theta_i}$, which depends on ϵ_i . Then Equation (10.194) becomes

$$\mu_i = m_i - \eta v_i^2 \mathbb{E}_{\epsilon_i} [g_i(\epsilon_i)] \quad (10.197)$$

where we have included an explicit learning rate η to compensate for the fact that the fixed point equation update is approximate. For the variance terms, Equation (10.196) becomes

$$0 = -\frac{1}{\text{diag}(\sigma_i)} + \frac{\text{diag}(\sigma_i)}{\text{diag}(v_i^2)} + \mathbb{E}_{\epsilon_i} [g_i \epsilon_i] \quad (10.198)$$

This is a quadratic equation for each σ_i :

$$\frac{1}{v_i^2} \sigma_i^2 + \mathbb{E}_{\epsilon_i} [g_i \epsilon_i] \sigma_i - 1 = 0 \quad (10.199)$$

the solution of which is given by the following (since $\sigma_i > 0$):

$$\sigma_i = \frac{-\mathbb{E}_{\epsilon_i} [g_i \epsilon_i] + \sqrt{(\mathbb{E}_{\epsilon_i} [g_i \epsilon_i])^2 + 4/v_i^2}}{2/v_i^2} = -\frac{1}{2} v_i^2 \mathbb{E}_{\epsilon_i} [g_i \epsilon_i] + \frac{1}{2} v_i^2 \sqrt{(\mathbb{E}_{\epsilon_i} [g_i \epsilon_i])^2 + 4/v_i^2} \quad (10.200)$$

$$= -\frac{1}{2} v_i^2 \mathbb{E}_{\epsilon_i} [g_i \epsilon_i] + \sqrt{\frac{v_i^4}{4} ((\mathbb{E}_{\epsilon_i} [g_i \epsilon_i])^2 + 4/v_i^2)} = -\frac{1}{2} v_i^2 \mathbb{E}_{\epsilon_i} [g_i \epsilon_i] + v_i \sqrt{1 + (\frac{1}{2} v_i \mathbb{E}_{\epsilon_i} [g_i \epsilon_i])^2} \quad (10.201)$$

We can approximate the above expectations using K Monte Carlo samples. Thus the overall algorithm is very similar to standard SGD, except we compute the gradient K times, and we update $\mu \in \mathbb{R}^D$ and $\sigma \in \mathbb{R}^D$ rather than $\theta \in \mathbb{R}^D$. In [Zen+18], they call the resulting algorithm “**Bayesian gradient descent**”, and they show it works well on some continual learning problems (see Section 20.7). See Algorithm 11 for the pseudocode, and see [Kur+20] for a related algorithm.

10.3.9.3 Generalized variational continual learning

One problem with the VCL objective in Equation (10.185) is that the KL term can cause the model to become too sparse, which can prevent the model from adapting or learning new tasks. This problem is called **variational overpruning** [TT17]. More precisely, the reason this happens as follows: some weights might not be needed to fit a given dataset, so their posterior will be equal to the prior; but sampling from these high-variance weights will add noise to the likelihood; to reduce this, the optimization method will prefer to set the bias term to a large negative value, so the corresponding unit is “turned off”, and thus has no effect on the likelihood. Unfortunately, these “dead units” become stuck, so there is not enough network capacity to learn the next task, as shown in Figure 20.19(b).

In [LST21], they propose a solution to this, known as **generalized variational continual learning**. The first step is to downweight the KL term by a factor $\beta < 1$ to get

$$\mathcal{L}_t = \mathbb{E}_{q(\theta|\psi)} [\ell_t(\theta)] + \beta D_{\text{KL}} (q(\theta|\psi) \| q(\theta|\psi_{t-1})) \quad (10.202)$$

Algorithm 11: One step of Bayesian gradient descent

```

1 Function  $(\mu_t, \sigma_t, \hat{L}_t) = \text{BGD-update}(\mu_{t-1}, \sigma_{t-1}, \mathcal{D}_t; \eta, K)$ :
2   for  $k = 1 : K$  do
3     Sample  $\epsilon^k \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
4      $\theta^k = \mu_{t-1} + \sigma_{t-1} \odot \epsilon^k$ 
5      $\mathbf{g}^k = \nabla_{\theta} - \log p(\mathcal{D}_t | \theta) |_{\theta^k}$ 
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     $\theta^k = \mu_t + \sigma_t \odot \epsilon^k$ 
13     $\ell_t^k = -\log p(\mathcal{D}_t | \theta^k)$ 
14   $\hat{L}_t = - \left[ \frac{1}{K} \sum_{k=1}^K \ell_t^k + D_{\text{KL}}(\mathcal{N}(\mu_t, \sigma_t) \| \mathcal{N}(\mu_{t-1}, \sigma_{t-1})) \right]$ 

```

²⁴ Interestingly, one can show that in the limit of $\beta \rightarrow 0$, this recovers several standard methods
²⁵ that use a Laplace approximation, based on the Hessian rather than the posterior covariance. In
²⁶ particular if Q is diagonal, this reduces to **Online elastic weight consolidation** [Sch+18]; if Q
²⁷ is block-diagonal and Kronecker factored, this reduces to **Online structured Laplace** [RBB18b];
²⁸ 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

$$\text{Eq. 10.203} \quad L_t = \mathbb{E}_{q(\theta|\psi)} [\ell_t(\theta)] + \beta D_{\text{KL}}(q(\theta|\psi)^\lambda \| q(\theta|\psi_{t-1})^\lambda)$$

³⁷ Using a value of $\lambda \gg 1$ is equivalent to using a “cold posterior”, which we discuss in Section 17.3.

40 10.4 More accurate variational posteriors

42 In general, we can improve the tightness of the ELBO lower bound, and hence reduce the KL
43 divergence of our posterior approximation, if we use more flexible posterior families (although
44 optimizing within more flexible families may be slower, and can incur statistical error if the sample
45 size is low [Bha+21]). In this section, we give several examples of more accurate variational posteriors,
46 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.,

$$\text{29 } q_\phi(\mathbf{z}|\mathbf{x}) = \int q_\phi(\mathbf{z}|\mathbf{x}, \mathbf{a}) q_\phi(\mathbf{a}|\mathbf{x}) d\mathbf{a} \neq \prod_k q_\phi(z_k|\mathbf{x}) \quad (10.204)$$

31 This is called a **hierarchical variational model** [Ran16], or an **auxiliary variable deep gener-
32 ative model** [Maa+16].

34 In [TRB16], they model $q_\phi(\mathbf{z}|\mathbf{x}, \mathbf{a})$ as a Gaussian process, which is a flexible nonparametric
35 distribution (see Chapter 18), where \mathbf{a} are the inducing points. This combination is called a
36 **variational GP**.

38 10.4.3 Normalizing flow posteriors

40 Normalizing flows are a class of probability models which work by passing a simple source distribution,
41 such as a diagonal Gaussian, through a series of nonlinear, but invertible, mappings f to create
42 a more complex distribution final distribution. This can be used to get more accurate posterior
43 approximations, as we discuss in Section 24.1.3.2.

44 For example, an autoregressive flow for a sequence model $p(\mathbf{y})$ has the form

$$\text{46 } y_i = \mu_i(\mathbf{y}_{1:i-1}) + \sigma_i(\mathbf{y}_{1:i-1}) \epsilon_i \quad (10.205)$$

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

$$q(\boldsymbol{\epsilon}_T) = q(\boldsymbol{\epsilon}_0) \left| \det \frac{\partial \boldsymbol{\epsilon}_T}{\partial \boldsymbol{\epsilon}_0} \right|^{-1} \quad (10.211)$$

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:

$$(\mathbf{m}_t, \mathbf{s}_t) = \text{AutoRegressiveNN}_t(\boldsymbol{\epsilon}_{t-1}, \mathbf{h}; \phi) \quad (10.214)$$

$$\boldsymbol{\sigma}_t = \boldsymbol{\sigma}(\mathbf{s}_t) \quad (10.215)$$

$$\boldsymbol{\epsilon}_t = \boldsymbol{\sigma}_t \odot \boldsymbol{\epsilon}_{t-1} + (1 - \boldsymbol{\sigma}_t) \odot \mathbf{m}_t \quad (10.216)$$

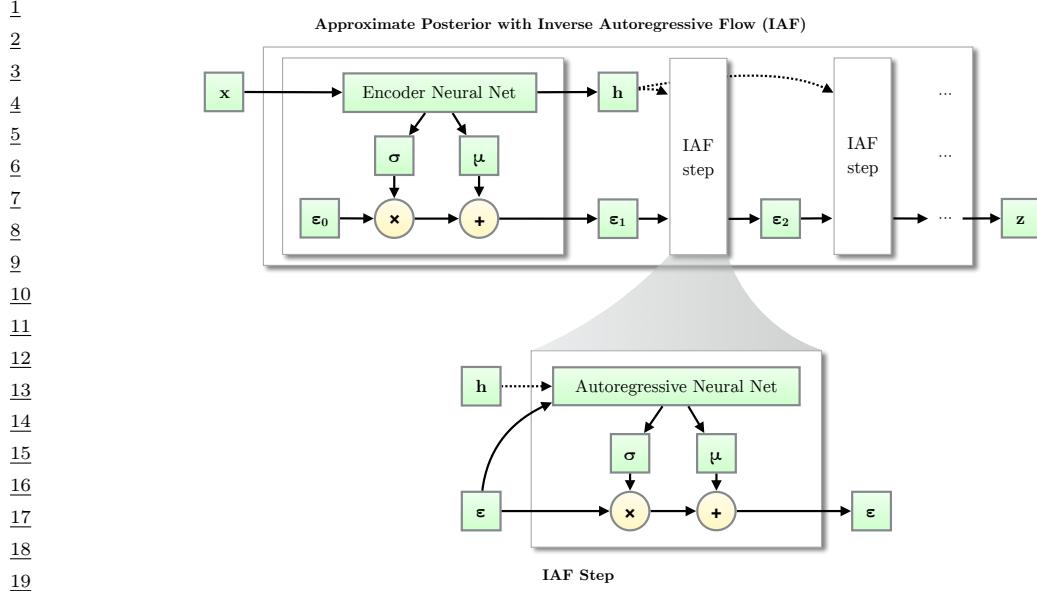


Figure 10.16: Using an inverse autoregressive flow network to approximate the posterior of a VAE. From Figure 3.1 of [KW19a]. Used with kind permission of Durk Kingma.

where $\boldsymbol{\sigma}(\mathbf{s}_t) \in [0, 1]^K$ is a vector of gating terms. At the end we set $\mathbf{z} = \epsilon_T$. 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.

¹ **10.4.5 Combining VI with MCMC inference**

² There are various ways to combine variational inference with MCMC to get an improved approximate
³ posterior. In [SKW15], they propose **Hamiltonian Variational Inference**, in which they train an
⁴ inference network to initialize an HMC sampler (Section 12.5). The gradient of the log posterior (wrt
⁵ the latents), which is needed by HMC, is given by
⁶

⁷
$$\nabla_{\mathbf{z}} \log p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = \nabla_{\mathbf{z}} \log [p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log p_{\boldsymbol{\theta}}(\mathbf{x})] = \nabla_{\mathbf{z}} \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) \quad (10.218)$$

⁸

⁹ This is easy to compute. They use the final sample to approximate the posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$. To compute
¹⁰ the entropy of this distribution, they also learn an auxiliary inverse inference network to reverse the
¹¹ HMC Markov chain.
¹²

¹³ A simpler approach is proposed in [Hof17]. Here they train an inference network to initialize an
¹⁴ HMC sampler, using the standard ELBO for $\boldsymbol{\phi}$, but they optimize the generative parameters $\boldsymbol{\theta}$ using
¹⁵ a stochastic approximation to the log marginal likelihood, given by $\log p_{\boldsymbol{\theta}}(\mathbf{z}, \mathbf{x})$ where \mathbf{z} is a sample
¹⁶ from the HMC chain. This does not require learning a reverse inference network, and avoids problems
¹⁷ with variational pruning, since it does not use the ELBO for training the generative model.
¹⁸

¹⁹ **10.5 Lower bounds**

²⁰ An alternative way to improve the quality of the posterior approximation is to optimize q wrt a bound
²¹ that is a tighter approximation to the log marginal likelihood compared to the standard ELBO.
²²

²³ **10.5.1 Multi-sample ELBO (IWAE bound)**

²⁴ In this section, we discuss a method known as the **importance weighted autoencoder** or **IWAE**
²⁵ [BGS16], which is a way to tighten the variational lower bound by using self-normalized importance
²⁶ sampling (Section 11.5.2). (It can also be interpreted as standard ELBO maximization in an expanded
²⁷ model, where we add extra auxiliary variables [CMD17; DS18; Tuc+19].)
²⁸

²⁹ Let the inference network $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ be viewed as a proposal distribution for the target posterior
³⁰ $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. Define $w_s^* = \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}_s)}{q_{\boldsymbol{\phi}}(\mathbf{z}_s|\mathbf{x})}$ as the unnormalized importance weight for a sample, and $w_s =$
³¹ $w_s^*/(\sum_{s'=1}^S w_s^*)$ as the normalized importance weights. From Equation (11.43) we can compute an
³² estimate of the marginal likelihood $p(\mathbf{x})$ using
³³

³⁴
$$\hat{p}_S(\mathbf{x}|\mathbf{z}_{1:S}) \triangleq \frac{1}{S} \sum_{k=1}^S \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}_s)}{q_{\boldsymbol{\phi}}(\mathbf{z}_s|\mathbf{x})} = \frac{1}{S} \sum_{k=1}^S w_s \quad (10.219)$$

³⁵

³⁶ This is unbiased, i.e. $\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}_{1:S}|\mathbf{x})} [\hat{p}_S(\mathbf{x}|\mathbf{z}_{1:S})] = p(\mathbf{x})$, where $q_{\boldsymbol{\phi}}(\mathbf{z}_{1:S}|\mathbf{x}) = \prod_{k=1}^S q_{\boldsymbol{\phi}}(\mathbf{z}_s|\mathbf{x})$. In addition,
³⁷ since the estimator is always positive, we can take logarithms, and thus obtain a stochastic lower
³⁸ bound on the log likelihood:
³⁹

⁴⁰
$$L_S(\boldsymbol{\phi}, \boldsymbol{\theta}|\mathbf{x}) \triangleq \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}_{1:S}|\mathbf{x})} \left[\log \left(\frac{1}{S} \sum_{s=1}^S w_s \right) \right] = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}_{1:S}|\mathbf{x})} [\log \hat{p}_S(\mathbf{z}_{1:S})] \quad (10.220)$$

⁴¹

⁴²
$$\leq \log \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}_{1:S}|\mathbf{x})} [\hat{p}_S(\mathbf{z}_{1:S})] = \log p(\mathbf{x}) \quad (10.221)$$

⁴³

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 One solution to this is to use the **doubly reparameterized gradient estimator** [TL18b].
 15 Another approach is to use alternative estimation methods that avoid ELBO maximization, such
 16 as using the thermodynamic variational objective (see Section 10.5.2) or the reweighted wake sleep
 17 algorithm (see Section 22.7).
 18

10.5.2 The thermodynamic variational objective (TVO)

19 In [MLW19; Bre+20b], they present the **thermodynamic variational objective** or **TVO**. This
 20 is an alternative to IWAE for creating tighter variational bounds, which has certain advantages,
 21 particularly for posteriors that are not reparameterizable (e.g., discrete latent variables). The
 22 framework also has close connections with the reweighted wake sleep algorithm from Section 22.7, as
 23 we will see in Section 10.6.2.

24 The TVO technique uses **thermodynamic integration**, also called **path sampling**, which is
 25 a technique used in physics and phylogenetics to approximate intractable normalization constants
 26 of high dimensional distributions (see e.g., [GM98; LP06; FP08]). This is based on the insight
 27 that it is easier to calculate the ratio of two unknown constants than to calculate the constants
 28 themselves. This is similar to the idea behind annealed importance sampling (Section 11.5.4), but TI
 29 is deterministic. For details, see [MLW19; Bre+20b].
 30

10.6 Upper bounds

31 Classical variational inference minimizes $D_{\text{KL}}(q \| p)$, which maximizes a lower bound on $\log Z =$
 32 $\log p(\mathbf{x})$. The reverse KL is zero-forcing (mode-seeking), and can cause problems with overconfidence
 33 in the variational posterior. One possible solution to this is to minimize the forwards KL, $D_{\text{KL}}(p \| q)$,
 34 which is zero avoiding (mode-covering). However, it is intractable to compute this objective, since
 35 it requires taking expectations wrt the true posterior p . The expectation propagation algorithm
 36 (Section 10.7) tackles this by optimizing the KL locally, one term at a time. Unfortunately EP is not
 37 optimizing an overall bound, and may not converge.
 38

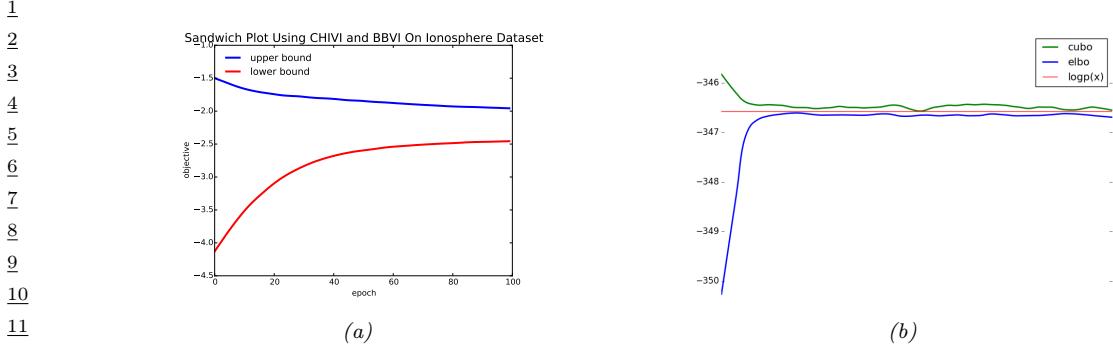


Figure 10.17: Illustration of the CUBO upper bound being minimized by CHIVI (Section 10.6.1) and the ELBO lower bound being maximized by BBVI (Section 10.3.1) on UCI Ionosphere dataset and a synthetic dataset where we know the true value of $\log p(\mathbf{x})$. From Figure 1 of [Die+17]. Used with kind 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

$$\chi^2(p\|q) \triangleq \mathbb{E}_{q(\mathbf{z}|\psi)} \left[\left(\frac{p(\mathbf{z}|\mathbf{x})}{q(\mathbf{z}|\psi)} \right)^2 - 1 \right] \quad (10.223)$$

Minimizing this is known as **χ -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

$$\mathbb{E}_{q(\mathbf{z}|\psi)} \left[\frac{p(\mathbf{x}, \mathbf{z})^2}{q(\mathbf{z}|\psi)^2} \right] = \mathbb{E}_{q(\mathbf{z}|\psi)} \left[\frac{p(\mathbf{z}|\mathbf{x})^2 p(\mathbf{x})^2}{q(\mathbf{z}|\psi)^2} \right] = p(\mathbf{x})^2 \left[1 + \chi^2(p(\mathbf{z}|\mathbf{x})\|q(\mathbf{z}|\psi)) \right] \quad (10.224)$$

so

$$\frac{1}{2} \log \left[1 + \chi^2(p(\mathbf{z}|\mathbf{x})\|q(\mathbf{z}|\psi)) \right] = -\log p(\mathbf{x}) + \frac{1}{2} \log \mathbb{E}_{q(\mathbf{z}|\psi)} \left[\left(\frac{p(\mathbf{z}, \mathbf{x})}{q(\mathbf{z}|\psi)} \right)^2 \right] \quad (10.225)$$

Since $\log p(\mathbf{x})$ is constant, minimizing this is equivalent to minimizing the **χ -divergence upper**

1 **bound or CUBO:**

3 $\text{CUBO}(\psi) \triangleq \frac{1}{2} \log \mathbb{E}_{q(z|\psi)} \left[\left(\frac{p(z, x)}{q(z|\psi)} \right)^2 \right]$ (10.226)

4

5 One can show that $L \leq \log p(x) \leq \text{CUBO}$.

6 We can compute a Monte Carlo approximation to this upper bound using

7 $U(\psi) = \frac{1}{2} \log \frac{1}{S} \sum_{s=1}^S \left(\frac{p(z^s, x)}{q(z^s|\psi)} \right)^2$ (10.227)

8

9 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

10 Jensen's inequality). So let us consider a different upper bound, namely $V(\psi) = \exp(2 \text{CUBO}(\psi))$.

11 We can compute an unbiased MC approximation of this using

12 $V(\psi) = \frac{1}{S} \sum_{s=1}^S \left(\frac{p(z^s, x)}{q(z^s|\psi)} \right)^2$ (10.228)

13

14 We will assume $q(z)$ is reparameterizable, so we can sample ϵ^s and then compute $z^s = g(\epsilon^s, \psi)$,
15 where the distribution of ϵ^s is independent of ψ . Hence we can approximate the gradient at the
16 current parameter values using

17 $\nabla_\psi V(\psi) = \frac{1}{S} \sum_{s=1}^S \nabla \left(\frac{p(z^s, x)}{q(z^s|\psi)} \right)^2 = -\frac{2}{S} \sum_{s=1}^S (w^{(s)})^2 \nabla_\psi \log q(z^s|\psi)$ (10.229)

18

19 where $w^{(s)} = \frac{p(z^s)p(x|z^s)}{q(z^s|\psi_t)}$.

20 Unfortunately, the use of exponentiation in the definition of $V(\psi)$ makes this a very high variance
21 estimator [PHDV19].

22 10.6.2 Minimizing the evidence upper bound

23 Recall that the ELBO is given by

24 $L(\theta, \phi|x) = \log p_\theta(x) - D_{\text{KL}}(q_\phi(z|x) \| p_\theta(z|x)) \leq \log p_\theta(x)$ (10.230)

25 By analogy, we can define the **evidence upper bound** or **EUBO** as follows:

26 $\text{EUBO}(\theta, \phi|x) = \log p_\theta(x) + D_{\text{KL}}(p_\theta(z|x) \| q_\phi(z|x)) \geq \log p_\theta(x)$ (10.231)

27

28 If we sample x from the generative model, and minimize $\mathbb{E}_{p_\theta(x)} [\text{EUBO}(\theta, \phi|x)]$ wrt ϕ , we recover
29 the sleep phase of the wake-sleep algorithm (see Section 22.7.2), which in turn is equivalent to
30 inference compilation.

31 Now suppose we sample x from the empirical distribution, and minimize $\mathbb{E}_{p_D(x)} [\text{EUBO}(\theta, \phi|x)]$
32 wrt ϕ . To approximate the expectation, we can use self-normalized importance sampling, as in
33 Equation (22.153), to get

34

35 $\nabla_\phi \text{EUBO}(\theta, \phi|x) = \sum_{s=1}^S \bar{w}_s \nabla_\phi \log q_\phi(z^s|x)$ (10.232)

36

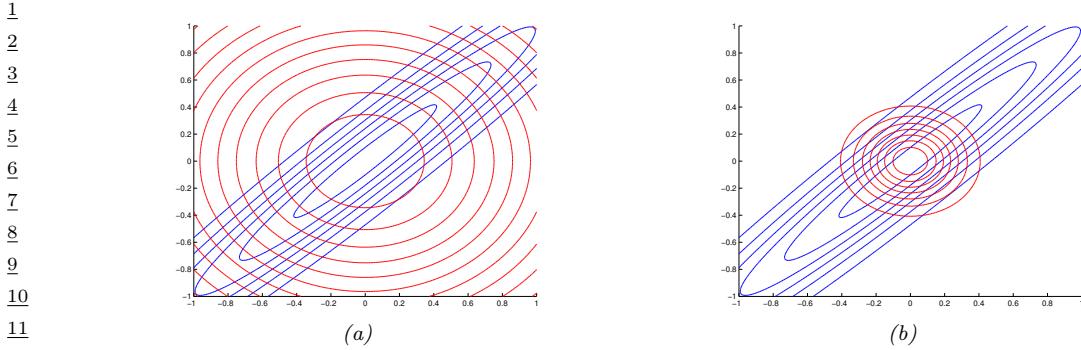


Figure 10.18: Illustrating forwards vs reverse KL on a symmetric Gaussian. The blue curves are the contours of the true distribution p . The red curves are the contours of a factorized approximation q . (a) Minimizing $D_{\text{KL}}(p\|q)$. (b) Minimizing $D_{\text{KL}}(q\|p)$. Adapted from Figure 10.2 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

$$\underline{q} = \operatorname{argmin}_q D_{\text{KL}}(p\|q) \quad (10.234)$$

4 This is called **M-projection**, or **moment projection**, since the optimal q matches the moments of
5 p (this is called **moment matching**). To see this, we assume q is an exponential family distribution,
6 and hence

7

$$q(\mathbf{x}) = h(\mathbf{x}) \exp[\boldsymbol{\eta}^\top \mathcal{T}(\mathbf{x}) - \log Z(\boldsymbol{\eta})] \quad (10.235)$$

8 where $\mathcal{T}(\mathbf{x})$ is the vector of sufficient statistics, and $\boldsymbol{\eta}$ are the natural parameters. The first order
9 optimality conditions are as follows:

10

$$\partial_{\eta_i} D_{\text{KL}}(p\|q) = -\partial_{\eta_i} \sum_{\mathbf{x}} p(\mathbf{x}) \log q(\mathbf{x}) \quad (10.236)$$

11

$$= -\partial_{\eta_i} \sum_{\mathbf{x}} p(\mathbf{x}) \log (h(\mathbf{x}) \exp[\boldsymbol{\eta}^\top \mathcal{T}(\mathbf{x}) - \log Z(\boldsymbol{\eta})]) \quad (10.237)$$

12

$$= -\partial_{\eta_i} \sum_{\mathbf{x}} p(\mathbf{x}) (\boldsymbol{\eta}^\top \mathcal{T}(\mathbf{x}) - \log Z(\boldsymbol{\eta})) \quad (10.238)$$

13

$$= -\sum_{\mathbf{x}} p(\mathbf{x}) \mathcal{T}_i(\mathbf{x}) + \mathbb{E}_{q(\mathbf{x})} [\mathcal{T}_i(\mathbf{x})] \quad (10.239)$$

14

$$= -\mathbb{E}_{p(\mathbf{x})} [\mathcal{T}_i(\mathbf{x})] + \mathbb{E}_{q(\mathbf{x})} [\mathcal{T}_i(\mathbf{x})] = 0 \quad (10.240)$$

15 where in the penultimate line we used the fact that the derivative of the log partition function yields
16 the expected sufficient statistics, as shown in Equation (2.177).

17 In this example, the optimal q must therefore have the following form:

18

$$q(\mathbf{x}) = \mathcal{N}(x_1|\mu_1, \Sigma_{11})\mathcal{N}(x_2|\mu_2, \Sigma_{22}) \quad (10.241)$$

19 where $\boldsymbol{\Sigma} = \boldsymbol{\Lambda}^{-1}$. In Figure 10.18(a), we show the resulting distribution. We see that q covers
20 (includes) p , but its support is too broad.

21

22 **10.7.1.2 Information projection**

23 Now suppose we compute q as follows:

24

$$\underline{q} = \operatorname{argmin}_q D_{\text{KL}}(q\|p) \quad (10.242)$$

25 This is called **I-projection**, or **information projection**.

26 In this example, one can show that the solution has the form

27

$$q(\mathbf{x}) = \mathcal{N}(x_1|m_1, \Lambda_{11}^{-1})\mathcal{N}(x_2|m_2, \Lambda_{22}^{-1}) \quad (10.243)$$

28

$$m_1 = \mu_1 - \Lambda_{11}^{-1} \Lambda_{12} (m_2 - \mu_2) \quad (10.244)$$

29

$$m_2 = \mu_2 - \Lambda_{22}^{-1} \Lambda_{21} (m_1 - \mu_1) \quad (10.245)$$

30 Unless $\boldsymbol{\Lambda}$ is singular, the solution must satisfy $m_i = \mu_i$, so we have again captured the mean exactly.

31 However, the posterior variance is in general too narrow as shown in Figure 10.18(b). (See [Tur+08]
32 for a discussion of this point.)

33

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: $q_{t+1} = \text{argmin}_{q \in \mathcal{Q}} D_{\text{KL}}(\hat{p}_{t+1} \| q)$.

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

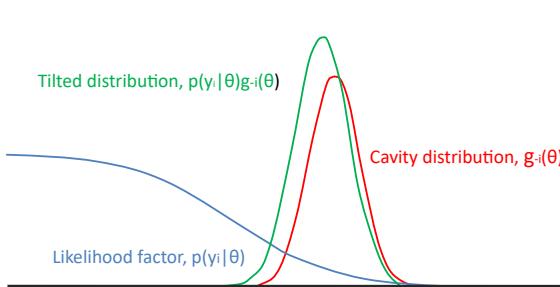


Figure 10.19: Combining a logistic likelihood factor $f_i = p(y_i|\theta)$ with the cavity prior, $q_{-i}^{cavity} = g_{-i}(\theta)$, to get the tilted distribution, $q_i^{tilted} = p(y_i|\theta)g_{-i}(\theta)$. Adapted from Figure 2 of [Gel+14b].

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.

¹ The idea behind **stochastic EP** [LHT15] is to approximate the local factors with a shared factor
² that acts like an aggregated likelihood, i.e.,
³

$$\prod_{n=1}^N f_n(\boldsymbol{\theta}) \approx \tilde{f}(\boldsymbol{\theta})^N \quad (10.255)$$

⁴
⁵ where typically $f_n(\boldsymbol{\theta}) = p(\mathbf{x}_n | \boldsymbol{\theta})$. This exploits the fact that the posterior only cares about approxi-
⁶ mating the product of the likelihoods, rather than each likelihood separately. Hence it suffices for
⁷ $\tilde{f}(\boldsymbol{\theta})$ to approximate the average likelihood.
⁸

⁹ We can modify EP to this setting as follows. First, when computing the cavity distribution, we
¹⁰ use
¹¹

$$q_{-1}(\boldsymbol{\theta}) \propto q(\boldsymbol{\theta}) / \tilde{f}(\boldsymbol{\theta}) \quad (10.256)$$

¹² We then compute the tilted distribution
¹³

$$q_{\setminus n}(\boldsymbol{\theta}) \propto f_n(\boldsymbol{\theta}) q_{-1}(\boldsymbol{\theta}) \quad (10.257)$$

¹⁴ Next we derive the new local factor for this datapoint using moment matching:
¹⁵

$$\tilde{f}_n(\boldsymbol{\theta}) = \text{proj}(q_{\setminus n}(\boldsymbol{\theta})) / q_{-1}(\boldsymbol{\theta}) \quad (10.258)$$

¹⁶ Finally we perform a damped update of the average likelihood $\tilde{f}(\boldsymbol{\theta})$ using this new local factor:
¹⁷

$$\tilde{f}_{\text{new}}(\boldsymbol{\theta}) = \tilde{f}_{\text{old}}(\boldsymbol{\theta})^{1-1/N} \tilde{f}_n(\boldsymbol{\theta})^{1/N} \quad (10.259)$$

¹⁸ The ADF algorithm is similar to SEP, in that we compute the tilted distribution $q_{\setminus t} \propto f_t q_{t-1}$ and
¹⁹ then project it, without needing to keep the f_t factors. The difference is that instead of using the
²⁰ cavity distribution $q_{-1}(\boldsymbol{\theta})$ as prior, it uses the posterior from the previous time step, q_{t-1} . This
²¹ avoids the need to compute and store \tilde{f} , but results in overconfidence in the batch setting.
²²

²³ 10.7.8 Applications

²⁴ EP has been used for a variety of problems, such as the following.
²⁵

- ²⁶ • The **TrueSkill** system from Microsoft, [HMG07], which is a system to rank players of the Xbox
²⁷ 360 system.
- ²⁸ • Approximate inference in Gaussian processes with non-Gaussian likelihoods (see Section 18.4),
²⁹ where [NR08] showed good results.
- ³⁰ • The **probabilistic backpropagation** (PBP) algorithm of [HLA15a], which performs approximate
³¹ Bayesian inference for DNNs (see Section 17.6.2).
- ³² • Group sparse regression using spike-and-slab priors (Section 15.2.4), where [HLHLD13] showed
³³ 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

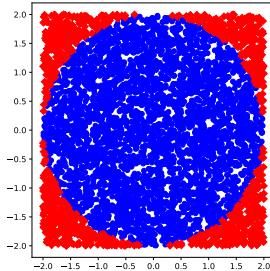
$$\mathbb{E}[f(\mathbf{x})] \approx \frac{1}{N_s} \sum_{n=1}^{N_s} f(\mathbf{x}_n) \quad (11.2)$$

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].

1
2
3
4
5
6
7
8
9
10



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

15 need to uniformly cover the entire space. In particular, it can be shown that the accuracy is in
16 principle independent of the dimensionality of \mathbf{x} , and only depends on the number of samples N_s
17 (see Section 11.2.2 for details). The catch is that we need a way to generate the samples $\mathbf{x}_n \sim p(\mathbf{x})$
18 in the first place. In addition, the estimator may have high variance. We will discuss this topic at
19 length in the sections below.

20

21 **11.2.1 Example: estimating π by Monte Carlo integration**

22

23 MC integration can be used for many applications, not just in ML and statistics. For example,
24 suppose we want to estimate π . We know that the area of a circle with radius r is πr^2 , but it is also
25 equal to the following definite integral:

26
$$I = \int_{-r}^r \int_{-r}^r \mathbb{I}(x^2 + y^2 \leq r^2) dx dy \quad (11.3)$$

27 Hence $\pi = I/(r^2)$. Let us approximate this by Monte Carlo integration. Let $f(x, y) = \mathbb{I}(x^2 + y^2 \leq r^2)$
28 be an indicator function that is 1 for points inside the circle, and 0 outside, and let $p(x)$ and $p(y)$ be
29 uniform distributions on $[-r, r]$, so $p(x) = p(y) = 1/(2r)$. Then

30
$$I = (2r)(2r) \int \int f(x, y)p(x)p(y)dx dy \quad (11.4)$$

31
$$= 4r^2 \int \int f(x, y)p(x)p(y)dx dy \quad (11.5)$$

32
$$\approx 4r^2 \frac{1}{N_s} \sum_{n=1}^{N_s} f(x_n, y_n) \quad (11.6)$$

33 Using 5000 samples, we find $\hat{\pi} = 3.10$ with standard error 0.09 compared to the true value of $\pi = 3.14$.
34 We can plot the points that are accepted or rejected as in Figure 11.1.

35

36 **11.2.2 Accuracy of Monte Carlo integration**

37

38 The accuracy of an MC approximation increases with sample size. This is illustrated in Figure 11.2.
39 On the top line, we plot a histogram of samples from a Gaussian distribution. On the bottom line,
40

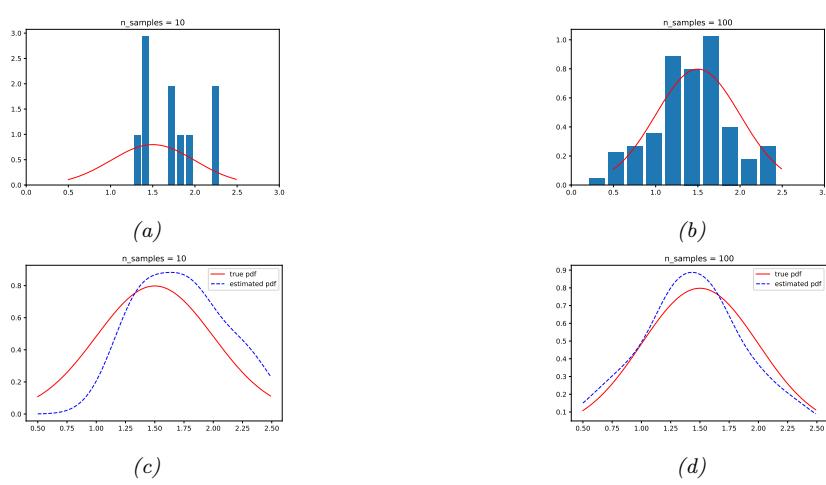


Figure 11.2: 10 and 100 samples from a Gaussian distribution, $\mathcal{N}(\mu = 1.5, \sigma^2 = 0.25)$. Solid red line is true pdf. Top row: histogram of samples. Bottom row: kernel density estimate derived from the samples. Generated by `mc_accuracy_demo.py`.

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,

$$(\hat{\mu} - \mu) \rightarrow \mathcal{N}(0, \frac{\sigma^2}{N_s}) \quad (11.7)$$

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:

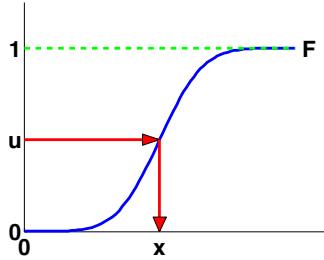
$$\hat{\sigma}^2 = \frac{1}{N_s} \sum_{n=1}^{N_s} (f(x_n) - \hat{\mu})^2 \quad (11.9)$$

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

$$39 \quad \Pr(F^{-1}(U) \leq x) = \Pr(U \leq F(x)) \quad (\text{applying } F \text{ to both sides}) \quad (11.11)$$

$$40 \quad = F(x) \quad (\text{because } \Pr(U \leq y) = y) \quad (11.12)$$

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

$$\text{Cov}[\mathbf{y}] = \mathbf{L}\text{Cov}[\mathbf{x}]\mathbf{L}^\top = \mathbf{L}\mathbf{I}\mathbf{L}^\top = \Sigma \quad (11.18)$$

11.4 Rejection sampling

Suppose we want to sample from the **target distribution**

$$p(\mathbf{x}) = \tilde{p}(\mathbf{x})/Z_p \quad (11.19)$$

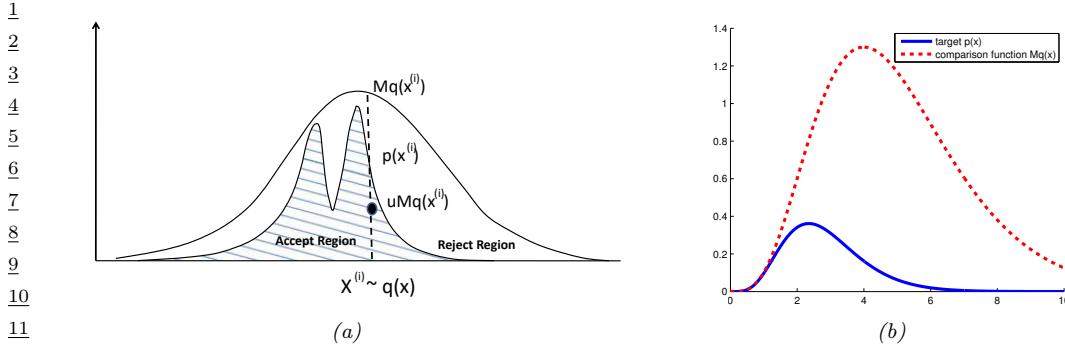


Figure 11.4: (a) Schematic illustration of rejection sampling. From Figure 2 of [And+03]. Used with kind permission of Nando de Freitas. (b) Rejection sampling from a $\text{Ga}(\alpha = 5.7, \lambda = 2)$ distribution (solid blue) using a proposal of the form $M\text{Ga}(k, \lambda - 1)$ (dotted red), 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 $Cq(x) \geq \tilde{p}(x)$.
12

13 14 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 where $\Gamma(\alpha)$ is the gamma function. One can show that if $X_i \stackrel{iid}{\sim} \text{Expon}(\lambda)$, and $Y = X_1 + \dots + X_k$,
20 then $Y \sim \text{Ga}(k, \lambda)$. For non-integer shape parameters α , we cannot use this trick. However, we can
21 use rejection sampling using a $\text{Ga}(k, \lambda - 1)$ distribution as a proposal, where $k = \lfloor \alpha \rfloor$. The ratio has
22 the form
23

24
$$\frac{p(x)}{q(x)} = \frac{\text{Ga}(x|\alpha, \lambda)}{\text{Ga}(x|k, \lambda - 1)} = \frac{x^{\alpha-1} \lambda^\alpha \exp(-\lambda x)/\Gamma(\alpha)}{x^{k-1} (\lambda - 1)^k \exp(-(\lambda - 1)x)/\Gamma(k)}$$
 (11.26)
25

26
$$= \frac{\Gamma(k) \lambda^\alpha}{\Gamma(\alpha) (\lambda - 1)^k} x^{\alpha-k} \exp(-x)$$
 (11.27)
27

28 This ratio attains its maximum when $x = \alpha - k$. Hence
29

30
$$C = \frac{\text{Ga}(\alpha - k|\alpha, \lambda)}{\text{Ga}(\alpha - k|k, \lambda - 1)}$$
 (11.28)
31

32 See Figure 11.4(b) for a plot.
33

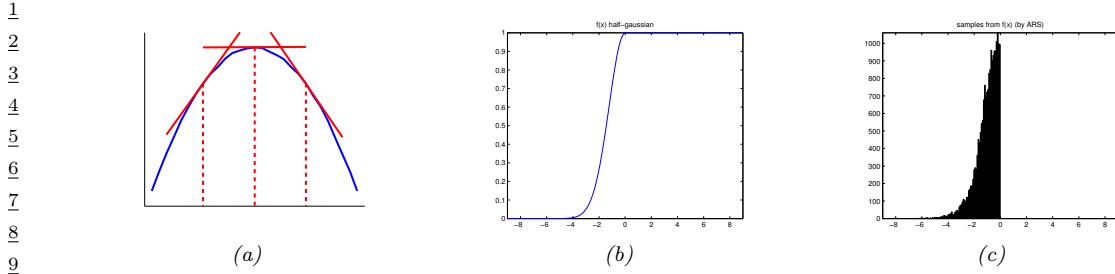
34 11.4.3 Adaptive rejection sampling

35 We now describe a method that can automatically come up with a tight upper envelope $q(x)$ to
36 any log concave 1d density $p(x)$. The idea is to upper bound the log density with a piecewise linear
37 function, as illustrated in Figure 11.5(a). We choose the initial locations for the pieces based on a
38 fixed grid over the support of the distribution. We then evaluate the gradient of the log density at
39 these locations, and make the lines be tangent at these points.
40

41 Since the log of the envelope is piecewise linear, the envelope itself is piecewise exponential:
42

43
$$q(x) = C_i \lambda_i \exp(-\lambda_i(x - x_{i-1})), \quad x_{i-1} < x \leq x_i$$
 (11.29)
44

45 3. This section is based on notes by Ioana A. Cosma, available at <http://users.aims.ac.za/~ioana/cp2.pdf>.
46



10 *Figure 11.5: (a) Idea behind adaptive rejection sampling. We place piecewise linear upper (and lower) bounds*
11 *on the log-concave density. Adapted from Figure 1 of [GW92]. Generated by `ars_envelope.py`. (b-c) Using*
12 *ARS to sample from a half-Gaussian. Generated by `ars_demo.py`.*

13

14

15 where x_i are the grid points. It is relatively straightforward to sample from this distribution. If the
16 sample x is rejected, we create a new grid point at x , and thereby refine the envelope. As the number
17 of grid points is increased, the tightness of the envelope improves, and the rejection rate goes down.
18 This is known as **adaptive rejection sampling** (ARS) [GW92]. Figure 11.5(b-c) gives an example
19 of the method in action. As with standard rejection sampling, it can be applied to unnormalized
20 distributions.

21

22 11.4.4 Rejection sampling in high dimensions

23

24 It is clear that we want to make our proposal $q(\mathbf{x})$ as close as possible to the target distribution $p(\mathbf{x})$,
25 while still being an upper bound. But this is quite hard to achieve, especially in high dimensions. To
26 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
27 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
28 by $C = (\sigma_q/\sigma_p)^D$. The acceptance rate is $1/C$ (since both p and q are normalized), which decreases
29 exponentially fast with dimension. For example, if σ_q exceeds σ_p by just 1%, then in 1000 dimensions
30 the acceptance ratio will be about 1/20,000. This is a fundamental weakness of rejection sampling.

31

32 11.5 Importance sampling

33

34 In this section, we describe a Monte Carlo method known as **importance sampling** for approximating integrals of the form

35
$$\mathbb{E}[\varphi(\mathbf{x})] = \int \varphi(\mathbf{x}) \pi(\mathbf{x}) \pi(\mathbf{x}) d\mathbf{x} \quad (11.30)$$

36

37 where φ is called a **target function**, and $\pi(\mathbf{x})$ is the **target distribution**, often a conditional
38 distribution of the form $\pi(\mathbf{x}) = p(\mathbf{x}|\mathbf{y})$. Since in general it is difficult to draw from the target
39 distribution, we will instead draw from some **proposal distribution** $q(\mathbf{x})$ (which will usually
40 depend on \mathbf{y}). We then adjust for the inaccuracies of this by associating weights with each sample,
41 so we end up with a weighted MC approximation:

42
$$\mathbb{E}[\varphi(\mathbf{x})] \approx \sum_{n=1}^N W_n \varphi(\mathbf{x}_n) \quad (11.31)$$

43

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:

$$\mathbb{E}[\varphi(\mathbf{x})] = \int \varphi(\mathbf{x})\pi(\mathbf{x})d\mathbf{x} = \frac{\int \varphi(\mathbf{x})\tilde{\gamma}(\mathbf{x})d\mathbf{x}}{\int \tilde{\gamma}(\mathbf{x})d\mathbf{x}} = \frac{\int \left[\frac{\tilde{\gamma}(\mathbf{x})}{q(\mathbf{x})} \varphi(\mathbf{x}) \right] q(\mathbf{x})d\mathbf{x}}{\int \left[\frac{\tilde{\gamma}(\mathbf{x})}{q(\mathbf{x})} \right] q(\mathbf{x})d\mathbf{x}} \quad (11.37)$$

$$\approx \frac{\frac{1}{N_s} \sum_{n=1}^{N_s} \tilde{w}_n \varphi(\mathbf{x}_n)}{\frac{1}{N_s} \sum_{n=1}^{N_s} \tilde{w}_n} \quad (11.38)$$

1 were we have defined the **unnormalized weights**

$$\frac{3}{4} \quad \tilde{w}_n = \frac{\tilde{\gamma}(\mathbf{x})}{q(\mathbf{x})} \quad (11.39)$$

6 We can write Equation (11.38) more compactly as

$$\frac{8}{9} \quad \mathbb{E}[\varphi(\mathbf{x})] \approx \sum_{n=1}^{N_s} W_n \varphi(\mathbf{x}_n) \quad (11.40)$$

11 where we have defined the **normalized weights** by

$$\frac{13}{14} \quad W_n = \frac{\tilde{w}_n}{\sum_{n'=1}^{N_s} \tilde{w}_{n'}} \quad (11.41)$$

16 This is equivalent to approximating the target distribution using a weighted sum of delta functions:

$$\frac{18}{19} \quad \pi(\mathbf{x}) \approx \sum_{n=1}^{N_s} W_n \delta(\mathbf{x} - \mathbf{x}_n) \triangleq \hat{\pi}(\mathbf{x}) \quad (11.42)$$

21 As a byproduct of this algorithm we get the following appoximation to the normalization constant:

$$\frac{23}{24} \quad Z \approx \frac{1}{N_s} \sum_{n=1}^{N_s} \tilde{w}_n \triangleq \hat{Z} \quad (11.43)$$

27 11.5.3 Choosing the proposal

29 The performance of importance sampling depends crucially on the quality of the proposal distribution.
30 As we mentioned, we require that the support of q cover the support of the target (i.e., $\tilde{\gamma}(\mathbf{x}) > 0 \implies$
31 $q(\mathbf{x}) > 0$). However, we also want the proposal to not be too “loose” or a “covering”. Ideally it should
32 also take into account properties of the target function φ as well, as shown in Figure 11.6. This can
33 yield subsantial benefits, as shown in the “**target aware Bayesian inference**” scheme of [Rai+20].
34 However, usually the target function φ is unknown or ignored, so we just try to find a “generally
35 useful” approximation to the target.

36 One way to come up with a good proposal is to learn one, by optimizing the variational lower
37 bound or ELBO (see Section 10.1.2). Indeed, if we fix the parameters of the generative model, we
38 can think of importance weighted autoencoders (Section 10.5.1) as learning a good IS proposal. More
39 details on this connection can be found in [DS18].

40

41 11.5.4 Annealed importance sampling (AIS)

42 In this section, we describe a method known as **annealed importance sampling** [Nea01] for
43 sampling from complex, possibly multimodal distributions. Assume we want to sample from some
44 target distribution $p_0(\mathbf{x}) \propto f_0(\mathbf{x})$ (where $f_0(\mathbf{x})$ is the unnormalized version), but we cannot do
45 so easily. However, suppose that there is an easier distribution which we *can* sample from, call it
46
47

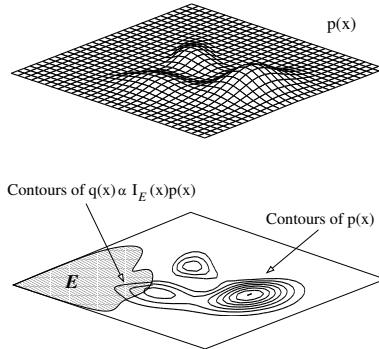


Figure 11.6: In importance sampling, we should sample from a distribution that takes into account regions where $\pi(\mathbf{x})$ has high probability and where $\varphi(\mathbf{x})$ is large. Here the function to be evaluated is an indicator function of a set, corresponding to a set of rare events in 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

$$\begin{aligned} q(\mathbf{v}) &\propto g(\mathbf{v}) = f_n(\mathbf{v}_n)T_{n-1}(\mathbf{v}_n, \mathbf{v}_{n-1}) \cdots T_2(\mathbf{v}_2, \mathbf{v}_1)T_0(\mathbf{v}_1, \mathbf{v}_0) \end{aligned} \quad (11.49)$$

$$\propto p(\mathbf{v}_n)p(\mathbf{v}_{n-1}|\mathbf{v}_n) \cdots p(\mathbf{v}_1|\mathbf{v}_0) \quad (11.50)$$

⁷ 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
⁸ marginals of the sampled sequences from this extended model are equivalent to samples from $p_0(\mathbf{x})$,
⁹ we see that we are using the correct weights.
¹⁰

¹¹ 11.5.4.1 Estimating normalizing constants using AIS

¹² An important application of AIS is to evaluate a ratio of partition functions. Notice that $Z_0 =$
¹³ $\int f_0(\mathbf{x})d\mathbf{x} = \int \varphi(\mathbf{v})d\mathbf{v}$, and $Z_n = \int f_n(\mathbf{x})d\mathbf{x} = \int g(\mathbf{v})d\mathbf{v}$. Hence
¹⁴

$$\begin{aligned} \frac{Z_0}{Z_n} &= \frac{\int \varphi(\mathbf{v})d\mathbf{v}}{\int g(\mathbf{v})d\mathbf{v}} = \frac{\int \frac{\varphi(\mathbf{v})}{g(\mathbf{v})}g(\mathbf{v})d\mathbf{v}}{\int g(\mathbf{v})d\mathbf{v}} = \mathbb{E}_g\left[\frac{\varphi(\mathbf{v})}{g(\mathbf{v})}\right] \approx \frac{1}{S} \sum_{s=1}^S w_s \end{aligned} \quad (11.51)$$

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.
²³

²⁴ 11.6 Controlling Monte Carlo variance

²⁵ As we mentioned in Section 11.2.2, the standard error in a Monte Carlo estimate is $O(1/\sqrt{S})$, where
²⁶ S is the number of (independent) samples. Consequently it may take many samples to reduce the
²⁷ variance to a sufficiently small value. In this section, we discuss some ways to reduce the variance of
²⁸ sampling methods. For more details, see e.g., [KTB11].
²⁹

³⁰ 11.6.1 Rao-Blackwellisation

³¹ In this section, we discuss a useful technique for reducing the variance of MC estimators known as
³² **Rao-Blackwellisation**. To explain the method, suppose we have two rv's, X and Y , and we want
³³ to estimate $\bar{f} = \mathbb{E}[f(X, Y)]$. The naive approach is to use an MC approximation
³⁴

$$\hat{f}_{MC} = \frac{1}{S} \sum_{s=1}^S f(X_s, Y_s) \quad (11.52)$$

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

⁴³ Now suppose we can analytically marginalize out Y , provided we know X , i.e., we can tractable
⁴⁴ compute

$$f_X(X_s) = \int dY p(Y|X_s) f(X_s, Y) = \mathbb{E}[f(X, Y)|X = X_s] \quad (11.53)$$

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.

15 16 11.6.2 Control variates

17 Suppose we want to estimate $\mu = \mathbb{E}[f(X)]$ using an unbiased estimator $m(\mathcal{X}) = \frac{1}{S} \sum_{s=1}^S m(x_s)$,
18 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
19 a single random variable as well as a set of samples.) Now consider the alternative estimator

20
21 $m^*(\mathcal{X}) = m(\mathcal{X}) + c(b(\mathcal{X}) - \mathbb{E}[b(\mathcal{X})])$ (11.56)

22 This is called a **control variate**, and b is called a **baseline**. (Once again we abuse notation and use
23 $b(\mathcal{X}) = \frac{1}{S} \sum_{s=1}^S b(x_s)$ and $m^*(\mathcal{X}) = \frac{1}{S} \sum_{s=1}^S m^*(x_s)$.)

24 It is easy to see that $m^*(\mathcal{X})$ is an unbiased estimator, since $\mathbb{E}[m^*(X)] = \mathbb{E}[m(X)] = \mu$. However,
25 it can have lower variance, provided b is correlated with m . To see this, note that

26
27 $\mathbb{V}[m^*(X)] = \mathbb{V}[m(X)] + c^2 \mathbb{V}[b(X)] + 2c \text{Cov}[m(X), b(X)]$ (11.57)

28 By taking the derivative of $\mathbb{V}[m^*(X)]$ wrt c and setting to 0, we find that the optimal value is

29
30 $c^* = -\frac{\text{Cov}[m(X), b(X)]}{\mathbb{V}[b(X)]}$ (11.58)

31 The corresponding variance of the new estimator is now

32
33 $\mathbb{V}[m^*(X)] = \mathbb{V}[m(X)] - \frac{\text{Cov}[m(X), b(X)]^2}{\mathbb{V}[b(X)]} = (1 - \rho_{m,b}^2) \mathbb{V}[m(X)] \leq \mathbb{V}[m(X)]$ (11.59)

34 where $\rho_{m,b}^2$ is the correlation of the basic estimator and the baseline function. If we can ensure
35 this correlation is high, we can reduce the variance. Intuitively, the CV estimator is exploiting
36 information about the errors in the estimate of a known quantity, namely $\mathbb{E}[b(X)]$, to reduce the
37 errors in estimating the unknown quantity, namely μ .

38 We give a simple worked example in Section 11.6.2.1. See Section 10.3.1 for an example of this
39 technique applied to blackbox variational inference.

1 **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

6
$$\mu = \int_0^1 \frac{1}{1+x} dx = \ln 2 \approx 0.693 \quad (11.60)$$

7
8

9 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
10 $\mathbb{E}[m(\mathcal{X})] = 0.6935$ with standard error $\text{se} = 0.0037$.

11 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
12 $\mathbb{E}[b(X)] = \int_0^1 (1+x) dx = \frac{3}{2}$. The control variate estimator is given by
13

14
$$m^*(\mathcal{X}) = \frac{1}{S} \sum_{s=1}^S f(x_s) + c \left(\frac{1}{S} \sum_{s=1}^S b(x_s) - \frac{3}{2} \right) \quad (11.61)$$

15
16

17 The optimal value can be estimated from the samples of $m(x_s)$ and $b(x_s)$, and plugging into
18 Equation (11.58) to get $c^* \approx 0.4773$. Using $S = 1500$, we find $\mathbb{E}[m^*(\mathcal{X})] = 0.6941$ and $\text{se} = 0.0007$.

19 See also Section 11.6.3.1, where we analyze this example using antithetic sampling.

20

21 **11.6.3 Antithetic sampling**

22 In this section, we discuss **antithetic sampling**, which is a simple way to reduce variance.⁵ Suppose
23 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
24 $\hat{\theta} = (Y_1 + Y_2)/2$. The variance of this estimate is
25

26
$$\mathbb{V}[\hat{\theta}] = \frac{\mathbb{V}[Y_1] + \mathbb{V}[Y_2] + 2\text{Cov}[Y_1, Y_2]}{4} \quad (11.62)$$

27
28
29

30 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
31 “opposite”, but with the same mean.

32 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
33 can define $y'_i = 1 - y_i$. The distribution of y'_i is still $\text{Unif}(0, 1)$, but $\text{Cov}[y_i, y'_i] < 1$.
34

35 **11.6.3.1 Example**

36 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
37 estimate using $2N$ samples from $\text{Unif}(0, 1)$, and let $\hat{\mu}_{\text{anti}}$ be the MC estimate using the above antithetic
38 sampling scheme applied to N base samples from $\text{Unif}(0, 1)$. The exact value is $\mu = \ln 2 \approx 0.6935$.
39 For the classical method, with $N = 750$, we find $\mathbb{E}[\hat{\mu}_{\text{mc}}] = 0.69365$ with a standard error of 0.0037.
40 For the antithetic method, we find $\mathbb{E}[\hat{\mu}_{\text{anti}}] = 0.6939$ with a standard error of 0.0007, which matches
41 the control variate method of Section 11.6.2.1.
42

43 ⁴. The example is from https://en.wikipedia.org/wiki/Control_variates, with modified notation. See `control_variates.py` for some code.

44 ⁵. Our presentation is based on https://en.wikipedia.org/wiki/Antithetic_variates. See `antithetic_sampling.py` for the code.

45

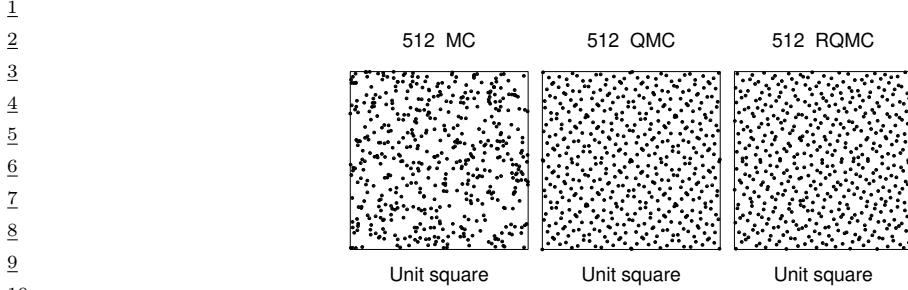


Figure 11.7: Illustration of Monte Carlo (MC), Quasi MC (QMC) from a Sobol sequence, and Randomized QMC using a scrambling method. Adapted from Figure 1 of [OR20]. Used with kind permission of Art Owen.

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 ² 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:

$$\begin{aligned} \text{14} \quad A &= \min(1, \frac{p^*(\mathbf{x}')}{p^*(\mathbf{x})}) \\ \text{15} \end{aligned} \tag{12.1}$$

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:

$$\begin{aligned} \text{25} \quad A &= \min(1, \alpha) \\ \text{26} \end{aligned} \tag{12.2}$$

$$\begin{aligned} \text{27} \quad \alpha &= \frac{p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}{p^*(\mathbf{x})q(\mathbf{x}'|\mathbf{x})} = \frac{p^*(\mathbf{x}')/q(\mathbf{x}'|\mathbf{x})}{p^*(\mathbf{x})/q(\mathbf{x}|\mathbf{x}')} \\ \text{28} \end{aligned} \tag{12.3}$$

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

$$\begin{aligned} \text{35} \quad \alpha &= \frac{(\tilde{p}(\mathbf{x}')/Z) q(\mathbf{x}|\mathbf{x}')}{(\tilde{p}(\mathbf{x})/Z) q(\mathbf{x}'|\mathbf{x})} \\ \text{36} \end{aligned} \tag{12.4}$$

³⁷ so the Z 's cancel. Hence we can sample from p^* even if Z is unknown.

³⁸ A proposal distribution q is valid or admissible if it “covers” the support of the target. Formally,
³⁹ we can write this as

$$\begin{aligned} \text{41} \quad \text{supp}(p^*) &\subseteq \cup_x \text{supp}(q(\cdot|x)) \\ \text{42} \end{aligned} \tag{12.5}$$

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 Furthermore, from Theorem 2.8.2, this distribution is unique, since the chain is ergodic and irreducible.

□

31 12.2.3 Proposal distributions

32 In this section, we discuss some common proposal distributions. Note, however, that good proposal
33 design is often intimately dependent on the form of the target distribution (most often the posterior).

34

35 12.2.3.1 Independence sampler

36 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
37 state, we get a method known as the **independence sampler**, which is similar to importance
38 sampling (Section 11.5). The function $q(\mathbf{x}')$ can be any suitable distribution, such as a Gaussian.
39 This has non-zero probability density on the entire state space, and hence is a valid proposal for any
40 unconstrained continuous state space.

41

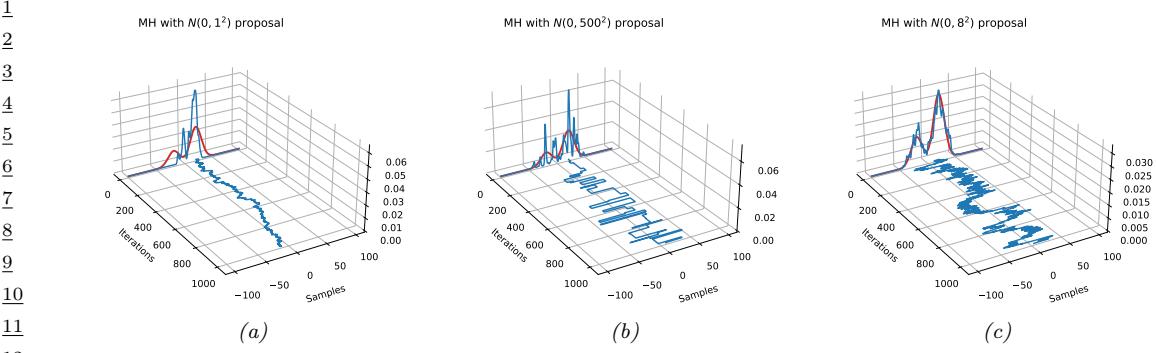


Figure 12.1: An example of the Metropolis Hastings algorithm for sampling from a mixture of two 1D Gaussians ($\mu = (-20, 20)$, $\pi = (0.3, 0.7)$, $\Sigma = (100, 100)$), using a Gaussian proposal with standard deviation of $\tau \in \{1, 8, 500\}$. (a) When $\tau = 1$, the chain gets trapped near 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

6
$$q(\mathbf{x}'|\mathbf{x}) = \sum_{k=1}^K w_k q_k(\mathbf{x}'|\mathbf{x}) \quad (12.16)$$

7

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

14
$$q(\mathbf{x}'|\mathbf{x}) = \sum_{\mathbf{x}_1} \cdots \sum_{\mathbf{x}_{K-1}} q_1(\mathbf{x}_1|\mathbf{x}) q_2(\mathbf{x}_2|\mathbf{x}_1) \cdots q_K(\mathbf{x}| \mathbf{x}_{K-1}) \quad (12.17)$$

15

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**
23

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

33
$$q(\mathbf{x}'|\mathbf{x}, \mathcal{D}) = \pi_0 q_0(\mathbf{x}'|\mathbf{x}) + \sum_k \pi_k q_k(x'_k | f_k(\mathcal{D})) \quad (12.18)$$

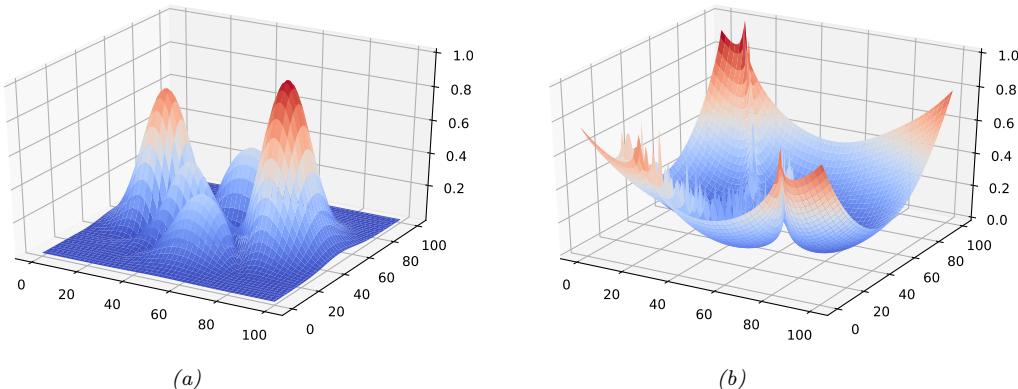
34

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

45 One can change the parameters of the proposal as the algorithm is running to increase efficiency.
46 This is called **adaptive MCMC**. This allows one to start with a broad covariance (say), allowing
47



16 Figure 12.2: (a) A peaky distribution. (b) Corresponding energy function. Generated by simulated annealing 2d demo.ipynb.
17

²⁰ large moves through the space until a mode is found, followed by a narrowing of the covariance to
²¹ ensure careful exploration of the region around the mode.
²²

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.

27 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

36 **Simulated annealing** [KJV83; LA87] is a **stochastic local search** algorithm that attempts to
37 find the global minimum of a black-box function $\mathcal{E}(\mathbf{x})$, where $\mathcal{E}()$ is known as the **energy function**.
38 The method works by converting the energy to an (unnormalized) probability distribution over states
39 by defining $p(\mathbf{x}) = \exp(-\mathcal{E}(\mathbf{x}))$, and then using a variant of the **Metropolis Hastings** algorithm to
40 sample from a set of probability distributions, designed so that at the final step, the method samples
41 from one of the modes of the distribution, i.e., it finds one of the most likely states, or lowest energy
42 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

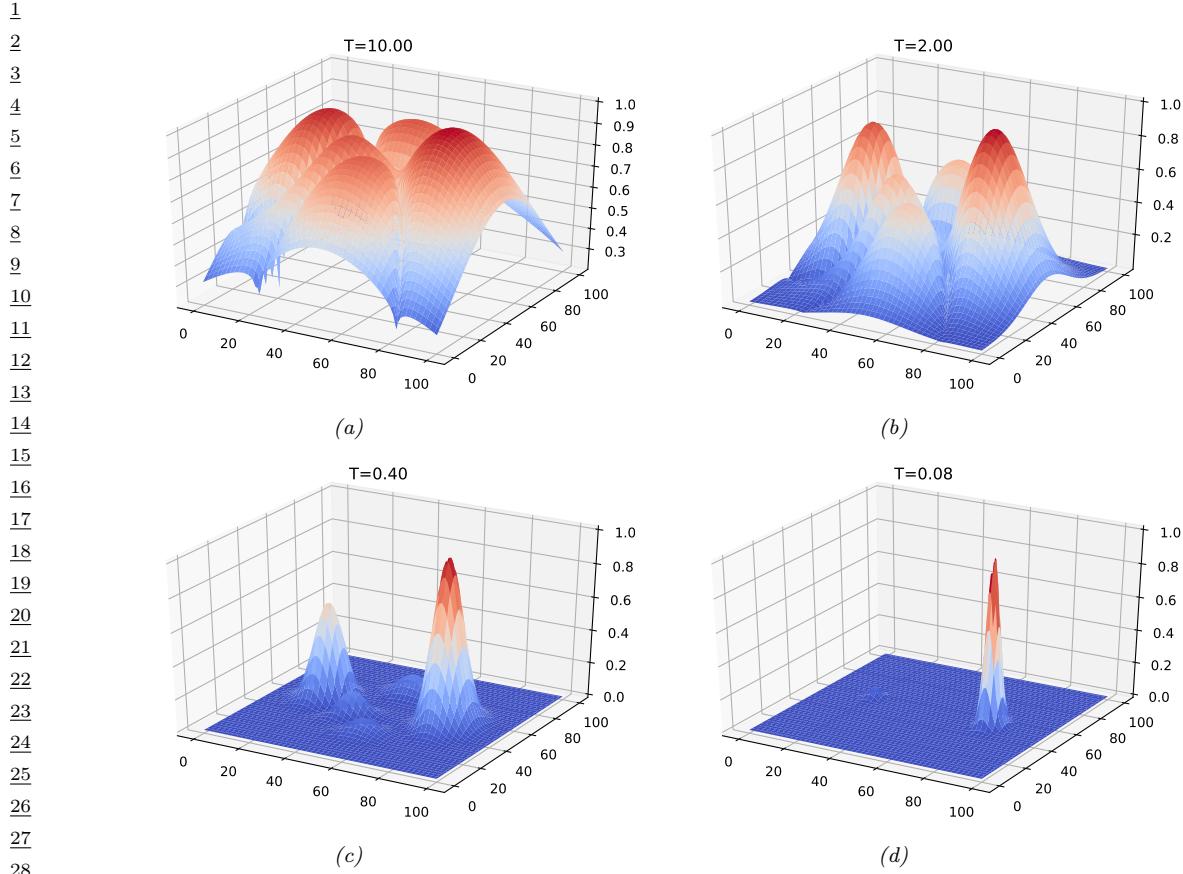


Figure 12.3: Annealed version of the distribution in Figure 12.2a at different temperatures. Generated by [simulated_annealing_2d_demo.ipynb](#).

31

32

33 states) that they have, by defining

$$34 \quad p_T(\mathbf{x}) = \exp(-\mathcal{E}(\mathbf{x})/T) \quad (12.19)$$

35 where T is the temperature, which is reduced over time. As an example, consider the **peaks function**:

36

$$37 \quad p(x, y) \propto |3(1-x)^2 e^{-x^2-(y+1)^2} - 10(\frac{x}{5} - x^3 - y^5) e^{-x^2-y^2} - \frac{1}{3} e^{-(x+1)^2-y^2}| \quad (12.20)$$

38 This is plotted in Figure 12.2a. The corresponding energy is in Figure 12.2b. We plot annealed
 39 versions of this distribution in Figure 12.3. At high temperatures, $T \gg 1$, the surface is approximately
 40 flat, and hence it is easy to move around (i.e., to avoid local optima). As the temperature cools,
 41 the largest peaks become larger, and the smallest peaks disappear. By cooling slowly enough, it is
 42 possible to “track” the largest peak, and thus find the global optimum (minimum energy state). This
 43 is an example of a **continuation method**.

44

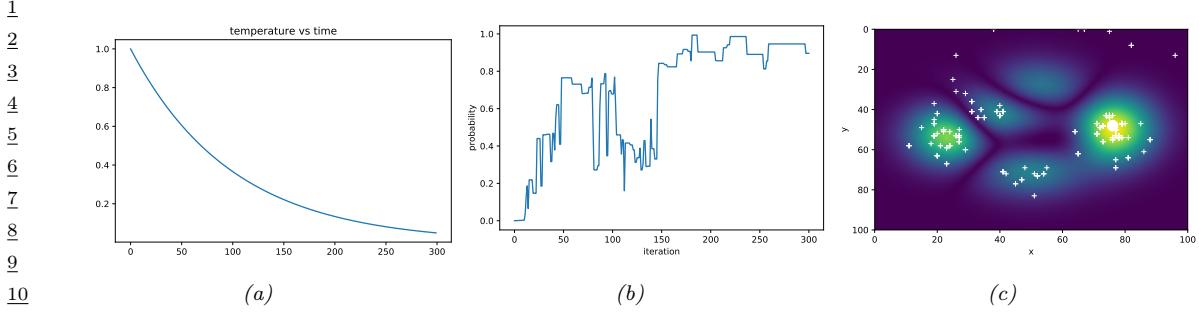


Figure 12.4: Simulated annealing applied to the distribution in Figure 12.2a. (a) Temperature vs time. (b) Probability of each visited point vs time. (c) Visited samples, superimposed on the target distribution. The big white dot is the highest probability point found. Generated by [simulated_annealing_2d_demo.ipynb](#).

In more detail, at each step, we sample a new state according to some proposal distribution $x' \sim q(\cdot | x_t)$. For real-valued parameters, this is often simply a random walk proposal centered on the current iterate, $x' = x_t + \epsilon_{t+1}$, where $\epsilon_{t+1} \sim \mathcal{N}(\mathbf{0}, \Sigma)$. (The matrix Σ 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

$$\alpha_{t+1} = \exp(-(\mathcal{E}(x') - \mathcal{E}(x_t))/T_t) \quad (12.21)$$

where T_t is the temperature of the system. We then accept the new state (i.e., set $x_{t+1} = x'$) with probability $\min(1, \alpha_{t+1})$, otherwise we stay in the current state (i.e., set $x_{t+1} = x_t$). 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

$$\begin{aligned} \alpha &= \frac{p(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}{p(\mathbf{x})q(\mathbf{x}'|\mathbf{x})} = \frac{p(x'_i|\mathbf{x}'_{-i})p(\mathbf{x}'_{-i})p(x_i|\mathbf{x}'_{-i})}{p(x_i|\mathbf{x}_{-i})p(\mathbf{x}_{-i})p(x'_i|\mathbf{x}_{-i})} \\ \text{³¹ } &= \frac{p(x'_i|\mathbf{x}_{-i})p(\mathbf{x}_{-i})p(x_i|\mathbf{x}_{-i})}{p(x_i|\mathbf{x}_{-i})p(\mathbf{x}_{-i})p(x'_i|\mathbf{x}_{-i})} = 1 \end{aligned} \tag{12.23}$$

³² where we exploited the fact that $\mathbf{x}'_{-i} = \mathbf{x}_{-i}$, and that $q(\mathbf{x}'|\mathbf{x}) = p(x'_i|\mathbf{x}_{-i})$.

³³ The fact that the acceptance rate is 100% does not necessarily mean that Gibbs will converge
³⁴ rapidly, since it only updates one coordinate at a time (see Section 12.3.7). However, if we can group
³⁵ together correlated variables, then we can sample them as a group, which can significantly help
³⁶ mixing.

³⁷ 3. Josiah Willard Gibbs, 1839–1903, was an American physicist.

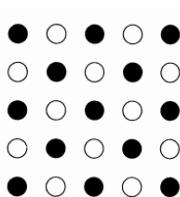


Figure 12.5: Illustration of checkerboard pattern for a 2d MRF. This allows for parallel updates.

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

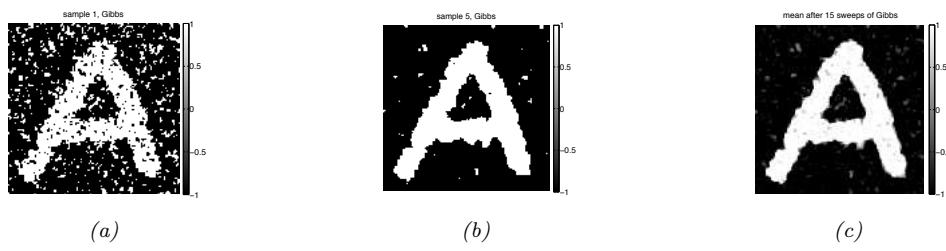


Figure 12.6: Example of image denoising using Gibbs sampling. We use an Ising prior with $J = 1$ and a Gaussian noise model with $\sigma = 2$. (a) Sample from the posterior after one sweep over the image. (b) Sample after 5 sweeps. (c) Posterior mean, computed by averaging over 15 sweeps. Compare to Figure 10.3 which shows the results of mean field inference. Generated by `ising_image_denoise_demo.py`.

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.

One application of Ising models is as a prior for binary image denoising problems. In particular, 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 $p(\mathbf{x})$ 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 Gaussian. Let $\psi_i(x_i) = \mathcal{N}(y_i|x_i, \sigma^2)$ be the corresponding “local evidence” term. The full conditional becomes

$$p(x_i = +1|\mathbf{x}_{-i}, \mathbf{y}, \boldsymbol{\theta}) = \frac{\exp[J\eta_i]\psi_i(+1)}{\exp[J\eta_i]\psi_i(+1) + \exp[-J\eta_i]\psi_i(-1)} \quad (12.30)$$

$$= \sigma\left(2J\eta_i - \log \frac{\psi_i(+1)}{\psi_i(-1)}\right) \quad (12.31)$$

Now the probability of x_i entering each state is determined both by compatibility with its neighbors (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

We can extend Section 12.3.3 to the Potts models as follows. Recall that the model has the following form:

$$p(\mathbf{x}) = \frac{1}{Z} \exp(-\mathcal{E}(\mathbf{x})) \quad (12.32)$$

$$\mathcal{E}(\mathbf{x}) = -J \sum_{i \sim j} \mathbb{I}(x_i = x_j) \quad (12.33)$$

For a node i with neighbors $\text{nbr}(i)$, the full conditional is thus given by

$$p(x_i = k|\mathbf{x}_{-i}) = \frac{\exp(J \sum_{n \in \text{nbr}(i)} \mathbb{I}(x_n = k))}{\sum_{k'} \exp(J \sum_{n \in \text{nbr}(i)} \mathbb{I}(x_n = k'))} \quad (12.34)$$

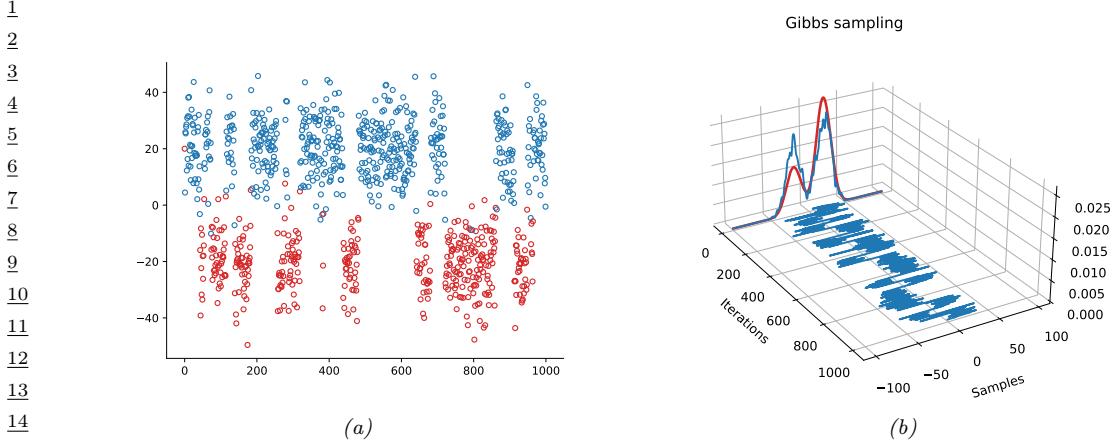


Figure 12.7: (a) Some samples from a mixture of two 1d Gaussians generated using Gibbs sampling. Color denotes the value of z , vertical location denotes the value of x . Horizontal axis represents time (sample number). (b) Traceplot of x over time, and the resulting empirical distribution is shown in 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

1 in state 1, we will likely stay in that state for a while, and generate values near μ_1 ; then we will
2 stochastically jump to state 2, and stay near there for a while, etc. By contrast, independent samples
3 from the joint would not be correlated at all. This means that MCMC samples carry less information
4 than independent samples, a fact we return to in Section 12.6.2.3.

5
6 In Section 12.3.5.2, we modify this example to sample the parameters of the GMM from their
7 posterior, $p(\boldsymbol{\theta}|\mathcal{D})$, instead of sampling from $p(\mathcal{D}|\boldsymbol{\theta})$.

89

10 12.3.5.2 Unknown parameters

11 Now suppose the parameters are unknown, so we want to fit the model to data. If we use a
12 conditionally conjugate factored prior, then the full joint distribution is given by
13

$$\begin{aligned} \text{14} \\ \text{15} \quad p(\mathbf{x}, \mathbf{z}, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) &= p(\mathbf{x}|\mathbf{z}, \boldsymbol{\mu}, \boldsymbol{\Sigma})p(\mathbf{z}|\boldsymbol{\pi})p(\boldsymbol{\pi}) \prod_{k=1}^K p(\boldsymbol{\mu}_k)p(\boldsymbol{\Sigma}_k) \end{aligned} \quad (12.38)$$

$$\begin{aligned} \text{16} \\ \text{17} \quad &= \left(\prod_{i=1}^N \prod_{k=1}^K (\pi_k \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k))^{\mathbb{I}(z_i=k)} \right) \times \\ \text{18} \quad &\quad \text{Dir}(\boldsymbol{\pi}|\boldsymbol{\alpha}) \prod_{k=1}^K \mathcal{N}(\boldsymbol{\mu}_k | \mathbf{m}_0, \mathbf{V}_0) \text{IW}(\boldsymbol{\Sigma}_k | \mathbf{S}_0, \nu_0) \end{aligned} \quad (12.39)$$

$$\begin{aligned} \text{19} \\ \text{20} \quad &\quad \text{Dir}(\boldsymbol{\pi}|\boldsymbol{\alpha}) \prod_{k=1}^K \mathcal{N}(\boldsymbol{\mu}_k | \mathbf{m}_0, \mathbf{V}_0) \text{IW}(\boldsymbol{\Sigma}_k | \mathbf{S}_0, \nu_0) \end{aligned} \quad (12.40)$$

21 We use the same prior for each mixture component.

22 The full conditionals are as follows. For the discrete indicators, we have

$$\begin{aligned} \text{23} \\ \text{24} \quad p(z_i = k | \mathbf{x}_i, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) &\propto \pi_k \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \end{aligned} \quad (12.41)$$

25 For the mixing weights, we have (using results from Section 3.2.2)

$$\begin{aligned} \text{26} \\ \text{27} \quad p(\boldsymbol{\pi}|\mathbf{z}) &= \text{Dir}(\{\alpha_k + \sum_{i=1}^N \mathbb{I}(z_i=k)\}_{k=1}^K) \end{aligned} \quad (12.42)$$

28 For the means, we have (using results from Section 3.2.4.1)

$$\begin{aligned} \text{29} \\ \text{30} \quad p(\boldsymbol{\mu}_k | \boldsymbol{\Sigma}_k, \mathbf{z}, \mathbf{x}) &= \mathcal{N}(\boldsymbol{\mu}_k | \mathbf{m}_k, \mathbf{V}_k) \end{aligned} \quad (12.43)$$

$$\begin{aligned} \text{31} \\ \text{32} \quad \mathbf{V}_k^{-1} &= \mathbf{V}_0^{-1} + N_k \boldsymbol{\Sigma}_k^{-1} \end{aligned} \quad (12.44)$$

$$\begin{aligned} \text{33} \\ \text{34} \quad \mathbf{m}_k &= \mathbf{V}_k (\boldsymbol{\Sigma}_k^{-1} N_k \bar{\mathbf{x}}_k + \mathbf{V}_0^{-1} \mathbf{m}_0) \end{aligned} \quad (12.45)$$

$$\begin{aligned} \text{35} \\ \text{36} \quad N_k &\triangleq \sum_{i=1}^N \mathbb{I}(z_i=k) \end{aligned} \quad (12.46)$$

$$\begin{aligned} \text{37} \\ \text{38} \quad \bar{\mathbf{x}}_k &\triangleq \frac{\sum_{i=1}^N \mathbb{I}(z_i=k) \mathbf{x}_i}{N_k} \end{aligned} \quad (12.47)$$

39

1 For the covariances, we have (using results from Section 3.2.4.2)

$$\underline{4} \quad p(\boldsymbol{\Sigma}_k | \boldsymbol{\mu}_k, \mathbf{z}, \mathbf{x}) = \text{IW}(\boldsymbol{\Sigma}_k | \mathbf{S}_k, \nu_k) \quad (12.48)$$

$$\underline{5} \quad \mathbf{S}_k = \mathbf{S}_0 + \sum_{i=1}^N \mathbb{I}(z_i = k) (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T \quad (12.49)$$

$$\underline{8} \quad \nu_k = \nu_0 + N_k \quad (12.50)$$

10 12.3.6 Sampling from the full conditionals

12 When implementing Gibbs sampling, we have to sample from the full conditionals. Sometimes this
13 is difficult, especially when we are not using a conjugate prior. We discuss some possible solutions
14 below.

16 12.3.6.1 Adaptive rejection Metropolis sampling

18 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
19 (Section 11.4.3). This is known as **adaptive rejection Metropolis sampling** [GBT95].

21 12.3.6.2 Metropolis within Gibbs

23 Another possibility is to use the MH algorithm; this is called **Metropolis within Gibbs**. In more
24 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:

26 1. Propose $x'_i \sim q(x'_i | x_i^s)$

28 2. Compute the acceptance probability $A_i = \min(1, \alpha_i)$ where

$$\underline{30} \quad \alpha_i = \frac{p(\mathbf{x}_{1:i-1}^{s+1}, x'_i, \mathbf{x}_{i+1:D}^s) / q(x'_i | x_i^s)}{p(\mathbf{x}_{1:i-1}^{s+1}, x_i^s, \mathbf{x}_{i+1:D}^s) / q(x_i^s | x'_i)} \quad (12.51)$$

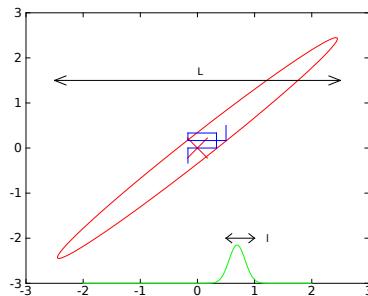
34 3. Sample $u \sim U(0, 1)$ and set $x_i^{s+1} = x'_i$ if $u < A_i$, and set $x_i^{s+1} = x_i^s$ otherwise.

36 12.3.7 Blocked Gibbs sampling

38 Gibbs sampling can be quite slow, since it only updates one variable at a time (so-called **single site**
39 **updating**). If the variables are highly correlated, it will take a long time to move away from the
40 current state. This is illustrated in Figure 12.8, where we illustrate sampling from a 2d Gaussian. If
41 the variables are highly correlated, the algorithm will move very slowly through the state space. In
42 particular, the size of the moves is controlled by the variance of the conditional distributions. If this
43 is ℓ in the x_1 direction, and the support of the distribution is L along this dimension, then we need
44 $O((L/\ell)^2)$ steps to obtain an independent sample.

45 In some cases we can efficiently sample groups of variables at a time. This is called **blocked**
46 **Gibbs sampling** [JKK95; WY02], and can make much bigger moves through the state space.

1
2
3
4
5
6
7
8
9
10



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`.

13

14

15

16

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
20

$$21 \quad p(\theta, z|x) \propto p(\theta) \prod_{t=1}^T p(x_t|z_t, \theta)p(z_t|z_{t-1}, \theta) \quad (12.52)$$

22

23 We can use blocked Gibbs sampling, where we alternate between sampling from $p(\theta|z, x)$ and
24 $p(z|x, \theta)$; The former is easy to do (assuming conjugate priors), since all variables in the model are
25 observed. The latter can be done using the forwards-filtering backwards-sampling (Section 8.3.7).

26 For details, see [Sco02].

27

28

29 12.3.8 Collapsed Gibbs sampling

30 We can sometimes gain even greater speedups by analytically integrating out some of the unknown
31 quantities. This is called a **collapsed Gibbs sampler**, and it tends to be more efficient, since it is
32 sampling in a lower dimensional space, which results in lower variance, as discussed in Section 11.6.1.
33 It can also be a useful stepping stone for building samplers for “infinite” models, as we discuss in
34 Chapter 33. We give some examples below.

35

36

37 12.3.8.1 Example: collapsed Gibbs for GMMs

38

39 Consider a GMM with a fully conjugate prior. This can be represented as a PGM-D as shown in
40 Figure 12.9a. Since the prior is conjugate, we can analytically integrate out the model parameters μ_k ,
41 Σ_k and π , so the only remaining hidden variables are the discrete indicator variables z . However,
42 once we integrate out π , all the z_i nodes become inter-dependent. Similarly, once we integrate out
43 $\theta_k = (\mu_k, \Sigma_k)$, all the x_i nodes become inter-dependent, as shown in Figure 12.9b. Nevertheless, we
44 can easily compute the full conditionals, and hence implement a Gibbs sampler.

45

46

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

$$p(z_i = k | \mathbf{z}_{-i}, \mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \propto p(z_i = k | \mathbf{z}_{-i}, \boldsymbol{\alpha}, \boldsymbol{\beta}) p(\mathbf{x} | z_i = k, \mathbf{z}_{-i}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \quad (12.53)$$

$$\propto p(z_i = k | \mathbf{z}_{-i}, \boldsymbol{\alpha}) p(\mathbf{x}_i | \mathbf{x}_{-i}, z_i = k, \mathbf{z}_{-i}, \boldsymbol{\beta}) \quad (12.54)$$

$$\propto p(z_i = k | \mathbf{z}_{-i}, \boldsymbol{\alpha}) p(\mathbf{x}_i | \mathbf{x}_{-i}, z_i = k, \mathbf{z}_{-i}, \boldsymbol{\beta}) \quad (12.55)$$

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

the above expression, we remove \mathbf{x}_i 's statistics from its current cluster (namely z_i), and then evaluate \mathbf{x}_i under each cluster's posterior predictive distribution. Once we have picked a new cluster, we add \mathbf{x}_i 's statistics to this new cluster.

Some pseudo-code for one step of the algorithm is shown in Algorithm 13, based on [Sud06, p94]. (We update the nodes in random order to improve the mixing time, as suggested in [RS97b].) We 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 the naive sampler and collapsed sampler take $O(NKD)$ time per step.

Algorithm 13: Collapsed Gibbs sampler for a mixture model

```

1  for each  $i = 1 : N$  in random order do
2    Remove  $\mathbf{x}_i$ 's sufficient statistics from old cluster  $z_i$  ;
3    for each  $k = 1 : K$  do
4      └ Compute  $p_k(\mathbf{x}_i|\boldsymbol{\beta}) = p(\mathbf{x}_i|\{\mathbf{x}_j : z_j = k, j \neq i\}, \boldsymbol{\beta})$  ;
5      Compute  $p(z_i = k|\mathbf{z}_{-i}, \alpha) \propto (N_{k,-i} + \alpha/K)p_k(\mathbf{x}_i)$ ;
6      Sample  $z_i \sim p(z_i|\cdot)$  ;
7      Add  $\mathbf{x}_i$ 's sufficient statistics to new cluster  $z_i$ 

```

A comparison of this method with the standard Gibbs sampler is shown in Figure 12.10. The vertical axis is the data log probability at each iteration, computed using

$$\log p(\mathcal{D}|\mathbf{z}, \boldsymbol{\theta}) = \sum_{i=1}^N \log [\pi_{z_i} p(\mathbf{x}_i|\boldsymbol{\theta}_{z_i})] \quad (12.60)$$

(To compute this quantity using the collapsed sampler, we have to sample $\boldsymbol{\theta} = (\boldsymbol{\pi}, \boldsymbol{\theta}_{1:K})$ given the data and the current assignment \mathbf{z} .) We see that the collapsed sampler does indeed generally work better than the vanilla sampler.

However, the primary advantage of using the collapsed sampler is that it extends to the case where we have an “infinite” number of mixture components, i.e., where we use a non-parametric Bayesian formulation with a prior such as the **Dirichlet process**. For details, see Section 33.2.4.1.

12.3.8.2 Example: Collapsed Gibbs sampling for LDA

In the supplementary material, we give an example of collapsed Gibbs sampling for fitting the **latent Dirichlet allocation (LDA)** model of [BNJ03a].

12.4 Auxiliary variable MCMC

Sometimes we can dramatically improve the efficiency of sampling by introducing dummy **auxiliary variables**, in order to reduce correlation between the original variables. If the original variables 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 $p(\mathbf{x}, \mathbf{v})$ is easier to sample from than just $p(\mathbf{x})$. If we meet these two conditions, we can sample in the enlarged model, and then throw away the sampled \mathbf{v} values, thereby recovering samples from $p(\mathbf{x})$. Annealed importance sampling (Section 11.5.4) is one example. We give some MCMC examples below.

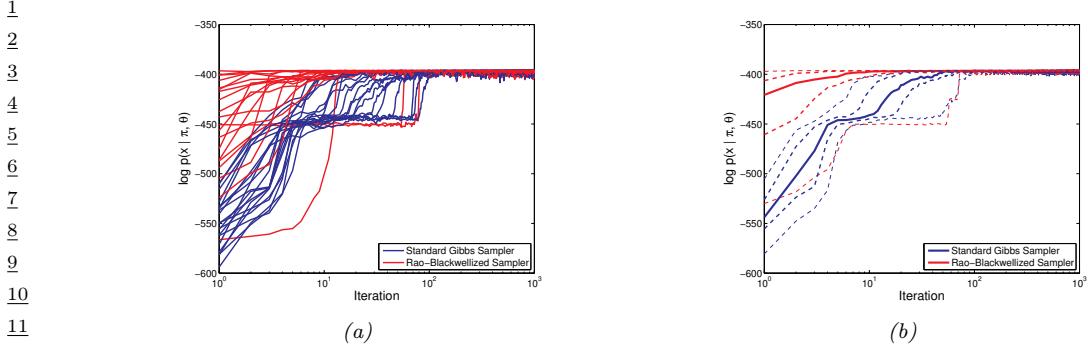


Figure 12.10: Comparison of collapsed (red) and vanilla (blue) Gibbs sampling for a mixture of $K = 4$ two-dimensional Gaussians applied to $N = 300$ data points. We plot log probability of the data vs iteration. (a) 20 different random initializations. (b) logprob averaged over 100 different random 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.

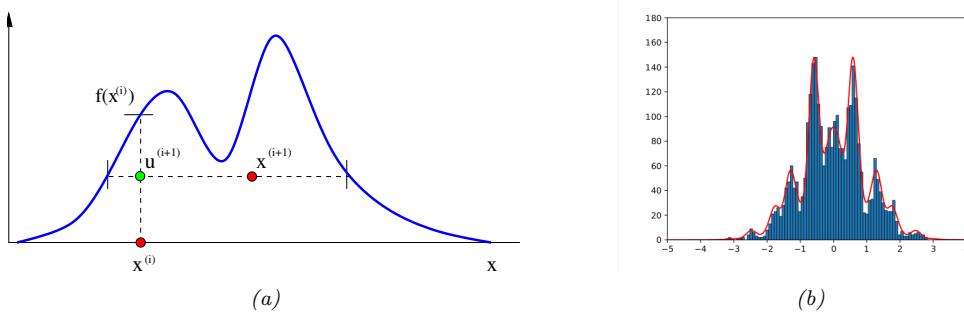


Figure 12.11: Slice sampling. (a) Illustration of one step of the algorithm in 1d. Given a previous sample x^i , we sample u^{i+1} uniformly on $[0, f(x^i)]$, where $f = \tilde{p}$ is the (unnormalized) target density. We then sample x^{i+1} along the slice where $f(x) \geq u^{i+1}$. From Figure 15 of [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

$$\int \hat{p}(x, v)dv = \int_0^{\tilde{p}(x)} \frac{1}{Z_p} dv = \frac{\tilde{p}(x)}{Z_p} = p(x) \quad (12.62)$$

1
2
3
4
5
6
7
8
9
10
11
12
13
14

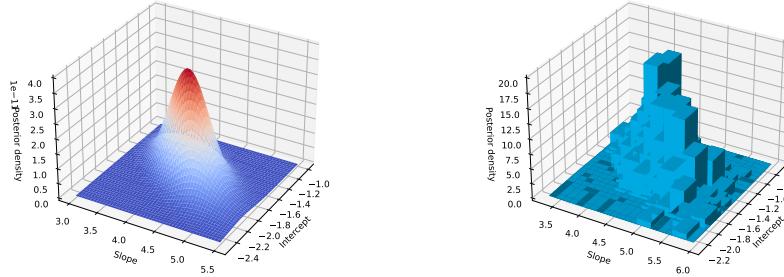


Figure 12.12: Binomial regression for 1d data. Left: Grid approximation to posterior. Right: Slice sampling approximation. Generated by [slice_sampling_demo_2d.py](#).

15
16
17
18
19

20 so we can sample from $p(x)$ by sampling from $\hat{p}(x, v)$ and then ignoring v .

21 We can sample from $\hat{p}(x, v)$ using Gibbs sampling. The full conditionals have the form
22

$$23 \quad p(v|x) = U_{[0, \hat{p}(x)]}(v) \quad (12.63)$$

$$24 \quad p(x|v) = U_A(x) \quad (12.64)$$

25

26 where $A = \{x : \hat{p}(x) \geq v\}$ is the set of points on or above the chosen height v . This corresponds to a
27 slice through the distribution, hence the term **slice sampling** [Nea03]. See Figure 12.11(a).

28 In practice, it can be difficult to identify the set A . So we can use the following approach: construct
29 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
30 end point lies within the slice. If it does, we keep extending in that direction until it lies outside the
31 slice. This is called **stepping out**. A candidate value x' is then chosen uniformly from this region.
32 If it lies within the slice, it is kept, so $x^{s+1} = x'$. Otherwise we shrink the region such that x' forms
33 one end and such that the region still contains x^s . Then another sample is drawn. We continue in
34 this way until a sample is accepted.

35 To apply the method to multivariate distributions, we can sample one extra auxiliary variable for
36 each dimension. The advantage of slice sampling over Gibbs is that it does not need a specification
37 of the full-conditionals, just the unnormalized joint. The advantage of slice sampling over MH is that
38 it does not need a user-specified proposal distribution (although it does require a specification of the
39 width of the stepping out interval).

40 Figure 12.11(b) illustrates the algorithm in action on a synthetic 1d problem. Figure 12.12 illustrates
41 its behavior on a slightly harder problem, namely binomial logistic regression. The model has the form
42 $y_i \sim \text{Bin}(n_i, \text{logit}(\beta_1 + \beta_2 x_i))$. We use a vague Gaussian prior for the β_j 's. Figure 12.12(a) shows a
43 grid-based approximation to the posterior, and Figure 12.12(b) shows a sample-based approximation.
44 In this example, the grid is faster to compute, but for any problem with more than 2 dimensions, the
45 grid approach is infeasible.

46

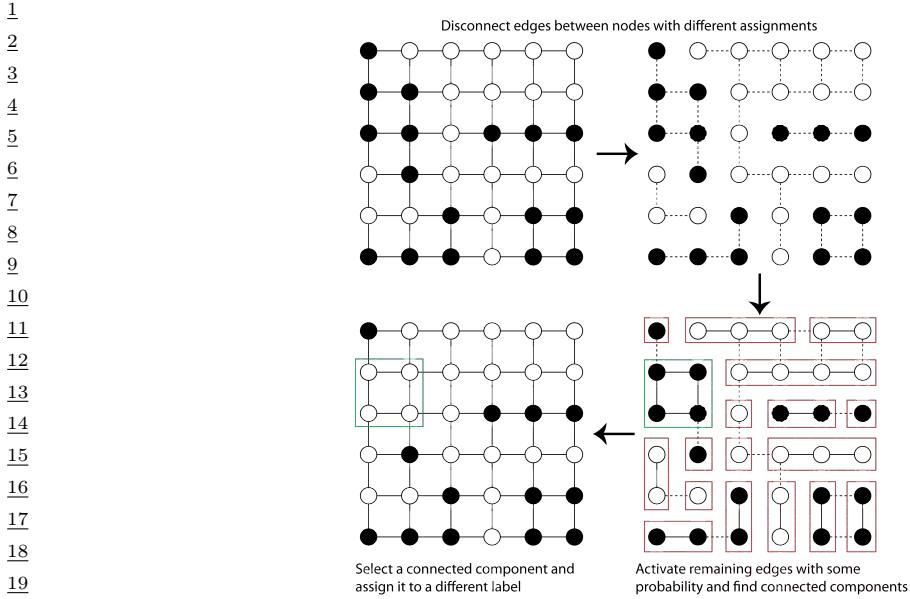


Figure 12.13: Illustration of the Swendsen Wang algorithm on a 2d grid. Used with kind permission of Kevin Tang.

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 Many MCMC algorithms perform poorly in high dimensional spaces, because they rely on a form
²⁸ of random search based on local perturbations. In this section, we discuss a method known as
²⁹ **Hamiltonian Monte Carlo** or **HMC**, that leverages gradient information to guide the local moves.
³⁰ This is an auxiliary variable method (Section 12.4) deriving from physics [Dua+87; Nea93; Mac03;
³¹ Nea10; Bet17].⁵ In particular, the method builds on **Hamiltonian mechanics**, which we describe
³² below.

³³

³⁴ 12.5.1 Hamiltonian mechanics

³⁵

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 5. The method was originally called **hybrid MC** [Dua+87]. It was introduced to the statistics community in [Nea93],
⁴³ and was renamed to Hamiltonian MC in [Mac03].

⁴⁴

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

$$\theta_{t+1} = \theta_t + \eta \Sigma^{-1} v_{t+1/2} \quad (12.73)$$

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:

$$v_{t+1} = v_t + \eta \frac{dv}{dt}(\theta_t, v_t) = v_t - \eta \frac{\partial \mathcal{E}(\theta_t)}{\partial \theta} \quad (12.74)$$

$$\theta_{t+1} = \theta_t + \eta \frac{d\theta}{dt}(\theta_t, v_{t+1}) = \theta_t + \eta \frac{\partial \mathcal{K}(v_{t+1})}{\partial v} \quad (12.75)$$

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:

$$v_{t+1/2} = v_t - \frac{\eta}{2} \frac{\partial \mathcal{E}(\theta_t)}{\partial \theta} \quad (12.76)$$

$$\theta_{t+1} = \theta_t + \eta \frac{\partial \mathcal{K}(v_{t+1/2})}{\partial v} \quad (12.77)$$

$$v_{t+1} = v_{t+1/2} - \frac{\eta}{2} \frac{\partial \mathcal{E}(\theta_{t+1})}{\partial \theta} \quad (12.78)$$

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 v at the beginning and end of the trajectory, and alternating
³⁵ between full step updates of θ and v in between.

³⁶ It can be shown that the leapfrog integrator is volume preserving. Furthermore, if we reverse
³⁷ the momentum at the end (by replacing it with $-v$), the method is also reversible. However, if
³⁸ the kinetic energy satisfies $\mathcal{K}(v) = \mathcal{K}(-v)$, this reversal is not needed. Unfortunately, this method
³⁹ does not exactly conserve energy, due to the finite step size. We will correct for this by treating the
⁴⁰ method as a proposal distribution, and using the MH acceptance criterion to ensure we sample from
⁴¹ the right distribution, as we discuss in Section 12.5.3.
⁴²

⁴³ 12.5.2.4 Higher order integrators ⁴⁴

⁴⁵ It is possible to define higher order integrators that are more accurate, but take more steps. For
⁴⁶ details, see [BRSS18].
⁴⁷

12.5.3 The HMC algorithm

We now describe how to use Hamiltonian dynamics to define an MCMC sampler in the expanded state space $(\boldsymbol{\theta}, \mathbf{v})$. The target distribution has the form

$$p(\boldsymbol{\theta}, \mathbf{v}) = \frac{1}{Z} \exp[-\mathcal{H}(\boldsymbol{\theta}, \mathbf{v})] = \frac{1}{Z} \exp \left[-\mathcal{E}(\boldsymbol{\theta}) - \frac{1}{2} \mathbf{v}^\top \boldsymbol{\Sigma} \mathbf{v} \right] \quad (12.79)$$

The marginal distribution over the latent variables of interest has the form

$$p(\boldsymbol{\theta}) = \int p(\boldsymbol{\theta}, \mathbf{v}) d\mathbf{v} = \frac{1}{Z_q} e^{-\mathcal{E}(\boldsymbol{\theta})} \int \frac{1}{Z_p} e^{-\frac{1}{2} \mathbf{v}^\top \boldsymbol{\Sigma} \mathbf{v}} d\mathbf{v} = \frac{1}{Z_q} e^{-\mathcal{E}(\boldsymbol{\theta})} \quad (12.80)$$

Suppose the previous state of the Markov chain is $(\boldsymbol{\theta}_{t-1}, \mathbf{v}_{t-1})$. To sample the next state, we 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 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 steps to get the final proposed state $(\boldsymbol{\theta}^*, \mathbf{v}^*) = (\boldsymbol{\theta}_L, \tilde{\mathbf{v}}_L)$. Next we compute the MH acceptance probability

$$\alpha = \min \left(1, \frac{p(\boldsymbol{\theta}^*, \mathbf{v}^*)}{p(\boldsymbol{\theta}_{t-1}, \mathbf{v}_{t-1})} \right) = \min (1, \exp [-\mathcal{H}(\boldsymbol{\theta}^*, \mathbf{v}^*) + \mathcal{H}(\boldsymbol{\theta}_{t-1}, \mathbf{v}_{t-1})]) \quad (12.81)$$

(The transition probabilities cancel since the proposal is reversible.) Finally, we accept the proposal 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 we don't need to keep the momentum term, it is only used inside of the leapfrog algorithm.) See Algorithm 14 for the pseudocode.

Algorithm 14: Hamiltonian Monte Carlo

```

1 for t = 1 : T do
2   Generate random momentum  $\mathbf{v}_{t-1} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$  ;
3   Set  $(\boldsymbol{\theta}'_0, \mathbf{v}'_0) = (\boldsymbol{\theta}_{t-1}, \mathbf{v}_{t-1})$  ;
4   Half step for momentum:  $\mathbf{v}'_{\frac{1}{2}} = \mathbf{v}'_0 - \frac{\eta}{2} \nabla \mathcal{E}(\boldsymbol{\theta}'_0)$  ;
5   for l = 1 : L - 1 do
6      $\boldsymbol{\theta}'_l = \boldsymbol{\theta}'_{l-1} + \eta \boldsymbol{\Sigma}^{-1} \mathbf{v}'_{l-1/2}$  ;
7      $\mathbf{v}'_{l+1/2} = \mathbf{v}'_{l-1/2} - \eta \nabla \mathcal{E}(\boldsymbol{\theta}'_l)$ 
8   Full step for location:  $\boldsymbol{\theta}'_L = \boldsymbol{\theta}'_{L-1} + \eta \boldsymbol{\Sigma}^{-1} \mathbf{v}'_{L-1/2}$  ;
9   Half step for momentum:  $\mathbf{v}'_L = \mathbf{v}'_{L-1/2} - \frac{\eta}{2} \nabla \mathcal{E}(\boldsymbol{\theta}'_L)$  ;
10  Compute proposal  $(\boldsymbol{\theta}^*, \mathbf{v}^*) = (\boldsymbol{\theta}'_L, \mathbf{v}'_L)$  ;
11  Compute  $\alpha = \min (1, \exp [-\mathcal{H}(\boldsymbol{\theta}^*, \mathbf{v}^*) + \mathcal{H}(\boldsymbol{\theta}_{t-1}, \mathbf{v}_{t-1})])$  ;
12  Set  $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}^*$  with probability  $\alpha$ , otherwise  $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t$ .

```

We need to sample a new momentum at each iteration to satisfy ergodicity. To see why, recall that $\mathcal{H}(\boldsymbol{\theta}, \mathbf{v})$ stays approximately constant as we move through phase space. If $\mathcal{H}(\boldsymbol{\theta}, \mathbf{v}) = \mathcal{E}(\boldsymbol{\theta}) + \frac{1}{2} \mathbf{v}^\top \boldsymbol{\Sigma} \mathbf{v}$, 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 reach states where $\mathcal{E}(\boldsymbol{\theta}) > h$. To ensure the sampler explores the full space, we must pick a random momentum at the start of each iteration.

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

¹³ **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].

²⁶

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:

³¹

$$\Sigma(\boldsymbol{\theta}) = \nabla^2 \mathcal{E}(\boldsymbol{\theta}) \tag{12.82}$$

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

³⁴

$$\Sigma(\boldsymbol{\theta}) = -\mathbb{E}_{p(\mathbf{x}|\boldsymbol{\theta})} [\nabla^2 \log p(\mathbf{x}|\boldsymbol{\theta})] \tag{12.83}$$

³⁵ Since the initial momentum is sampled from $\mathcal{N}(\mathbf{0}, \Sigma)$, this ensures that the noise that we add to
³⁶ the system follows the local covariance structure, taking bigger moves along “flat” directions, thus
³⁷ exploring along valley floors, where there is highest posterior uncertainty.

³⁸ One way to estimate a fixed Σ is to run HMC with $\Sigma = \mathbf{I}$ for a **warmup** period, and then to run for
³⁹ a few more steps, so we can compute the empirical covariance matrix using $\Sigma = \mathbb{E}[(\boldsymbol{\theta} - \bar{\boldsymbol{\theta}})(\boldsymbol{\theta} - \bar{\boldsymbol{\theta}})^T]$.
⁴⁰ Another approach is to use an L-BFGS [ZS11]. In [Hof+19] they propose a method called **NeuTra**
⁴¹ **HMC** algorithm which “neutralizes” bad geometry by learning an inverse autoregressive flow model
⁴² (Section 24.2.4.3) in order to map the warped distribution to an isotropic Gaussian. This is often an
⁴³ order of magnitude faster than vanilla HMC.

⁴⁴

12.5.5 Riemann Manifold HMC

If we let the covariance matrix change as we move position, so Σ is a function of θ , the method is known as **Riemann Manifold HMC** [GC11; Bet13], since the moves follow a curved manifold, rather than the flat manifold induced by a constant Σ . In the RM-HMC case, the kinetic energy has to be generalized to

$$\mathcal{K}(\theta, v) = \frac{1}{2} \log((2\pi)^D |\Sigma(\theta)|) + \frac{1}{2} v^\top \Sigma(\theta) v \quad (12.84)$$

Unfortunately the Hamiltonian updates of θ and v are no longer separable, making the RM-HMC algorithm more complex (e.g., it requires a fixed point iteration and third order derivatives).

A simplification to this method is to update Σ at each step of the outer loop, but to keep it constant during the leapfrog integration inner loop. In this case, we can use the standard HMC method. (Updating Σ between proposals does not violate the Markov property, since the momentum terms are not shared across outer loop steps, and the inner loop is reversible for any value of Σ .)

Since estimating a high dimensional covariance matrix can be computationally and statistically inefficient, it is more common to reparameterize the problem in terms of $\Sigma = \Sigma^{\top/2} \Sigma^{\frac{1}{2}}$, where $\Sigma^{\frac{1}{2}}$ is a 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 $\Lambda^{\frac{1}{2}} = \text{diag}(\sqrt{\lambda_d})$ is a diagonal matrix with the square root of the eigenvalues.)

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 we modify the leapfrog steps as follows:

$$v_{t+1/2} = v_t - \frac{\eta}{2} \Sigma^{-\frac{1}{2}} \nabla \mathcal{E}(\theta_t) \quad (12.85)$$

$$\theta_{t+1} = \theta_t + \eta \Sigma^{-\frac{1}{2}} v_{t+1/2} \quad (12.86)$$

$$v_{t+1} = v_{t+1/2} - \frac{\eta}{2} \Sigma^{-\frac{1}{2}} \nabla \mathcal{E}(\theta_{t+1}) \quad (12.87)$$

The advantage of this formulation is that it is often easier to directly estimate $\Sigma^{-\frac{1}{2}}$ using methods such as [ZS11]. Furthermore, if we use a diagonal approximation, we can avoid all matrix inversions.

12.5.6 Langevin Monte Carlo (MALA)

A special case of HMC occurs when we take $L = 1$ leapfrog steps. This is known as **Langevin Monte Carlo (LMC)**, or **Metropolis Adjusted Langevin Algorithm (MALA)** [RT96]. The inner loop now consists of the following steps:

- Sample momentum $v'_0 \sim \mathcal{N}(\mathbf{0}, \Sigma)$
- Set $\theta'_0 = \theta_{t-1}$
- Take half step for momentum: $v'_{\frac{1}{2}} = v'_0 - \frac{\eta}{2} \nabla \mathcal{E}(\theta'_0)$.
- Take full step for location: $\theta'_1 = \theta'_0 + \eta \Sigma^{-1} v'_{\frac{1}{2}}$.
- Take half step for momentum: $v'_1 = v'_{\frac{1}{2}} - \frac{\eta}{2} \nabla \mathcal{E}(\theta'_1)$.
- Propose $(\theta^*, v^*) = (\theta'_1, v'_1)$.
- Perform MH update.

We can simplify the 3 equations into 2 by substituting $v'_{\frac{1}{2}}$ directly into θ'_1 and v'_1 . This gives rise to the simplified algorithm shown in Algorithm 15.

1

2 **Algorithm 15:** Langevin Monte Carlo

3 **for** $t = 1 : T$ **do**

4 2 Generate random momentum $\mathbf{v}_{t-1} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$;

5 3 $\boldsymbol{\theta}^* = \boldsymbol{\theta}_{t-1} - \frac{\eta^2}{2} \boldsymbol{\Sigma}^{-1} \nabla \mathcal{E}(\boldsymbol{\theta}_{t-1}) + \eta \boldsymbol{\Sigma}^{-1} \mathbf{v}_{t-1}$;

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 $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t$.

10

11

12 LMC has a different properties than “full” HMC: since the momenta are discarded after each step,

13 successive proposals are not encouraged to move in the same direction, and so LMC explores the

14 landscape in a random walk fashion (albeit guided by the gradient). On the other hand, the method

15 is simpler than HMC.

16 A further simplification of LMC is to eliminate the MH acceptance step. In this case, the update

17 becomes

19
$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \frac{\eta^2}{2} \boldsymbol{\Sigma}^{-1} \nabla \mathcal{E}(\boldsymbol{\theta}_{t-1}) + \eta \boldsymbol{\Sigma}^{-1} \mathbf{v}_{t-1} \quad (12.88)$$

21
$$= \boldsymbol{\theta}_{t-1} - \frac{\eta^2}{2} \boldsymbol{\Sigma}^{-1} \nabla \mathcal{E}(\boldsymbol{\theta}_{t-1}) + \eta \sqrt{\boldsymbol{\Sigma}^{-1}} \boldsymbol{\epsilon}_{t-1} \quad (12.89)$$

23 where $\mathbf{v}_{t-1} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ and $\boldsymbol{\epsilon}_{t-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. This is just like gradient descent with added noise. If we

24 set $\boldsymbol{\Sigma}$ to be the Fisher information matrix, this becomes natural gradient descent (Section 6.4) with

25 added noise. If we approximate the gradient with a stochastic gradient, we get a method known as

26 SGLD, or Stochastic Gradient Langevin Descent (see Section 12.7.1 for details).

27 Now suppose $\boldsymbol{\Sigma} = \mathbf{I}$, and we set $\eta = \sqrt{2}$. In continuous time, we get the following stochastic

28 differential equation (SDE), known as **Langevin diffusion**:

30
$$d\boldsymbol{\theta}_t = -\nabla \mathcal{E}(\boldsymbol{\theta}_t) dt + \sqrt{2} d\mathbf{B}_t \quad (12.90)$$

32 where \mathbf{B}_t represents D -dimensional **Brownian motion**. If we use this to generate the samples, the

33 method is known as the **Unadjusted Langevin Algorithm** or **ULA** [Par81]. It can be shown that

34 this process has $\pi(\boldsymbol{\theta})$ as its stationary distribution [RT96].

36 **12.5.7 Connection between SGD and Langevin sampling**

38 In this section, we discuss a deep connection between stochastic gradient descent (SGD) and Langevin

39 sampling, following the presentation of [BZ20, Sec 10.2.3].

41 Consider the minimization of the additive loss

42
$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{n=1}^N \mathcal{L}_n(\boldsymbol{\theta}) \quad (12.91)$$

45 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

1
2 the gradients:

3
4 $\nabla_B \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{B} \sum_{n \in \mathcal{S}} \nabla \mathcal{L}_n(\boldsymbol{\theta})$ 5 (12.92)

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 $\mathbf{v}_t = \sqrt{\eta} (\nabla \mathcal{L}(\boldsymbol{\theta}_t) - \nabla_B \mathcal{L}(\boldsymbol{\theta}_t))$ 10 (12.93)

11 Then we can rewrite the SGD update as

12 $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_B \mathcal{L}(\boldsymbol{\theta}_t) = \boldsymbol{\theta}_t - \eta \nabla \mathcal{L}(\boldsymbol{\theta}_t) + \sqrt{\eta} \mathbf{v}_t$ 13 (12.94)

14 The **diffusion term** \mathbf{v}_t has mean 0, since

15
16 $\mathbb{E} [\nabla_B \mathcal{L}(\boldsymbol{\theta})] = \frac{1}{B} \sum_{j=1}^B \mathbb{E} [\nabla \mathcal{L}_{i_j}(\boldsymbol{\theta})] = \frac{1}{B} \sum_{j=1}^B \nabla \mathcal{L}(\boldsymbol{\theta}) = \nabla \mathcal{L}(\boldsymbol{\theta})$ 17 (12.95)

19 To compute the variance of the diffusion term, note that

20
21 $\mathbb{V} [\nabla_B \mathcal{L}(\boldsymbol{\theta})] = \frac{1}{B^2} \sum_{j=1}^B \mathbb{V} [\nabla \mathcal{L}_{i_j}(\boldsymbol{\theta})]$ 22 (12.96)

23 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)

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 $d\boldsymbol{\theta}(t) = -\nabla \mathcal{L}(\boldsymbol{\theta}(t)) dt + \sqrt{\frac{\eta}{B} \mathbf{D}(\boldsymbol{\theta}_t)} d\mathbf{B}(t)$ 35 (12.99)

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 $\mathbf{D}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N \nabla \mathcal{L}_n(\boldsymbol{\theta}) \nabla \mathcal{L}_n(\boldsymbol{\theta})^\top \approx \mathbb{E} [\nabla \mathcal{L}_n(\boldsymbol{\theta}) \nabla \mathcal{L}_n(\boldsymbol{\theta})^\top] = \mathbf{F}(\boldsymbol{\theta})$ 46 (12.100)

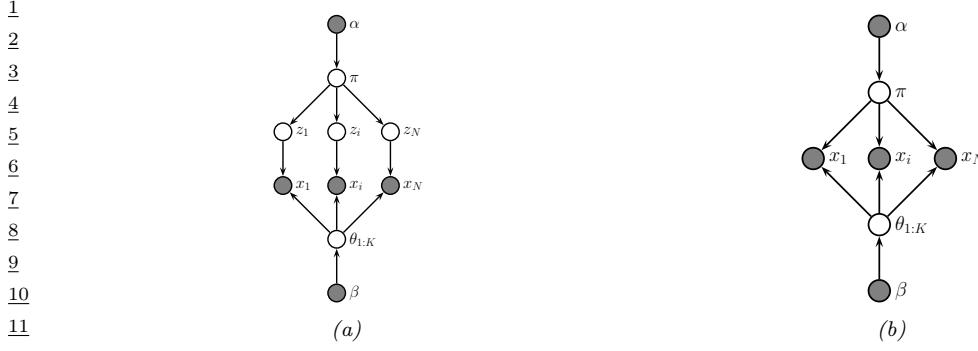


Figure 12.14: (a) A mixture model. (b) After integrating out the discrete latent variables.

From Equation (12.99), we see that SGD adds a noise term of the form $\sqrt{\mathbf{F}}\boldsymbol{\epsilon}$, while from Equation (12.90), we see that Langevin MC adds a noise term of the form $\sqrt{\mathbf{F}^{-1}}\boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. 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.

1 models. For example, to apply HMC to HMMs, we can use the forwards algorithm (Section 8.3.1) to
2 efficiently compute
3

$$\sum_{\mathbf{z}_{1:T}} p(\mathbf{x}_n | \boldsymbol{\theta}) = \sum_{\mathbf{z}_{1:T}} p(\mathbf{x}_n, \mathbf{z}_{n,1:T} | \boldsymbol{\theta}) \quad (12.102)$$

4
5 In addition to marginalizing out any discrete latent variables, we need to ensure the remaining
6 continuous latent variables are unconstrained. This often requires performing a change of variables
7 using a bijector. For example, instead of sampling the discrete probability vector from the probability
8 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
9 bijectors are invertible.
10

11 12.5.9 Speeding up HMC

12 Although HMC uses gradient information to explore the typical set, sometimes the geometry of the
13 typical set can be difficult to sample from. Recently [Hof+19] proposed the **NeuTra** HMC algorithm
14 which “neutralizes” bad geometry by learning an inverse autoregressive flow model (Section 24.2.4.3)
15 in order to map the warped distribution to an isotropic Gaussian. This is often an order of magnitude
16 faster than vanilla HMC.

17 An orthogonal issue to the geometry of the space is the cost of evaluating the target distri-
18 bution, $\mathcal{E}(\boldsymbol{\theta}) = -\log \tilde{p}(\boldsymbol{\theta})$. For many ML applications, this has the form $\log \tilde{p}(\boldsymbol{\theta}) = \log p_0(\boldsymbol{\theta}) +$
19 $\sum_{n=1}^N \log p(\boldsymbol{\theta}_n | \boldsymbol{\theta})$. This takes $O(N)$ time to compute. We can speed this up by subsampling the
20 data, although this introduces noise into the system. See Section 12.7 for details.
21

22 S

23 12.6 MCMC convergence

24 We start MCMC from an arbitrary initial state. As we explained in Section 2.8.4, the samples will be
25 coming from the chain’s stationary distribution only when the chain has “forgotten” where it started
26 from. The amount of time it takes to enter the stationary distribution is called the mixing time (see
27 Section 12.6.1 for details). Samples collected before the chain has reached its stationary distribution
28 do not come from p^* , and are usually thrown away. The initial period, whose samples will be ignored,
29 is called the **burn-in phase**.

30 For example, consider a uniform distribution on the integers $\{0, 1, \dots, 20\}$. Suppose we sample
31 from this using a symmetric random walk. In Figure 12.15, we show two runs of the algorithm. On
32 the left, we start in state 10; on the right, we start in state 17. Even in this small problem it takes
33 over 200 steps until the chain has “forgotten” where it started from. Proposal distributions that
34 make larger changes can converge faster. For example, [BD92; Man] prove that it takes about 7 riffle
35 shuffles to properly mix a deck of 52 cards (i.e., to ensure the distribution is uniform).

36 In Section 12.6.1 we discuss how to compute the mixing time theoretically. In practice, this can be
37 very hard [BBM10] (this is one of the fundamental weaknesses of MCMC), so in Section 12.6.2, we
38 discuss practical heuristics.

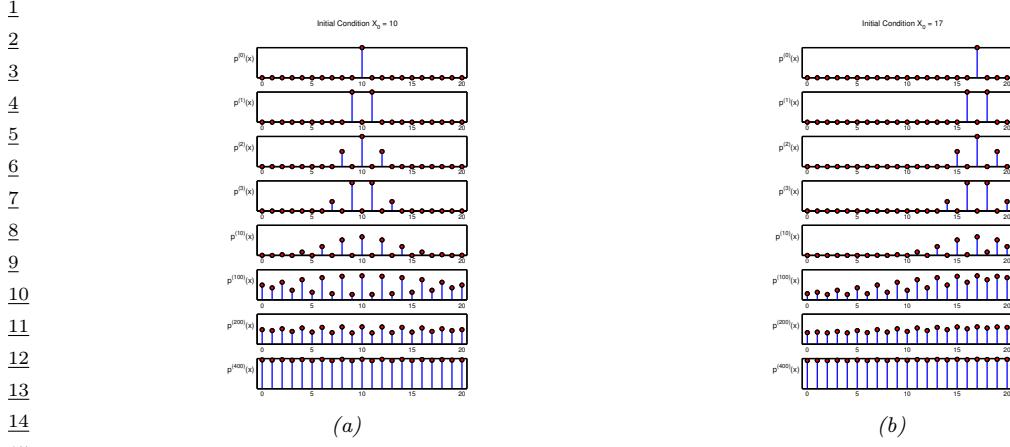


Figure 12.15: Illustration of convergence to the uniform distribution over $\{0, 1, \dots, 20\}$ using a symmetric random walk starting from (left) state 10, and (right) state 17. Adapted from Figures 29.14 and 29.15 of [Mac03]. Generated by `random_walk_integers.py`.

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

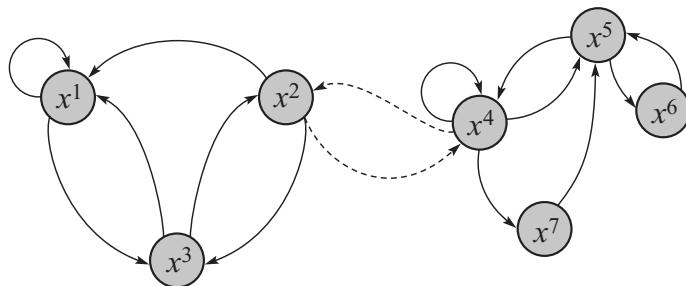
$$\tau_\epsilon \triangleq \max_{x_0} \tau_\epsilon(x_0) \quad (12.104)$$

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:

$$\phi \triangleq \min_{S: 0 \leq p^*(S) \leq 0.5} \frac{\sum_{x \in S, x' \in S^c} T(x \rightarrow x')}{p^*(S)}, \quad (12.105)$$

1
2
3
4
5
6
7
8
9
10



11
12
13

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.

14
15
16
17
18
19
20

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]).

21
22
23
24
25
26
27
28
29
30
31

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.

32

12.6.2.1 Trace plots

33
34
35
36
37
38
39

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.

40
41
42
43
44
45
46
47

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.

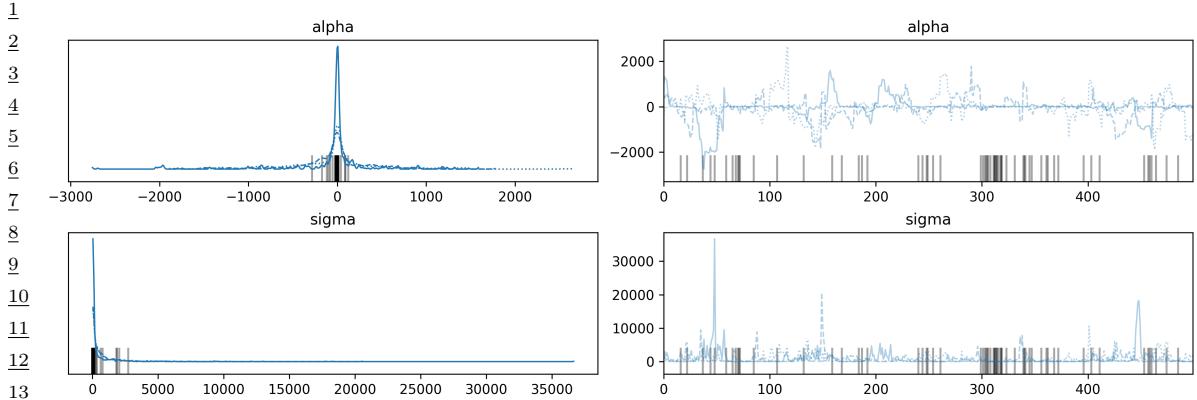


Figure 12.17: Marginals (left) and trace plot (right) for the univariate Gaussian using the diffuse prior. Black vertical lines indicate HMC divergences. Adapted from Figures 9.9–9.10 of [McE20]. Generated by `mcmc_traceplots_unigauss_numpyro.py`.

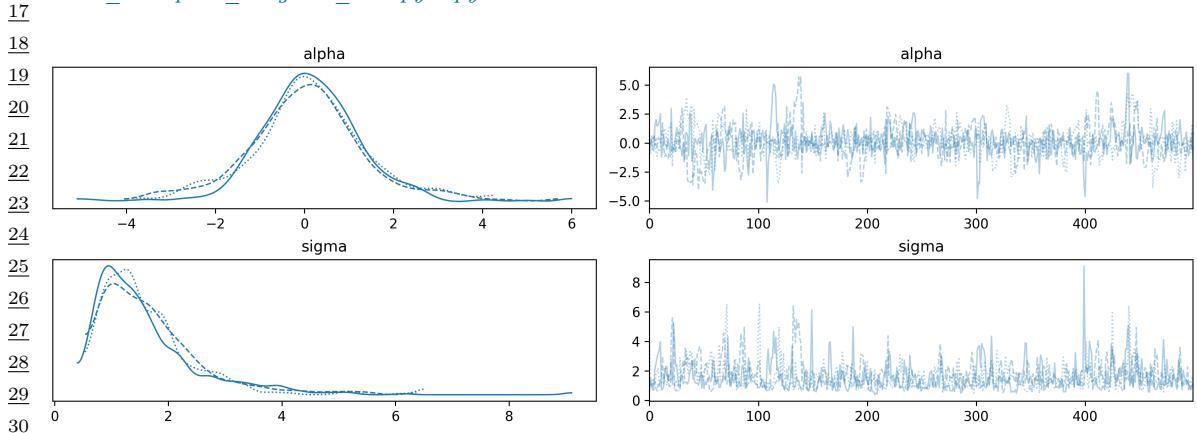


Figure 12.18: Marginals (left) and trace plot (right) for the univariate Gaussian using the sensible prior. Adapted from Figures 9.9–9.10 of [McE20]. Generated by `mcmc_traceplots_unigauss_numpyro.py`.

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

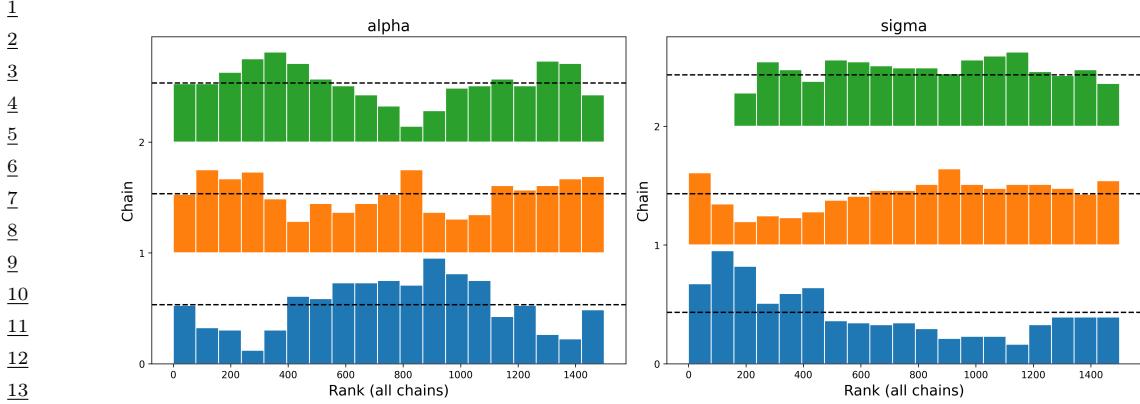


Figure 12.19: Trace rank plot for the univariate Gaussian using the diffuse prior. Black vertical lines indicate HMC divergences. Adapted from Figures 9.9–9.10 of [McE20]. Generated by `mcmc_traceplots_unigauss_numpyro.py`.

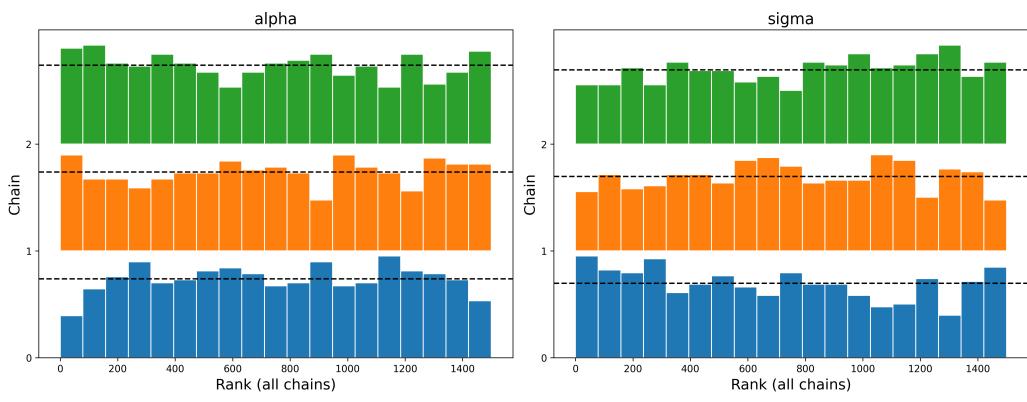


Figure 12.20: Trace rank plot for the univariate Gaussian using the sensible prior. Adapted from Figures 9.9–9.10 of [McE20]. Generated by `mcmc_traceplots_unigauss_numpyro.py`.

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:

$$\frac{7}{8} \quad B = \frac{N}{M-1} \sum_{m=1}^M (\bar{x}_{\cdot m} - \bar{x}_{\cdot \cdot})^2, \text{ where } \bar{x}_{\cdot m} = \frac{1}{N} \sum_{n=1}^N x_{nm}, \quad \bar{x}_{\cdot \cdot} = \frac{1}{M} \sum_{m=1}^M \bar{x}_{\cdot m} \quad (12.106)$$

$$\frac{11}{12} \quad W = \frac{1}{M} \sum_{m=1}^M s_m^2, \text{ where } s_m^2 = \frac{1}{N-1} \sum_{n=1}^N (x_{nm} - \bar{x}_{\cdot m})^2 \quad (12.107)$$

14 The formula for s_m^2 is the usual unbiased estimate for the variance from a set of N samples; W is
15 just the average of this. The formula for B is similar, but scaled up by N since it is based on the
16 variance of $\bar{x}_{\cdot m}$, which are averaged over N values.

17 Let $\mathbb{V}[x] = \sigma^2$ be the true variance of the quantity of interest under the target distribution. The
18 within-chain variance W will generally underestimate V , because each chain may not have fully
19 explored the target distribution. However, one can show that the following estimate will overestimate
20 V , and hence is a conservative estimate:

$$\frac{22}{23} \quad \hat{V}^+ \triangleq \frac{N-1}{N} W + \frac{1}{N} B \quad (12.108)$$

24 One can also show that $\hat{V}^+ \rightarrow \sigma^2$ as $N \rightarrow \infty$, so this is a consistent estimator.

25 We can estimate the benefits of running the chains for longer by comparing \hat{V}^+ to W . As $N \rightarrow \infty$,
26 this ratio will tend to 1. Thus we define the **estimated potential scale reduction**, also known as
27 **R-hat**, as follows:

$$\frac{29}{30} \quad \hat{R} \triangleq \sqrt{\frac{\hat{V}^+}{W}} \quad (12.109)$$

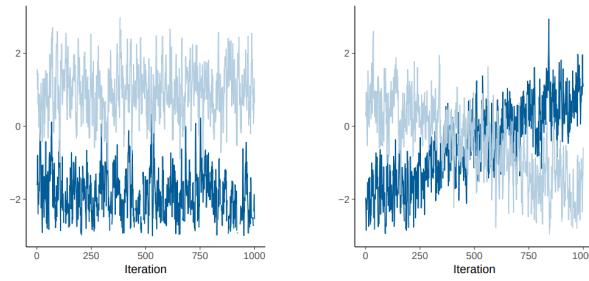
32 In [Veh+19], they recommend checking if $\hat{R} < 1.01$ before declaring convergence.

34 For example, consider the \hat{R} values for various samplers for our univariate GMM example. In
35 particular, consider the 3 MH samplers in Figure 12.1, and the Gibbs sampler in Figure 12.7. The \hat{R}
36 values are 1.493, 1.039, 1.005 and 1.007. So this diagnostic has correctly identified that the first two
37 samplers are unreliable. The third sampler seems to be mixing, but the samples are highly correlated,
38 which reduces the effective sample size; we discuss this in Section 12.6.2.3.

39 Although the above definition can detect non-convergence of the kind shown in Figure 12.21(a),
40 it will fail on non-stationary chains of the kind shown in Figure 12.21(b). For this reason, it is
41 recommended to split each chain in the first and second halves, thus doubling M (but halving N),
42 and then to compute \hat{R} ; this is called the **split- \hat{R}** statistic.

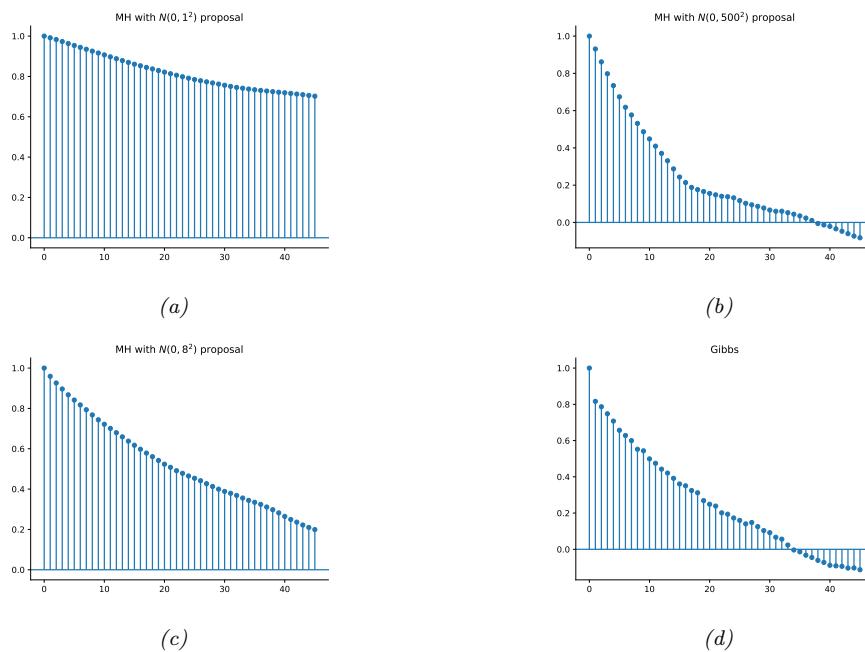
43 In addition, the above definition can fail when estimating quantities which do not have finite means
44 or variances. To combat this, [Veh+19] suggest first computing the ranks r_{nm} of each sample x_{nm} ,
45 and then normalizing them using the inverse of the normal cdf, $z_{nm} = \Phi^{-1}((r_{nm} - 0.5)/S)$, and
46 finally computing the (split) \hat{R} from the z_{nm} values.

1
2
3
4
5
6
7
8
9
10
11
12



13 *Figure 12.21: Examples of two distinct challenges that arise when trying to assess if Markov chain samples*
14 *have converged. (a) Each chain looks stationary, but they have not converged to a common distribution. (b)*
15 *The two sequences cover a common distribution, but are non-stationary. From Figure 1 of [Veh+19]. Used*
16 *with kind permission of Aki Vehtari.*

17
18
19
20
21
22
23
24
25
26
27
28
29



40 *Figure 12.22: Autocorrelation functions for various MCMC samplers for the mixture of two 1D Gaussians.*
41 *(a-c) These are the MH samplers in Figure 12.1. (d) This is the Gibbs sampler in Figure 12.7. Generated by*
42 *mcmc_gmm_demo.py.*

43
44
45
46
47

1 **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

$$\mathbb{V}[\hat{x}] = \frac{1}{N^2} \mathbb{V}\left[\sum_{n=1}^N x_n\right] = \frac{1}{N^2} \sum_{n=1}^N \mathbb{V}[x_n] = \frac{1}{N} \sigma^2 \quad (12.110)$$

9 where $\sigma^2 = \mathbb{V}[X]$. If the samples are correlated, as they usually will be when drawn from a Markov
10 chain, the variance of the estimate will be higher, as we show below.

11 Recall that for N (not necessarily independent) random variables we have

12

$$\mathbb{V}\left[\sum_{n=1}^N x_n\right] = \sum_{i=1}^N \sum_{j=1}^N \text{Cov}[x_i, x_j] = \sum_{i=1}^N \mathbb{V}[x_i] + 2 \sum_{1 \leq i < j \leq N} \text{Cov}[x_i, x_j] \quad (12.111)$$

16 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
17 estimate based on these correlated samples. The variance of this estimate is given by

19

$$\mathbb{V}[\bar{x}] = \frac{1}{N^2} \sum_{i=1}^N \sum_{j=1}^N \text{Cov}[x_i, x_j] \quad (12.112)$$

23 We now try to rewrite this in a more convenient form. First recall that the correlation of x_i and
24 x_j is given by

25

$$\text{corr}[x_i, x_j] = \frac{\text{Cov}[x_i, x_j]}{\sqrt{\mathbb{V}[x_i] \mathbb{V}[x_j]}} \quad (12.113)$$

29 Since we assume we are drawing samples from the target distribution, we have $\mathbb{V}[x_i] = \sigma^2$, and hence

31

$$\mathbb{V}[\bar{x}] = \frac{\sigma^2}{N^2} \sum_{i=1}^N \sum_{j=1}^N \text{corr}[x_i, x_j] \quad (12.114)$$

35 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
36 from i . Hence

37

$$\sum_{j=1}^N \text{corr}[x_i, x_j] \approx \sum_{\ell=-\infty}^{\infty} \text{corr}[x_i, x_{i+\ell}] = 1 + 2 \sum_{\ell=1}^{\infty} \text{corr}[x_i, x_{i+\ell}] \quad (12.115)$$

41 Since we assume the samples are coming from a stationary distribution, the index i does not matter.
42 Thus we can rewrite the above some in terms of the **autocorrelation function**, defined as

43

$$\rho(\ell) \triangleq \text{corr}[x_0, x_\ell] \quad (12.116)$$

45 where ℓ is called the **lag**. The ACF can be computed efficiently by convolving the signal \mathbf{x} with itself.

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

18 12.6.3 Improving speed of convergence

19 There are many possible things you could try if the \hat{R} value is too large, and/or the effective sample
20 size is too low. Here is a brief list:
21

- 22 • Try using a non-centered parameterization (see Section 12.6.4).
 - 23 • Try sampling variables in groups or blocks (see Section 12.3.7).
 - 24 • Try using Rao-Blackwellisation, i.e., analytically integrating out some of the variables (see
25 Section 12.3.8).
 - 26 • Try adding auxiliary variables (see Section 12.4).
 - 27 • Try using adaptive proposal distributions (see Section 12.2.3.5).
- 28

29 More details can be found in [Rob+18].

30 12.6.4 Non-centered parameterizations and Neal's funnel

31 A common problem that arises with hierarchical Bayesian models is when a set of parameters at
32 one level of the model have a tight dependence on parameters at the level above. We saw examples
33 of this in the hierarchical Gaussian 8-schools example in Section 3.5.2.2 and the hierarchical radon
34 regression example in Section 15.5.3.2.

35

36 7. See also <https://bit.ly/2vhA1xa>.

37

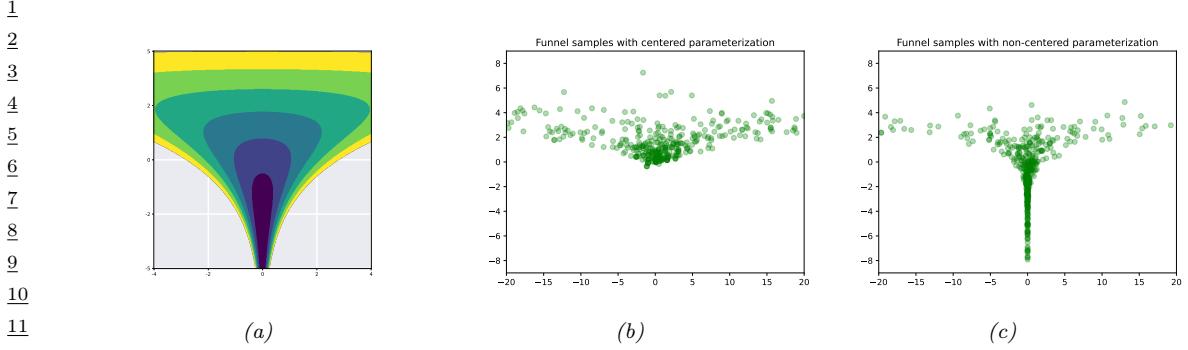


Figure 12.23: Neal’s funnel. (a) Joint density. Generated by [neals_funnel.py](#). (a) HMC samples from centered representation. (b) HMC samples from non-centered representation. Generated by [funnel_numpyro.py](#).

A simple toy model that captures the essence of the problem is the following distribution:

$$\nu \sim \mathcal{N}(0, 3) \tag{12.125}$$

$$x \sim \mathcal{N}(0, \exp(\nu)) \tag{12.126}$$

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

$$\nu \sim \mathcal{N}(0, 3) \tag{12.127}$$

$$z \sim \mathcal{N}(0, 1) \tag{12.128}$$

$$x = z \exp(\nu) \tag{12.129}$$

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:

$$\mu \sim p(\mu) \tag{12.130}$$

$$\sigma \sim p(\sigma) \tag{12.131}$$

$$R \sim \text{LKJ}(R|\eta) \tag{12.132}$$

$$\Sigma = \text{diag}(\sigma) R \text{ diag}(\sigma) \tag{12.133}$$

$$\beta \sim \mathcal{N}(\mu, \Sigma) \tag{12.134}$$

1 We can write the non-centered representation as follows:
2

$$\underline{3} \quad \boldsymbol{\mu} \sim p(\boldsymbol{\mu}) \quad (12.135)$$

$$\underline{4} \quad \boldsymbol{\sigma} \sim p(\boldsymbol{\sigma}) \quad (12.136)$$

$$\underline{5} \quad \mathbf{L} \sim \text{LKJchol}(\mathbf{L}|\eta) \quad (12.137)$$

$$\underline{6} \quad \mathbf{R} = \mathbf{L}\mathbf{L}^\top \quad (12.138)$$

$$\underline{7} \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \quad (12.139)$$

$$\underline{8} \quad \boldsymbol{\beta} = \boldsymbol{\mu} + \boldsymbol{\sigma}\mathbf{L}\mathbf{z} \quad (12.140)$$

9 Here $\mathbf{L}\mathbf{L}^\top$ is the Cholesky decomposition of the correlation matrix \mathbf{R} , so \mathbf{L} is lower diagonal. The
10 distribution LKJchol is the distribution induced on \mathbf{L} by applying the LKJ distribution to \mathbf{R} . We
11 then just have to sample \mathbf{z} from a standard normal, and scale the results. This typically simplifies
12 the inference problem considerably.

13 A method to automatically derive such reparameterizations is discussed in [GMH20].

14

15 12.7 Stochastic gradient MCMC

16 Consider an unnormalized target distribution of the following form:
17

$$\underline{18} \quad \pi(\boldsymbol{\theta}) \propto p(\boldsymbol{\theta}, \mathcal{D}) = p_0(\boldsymbol{\theta}) \prod_{n=1}^N p(\mathbf{x}_n | \boldsymbol{\theta}) \quad (12.141)$$

19 where $\mathcal{D} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$. Alternatively we can define the target distribution in terms of an energy
20 function (negative log joint) as follows:
21

$$\underline{22} \quad p(\boldsymbol{\theta}, \mathcal{D}) \propto \exp(-\mathcal{E}(\boldsymbol{\theta})) \quad (12.142)$$

23 The energy function can be decomposed over data samples:
24

$$\underline{25} \quad \mathcal{E}(\boldsymbol{\theta}) = \sum_{n=1}^N \mathcal{E}_n(\boldsymbol{\theta}) \quad (12.143)$$

$$\underline{26} \quad \mathcal{E}_n(\boldsymbol{\theta}) = -\log p(\mathbf{x}_n | \boldsymbol{\theta}) - \frac{1}{N} \log p_0(\boldsymbol{\theta}) \quad (12.144)$$

27 Evaluating the full energy (e.g., to compute an acceptance probability in the Metropolis Hastings
28 algorithm, or to compute the gradient in HMC) takes $O(N)$ time, which does not scale to large data.
29 In this section, we discuss some solutions to this problem.

30

31 12.7.1 Stochastic Gradient Langevin Dynamics (SGLD)

32 Recall from Equation (12.90) that the (overdamped) **Langevin diffusion** SDE has the following
33 form

$$\underline{34} \quad d\boldsymbol{\theta}_t = -\nabla \mathcal{E}(\boldsymbol{\theta}_t) dt + \sqrt{2} d\mathbf{W}_t \quad (12.145)$$

35

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 Preconditioning

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:

$$\hat{\nabla}_{cv}\mathcal{E}(\boldsymbol{\theta}_t) = \nabla\mathcal{E}(\hat{\boldsymbol{\theta}}) + \frac{N}{B} \sum_{n \in \mathcal{S}_t} (\nabla\mathcal{E}_n(\boldsymbol{\theta}_t) - \nabla\mathcal{E}_n(\hat{\boldsymbol{\theta}})) \quad (12.158)$$

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:

$$\mathbb{E} [\hat{\nabla}_{cv}\mathcal{E}(\boldsymbol{\theta}_t)] = \nabla\mathcal{E}(\hat{\boldsymbol{\theta}}) + \mathbb{E} \left[\frac{N}{B} \sum_{n \in \mathcal{S}_t} (\nabla\mathcal{E}_n(\boldsymbol{\theta}_t) - \nabla\mathcal{E}_n(\hat{\boldsymbol{\theta}})) \right] \quad (12.159)$$

$$= \nabla\mathcal{E}(\hat{\boldsymbol{\theta}}) + \nabla\mathcal{E}(\boldsymbol{\theta}_t) - \nabla\mathcal{E}(\hat{\boldsymbol{\theta}}) = \nabla\mathcal{E}(\boldsymbol{\theta}_t) \quad (12.160)$$

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

$$\hat{\nabla}_{svrg}\mathcal{E}(\boldsymbol{\theta}_t) = \nabla\mathcal{E}(\tilde{\boldsymbol{\theta}}_t) + \frac{N}{B} \sum_{n \in \mathcal{S}_t} (\nabla\mathcal{E}_n(\boldsymbol{\theta}_t) - \nabla\mathcal{E}_n(\tilde{\boldsymbol{\theta}}_t)) \quad (12.161)$$

³² We recompute the snapshot every τ steps (known as the epoch length). See Algorithm 16 for the
³³ pseudo-code.

³⁴

Algorithm 16: SVRG Langevin Descent

```

36 1 Initialize  $\boldsymbol{\theta}_0$  ;
37 2 for  $t = 1 : T$  do
38 3   if  $t \bmod \tau = 0$  then
39 4      $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta}_t$  ;
40 5      $\tilde{\mathbf{g}} = \sum_{n=1}^N \mathcal{E}_n(\tilde{\boldsymbol{\theta}})$  ;
41 6     Sample minibatch  $\mathcal{S}_t \in \{1, \dots, N\}$  ;
42 7      $\mathbf{g}_t = \tilde{\mathbf{g}} + \frac{N}{B} \sum_{n \in \mathcal{S}_t} (\nabla\mathcal{E}_n(\boldsymbol{\theta}_t) - \nabla\mathcal{E}_n(\tilde{\boldsymbol{\theta}}))$  ;
43 8      $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \mathbf{g}_t + \sqrt{2\eta_t} \mathcal{N}(\mathbf{0}, \mathbf{I})$  ;
44
45
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]:

$$\begin{aligned} d\boldsymbol{\theta}_t &= \mathbf{v}_t dt \\ d\mathbf{v}_t &= -\mathbf{g}(\boldsymbol{\theta}_t) dt - \gamma \mathbf{v}_t dt + \sqrt{2\gamma} d\mathbf{W}_t \end{aligned} \quad (12.168)$$

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.

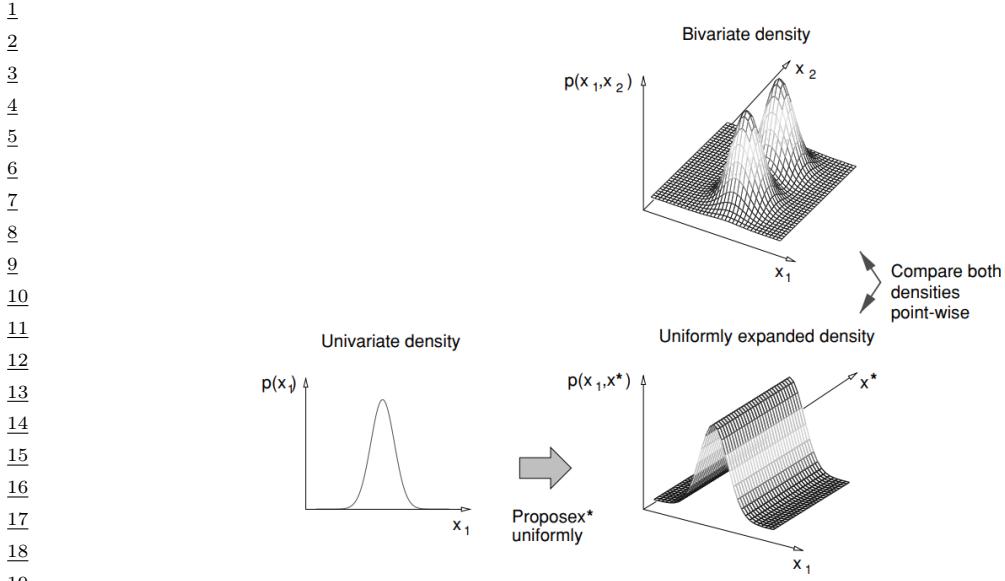


Figure 12.24: To compare a 1d model against a 2d model, we first have to map the 1d model to 2d space so the two have a common measure. Note that we assume the ridge has finite support, so it is integrable. From Figure 17 of [And+03]. Used with kind permission of Nando de Freitas.

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].

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

$$\underline{7} \quad (\mathbf{x}_m, \mathbf{u}_{m,n}) = f_{n \rightarrow m}(\mathbf{x}_n, \mathbf{u}_{n,m}) = (f_{n \rightarrow m}^x(\mathbf{x}_n, \mathbf{u}_{n,m}), f_{n \rightarrow m}^u(\mathbf{x}_n, \mathbf{u}_{n,m})) \quad (12.179)$$

9 Invertibility means that

$$\underline{11} \quad f_{m \rightarrow n}(f_{n \rightarrow m}(\mathbf{x}_n, \mathbf{u}_{n,m})) = (\mathbf{x}_n, \mathbf{u}_{n,m}) \quad (12.180)$$

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

$$\underline{21} \quad A_{n \rightarrow m} = \min \left\{ 1, \frac{p(m, \mathbf{x}_m^*)}{p(n, \mathbf{x}_n)} \times \frac{q(n|m)}{q(m|n)} \times \frac{q_{m \rightarrow n}(\mathbf{u}_{m,n}|m, \mathbf{x}_m^*)}{q_{n \rightarrow m}(\mathbf{u}_{n,m}|n, \mathbf{x}_n)} \times |\det \mathbf{J}_{f_{m \rightarrow n}}| \right\} \quad (12.181)$$

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

$$\underline{27} \quad J_{f_{m \rightarrow n}} = \frac{\partial f_{n \rightarrow m}(\mathbf{x}_m, \mathbf{u}_{m,n})}{\partial (\mathbf{x}_m, \mathbf{u}_{m,n})} \quad (12.182)$$

30 and $|\det \mathbf{J}|$ is the absolute value of the determinant of the Jacobian.

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.

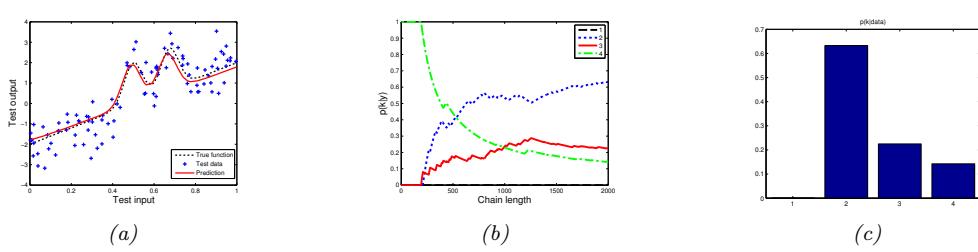


Figure 12.25: Fitting an RBF network to some 1d data using RJMCMC. (a) Prediction on test set. (b) Plot of $p(k|D)$ vs iteration. (c) Final posterior $p(k|D)$. Adapted from Figure 4 of [AFD01].

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

$$\frac{3}{4} A_{merge} = \min \left\{ 1, \frac{p(k-1, \mu_{k-1})}{p(k, \mu_k)} \times \frac{1/(k-1)}{1/k} \times |\det \mathbf{J}_{merge}| \right\} \quad (12.188)$$

5

6 where $|\det \mathbf{J}_{merge}| = 1/(2\beta)$.
7

8 12.8.3 Discussion

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 In Chapter 33, we discuss **Bayesian nonparametric models**, which allow for an infinite number
18 of different models. Surprisingly, such models are often easier to deal with computationally (as well
19 as more realistic, statistically) than working with a finite set of different models.
20

21 12.9 Annealing methods

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

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

$$\pi_t(\mathbf{z}_{1:t}) = \frac{1}{Z_t} \tilde{\gamma}_t(\mathbf{z}_{1:t}) \quad (13.1)$$

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:

$$\pi_T(\mathbf{z}_{1:T}) \propto p(\mathbf{z}_{1:T}, \mathbf{y}_{1:T}) = p(\mathbf{z}_1)p(\mathbf{y}_1|\mathbf{z}_1) \prod_{t=1}^T p(\mathbf{z}_t|\mathbf{z}_{t-1})p(\mathbf{y}_t|\mathbf{z}_t) \quad (13.2)$$

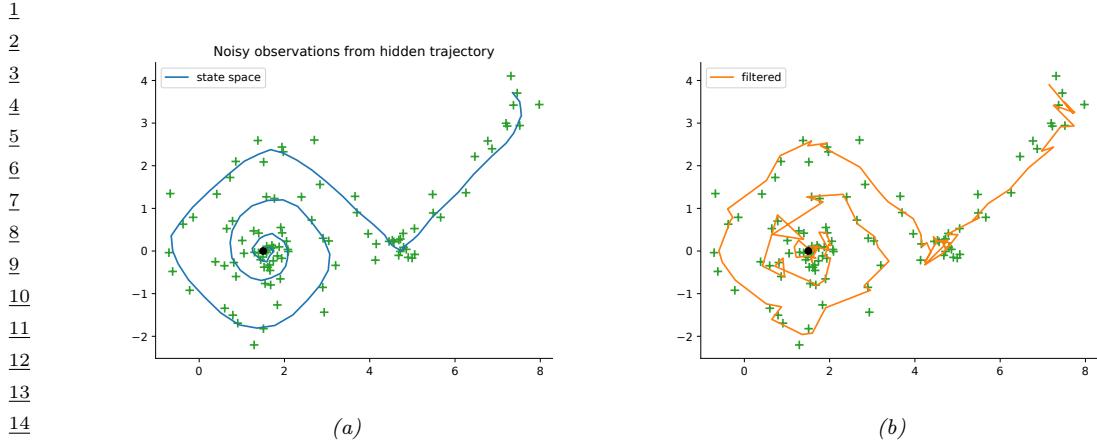


Figure 13.1: Illustration of particle filtering (using the dynamical prior as the proposal) applied to a 2d nonlinear dynamical system. (a) True underlying state and observed data. (b) PF estimate of the posterior mean. Generated by `bootstrap_filter.py`.

19

20

21 A common choice is to define the unnormalized target distribution at step t to be

$$23 \quad \tilde{\gamma}_t(\mathbf{z}_{1:t}) = p(\mathbf{z}_{1:t}, \mathbf{y}_{1:t}) = p(\mathbf{z}_1)p(\mathbf{y}_1|\mathbf{z}_1) \prod_{s=1}^t p(\mathbf{z}_s|\mathbf{z}_{s-1})p(\mathbf{y}_s|\mathbf{z}_s) \quad (13.3)$$

26 Note that this is a distribution over an (ever growing) sequence of variables. However, we often only
27 care about the most recent marginal of this distribution, in which case we just need to compute
28 $\tilde{\gamma}_t(\mathbf{z}_t) = p(\mathbf{z}_t, \mathbf{y}_{1:t})$, which avoids having to store the full history.

29 For example, consider the following 2d nonlinear tracking problem:

$$31 \quad p(\mathbf{z}_t|\mathbf{z}_{t-1}) = \mathcal{N}(\mathbf{z}_t|f(\mathbf{z}_{t-1}), q\mathbf{I}) \\ 32 \quad p(\mathbf{y}_t|\mathbf{z}_t) = \mathcal{N}(\mathbf{y}_t|\mathbf{z}_t, r\mathbf{I}) \\ 33 \quad f(\mathbf{z}) = (z_1 + \Delta \sin(z_2), z_2 + \Delta \cos(z_1)) \quad (13.4)$$

35 where Δ is the step size of the underlying continuous system, q is the variance of the system noise, and
36 r is the variance of the observation noise. The true underlying state trajectory, and the corresponding
37 noisy measurements, are shown in Figure 13.1a. The posterior mean estimate of the state, computed
38 using 2000 samples in a simple form of SMC called the bootstrap filter (Section 13.2.3.1), is shown in
39 Figure 13.1b.
40

41 Particle filtering can also be applied to **non-Markovian models**, where \mathbf{z}_t may depend on all
42 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
43 states, $\mathbf{z}_{1:t-1}$. In this case, the target distribution at step t is

$$44 \quad \tilde{\gamma}_t(\mathbf{z}_{1:t}) = p(\mathbf{z}_1)p(\mathbf{y}_1|\mathbf{z}_1) \prod_{s=1}^t p(\mathbf{z}_s|\mathbf{z}_{1:s-1})p(\mathbf{y}_s|\mathbf{z}_{1:s}) \quad (13.5)$$

For example, consider a 1d Gaussian sequence model where the dynamics are first-order Markov, but the observations depend on the entire past sequence:

$$\begin{aligned} p(z_t | \mathbf{z}_{1:t-1}) &= \mathcal{N}(z_t | \phi z_{t-1}, q^2) \\ p(y_t | \mathbf{z}_{1:t}) &= \mathcal{N}(y_t | \sum_{s=1}^t \beta^{t-s} z_s, r^2) \end{aligned} \quad (13.6)$$

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

$$\frac{3}{4} \quad W_t^i = \frac{\tilde{w}_t^i}{\sum_j \tilde{w}_t^j} \quad (13.9)$$

6 and the **unnormalized weights** are
7

$$\frac{8}{9} \quad \tilde{w}_t^i = \frac{\pi_t(\mathbf{z}_{1:t}^i)}{q_t(\mathbf{z}_{1:t}^i)} \quad (13.10)$$

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

$$\frac{45}{46} \quad \alpha_t(\mathbf{z}_{1:t}) = \frac{\tilde{\gamma}_t(\mathbf{z}_{1:t})}{\tilde{\gamma}_{t-1}(\mathbf{z}_{1:t-1}) q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1})} \quad (13.16)$$

See Algorithm 18 for pseudocode for the resulting SIS algorithm. (In practice we compute the weights in log-space, and convert back using the log-sum-exp trick.)

Note that, in the special case of state space models, the weight computation can be further simplified. In particular, suppose we have

$$\tilde{\gamma}_t(\mathbf{z}_{1:t}) = p(\mathbf{z}_{1:t}, \mathbf{y}_{1:t}) = p(\mathbf{y}_t|\mathbf{z}_{1:t})p(\mathbf{z}_t|\mathbf{z}_{1:t-1})p(\mathbf{z}_{1:t-1}, \mathbf{y}_{1:t-1}) = p(\mathbf{y}_t|\mathbf{z}_{1:t})p(\mathbf{z}_t|\mathbf{z}_{1:t-1})\tilde{\gamma}_{t-1}(\mathbf{z}_{1:t-1}) \quad (13.17)$$

Then the incremental weight is given by

$$\alpha_t(\mathbf{z}_{1:t}) = \frac{p(\mathbf{y}_t|\mathbf{z}_{1:t})p(\mathbf{z}_t|\mathbf{z}_{1:t-1})}{q_t(\mathbf{z}_t|\mathbf{z}_{1:t-1})} \quad (13.18)$$

Algorithm 18: Sequential importance sampling

```

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   for  $i = 1 : N_s$  do
4     Sample  $\mathbf{z}_t^i \sim q_t(\mathbf{z}_t|\mathbf{z}_{1:t-1}^i)$ 
5     Append  $\mathbf{z}_{1:t}^i = (\mathbf{z}_{1:t-1}^i, \mathbf{z}_t^i)$ 
6     Compute incremental weight  $\alpha_t^i = \frac{\tilde{\gamma}_t(\mathbf{z}_{1:t}^i)}{\tilde{\gamma}_{t-1}(\mathbf{z}_{1:t-1}^i)q_t(\mathbf{z}_t^i|\mathbf{z}_{1:t-1}^i)}$ 
7     Compute unnormalized weight  $\tilde{w}_t^i = \tilde{w}_{t-1}^i \alpha_t^i$ 
8     Compute normalized weights  $W_t^i = \frac{\tilde{w}_t^i}{\sum_j \tilde{w}_t^j}$  for  $i = 1 : N_s$ 
9   Define  $\hat{\pi}_t(\mathbf{z}_{1:t}) = \sum_{i=1}^{N_s} W_t^i \delta(\mathbf{z}_{1:t} - \mathbf{z}_{1:t}^i)$ 

```

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 amongst 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.

13.2.3 Sequential importance sampling with resampling

In this section, we describe **sequential importance sampling with resampling (SISR)**. The basic idea is this: instead of “growing” all of the old particle sequences by one step, we first select the N_s “fittest” particles, by sampling from the old posterior, and then letting these survivors grow by one step.

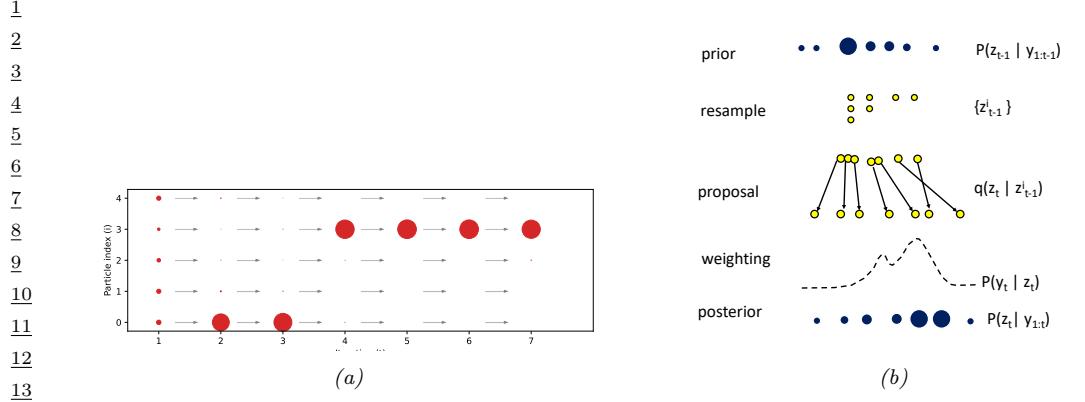


Figure 13.2: (a) Illustration of weight degeneracy for SIS applied to the model in Equation (13.6). with parameters $(\phi, q, \beta, r) = (0.9, 10.0, 0.5, 1.0)$. We use $T = 6$ steps ad $N_s = 5$ samples. We see that as t increases, almost all the probability mass concentrates on particle 3. Generated by [sis_vs_smcmc_demo.py](#). Adapted from Figure 2 of [NLS19]. (b) Illustration of the bootstrap particle filtering algorithm.

18

Algorithm 19: Sequential importance sampling with resampling

```

21 1 Initialization:  $z_1^i \sim q_1(z_1)$ ,  $\tilde{w}_1^i = \frac{\tilde{\gamma}_1(z_1^i)}{q_1(z_1^i)}$ ,  $W_1^i = \frac{\tilde{w}_1^i}{\sum_j \tilde{w}_1^j}$ ,  $\hat{\pi}_1(z_1) = \sum_{i=1}^{N_s} W_1^i \delta(z_1 - z_1^i)$ 
22 2 for  $t = 2 : T$  do
23 3   for  $i = 1 : N_s$  do
24 4     Resample  $z_{t-1}^{1:N_s} = \text{resample}(\hat{\pi}_{t-1})$ 
25 5     Move  $z_t^i \sim q_t(z_t | z_{1:t-1}^i)$ 
26 6     Weight  $\tilde{w}_t^i = \alpha_t^i = \frac{\tilde{\gamma}_t(z_{1:t}^i)}{\tilde{\gamma}_{t-1}(z_{1:t-1}^i) q_t(z_t^i | z_{1:t-1}^i)}$ 
27 7     Compute  $W_t^i = \frac{\tilde{w}_t^i}{\sum_j \tilde{w}_t^j}$  for  $i = 1 : N_s$ 
28 8   Define  $\hat{\pi}_t(z_{1:t}) = \sum_{i=1}^{N_s} W_t^i \delta(z_{1:t} - z_{1:t}^i)$ 
31
32
33
```

34 In more detail, at step t , we sample from

$$q_t^{\text{SISR}}(z_{1:t}) = \hat{\pi}_{t-1}(z_{1:t-1}) q_t(z_t | z_{1:t-1}) \quad (13.19)$$

35 where $\hat{\pi}_{t-1}(z_{1:t-1})$ is the previous weighted posterior approximation. By contrast, in SIS, we sample
36 from

$$q_t^{\text{SIS}}(z_{1:t}) = q_{t-1}^{\text{SIS}}(z_{1:t-1}) q_t(z_t | z_{1:t-1}) \quad (13.20)$$

42 We can sample from Equation (13.19) in two steps. First we **resample** N_s samples from $\hat{\pi}_{t-1}(z_{1:t-1})$
43 to get a *uniformly weighted* set of new samples $z_{1:t-1}^i$. (See Section 13.2.4 for details on how to do
44 this.) Then we extend each sample using $z_t^i \sim q_t(z_t | z_{1:t-1}^i)$, and concatenate z_t^i to $z_{1:t-1}^i$,
45 After making a proposal, we compute the unnormalized weights. We use the usual expression
46 for target over proposal, except we “pretend” that the proposal is given by $\tilde{\gamma}_{t-1}(z_{1:t-1}^i) q_t(z_t^i | z_{1:t-1}^i)$

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

$$\hat{Z}_T = \prod_{t=1}^T \frac{1}{N_s} \sum_{i=1}^{N_s} \tilde{w}_t^i \quad (13.26)$$

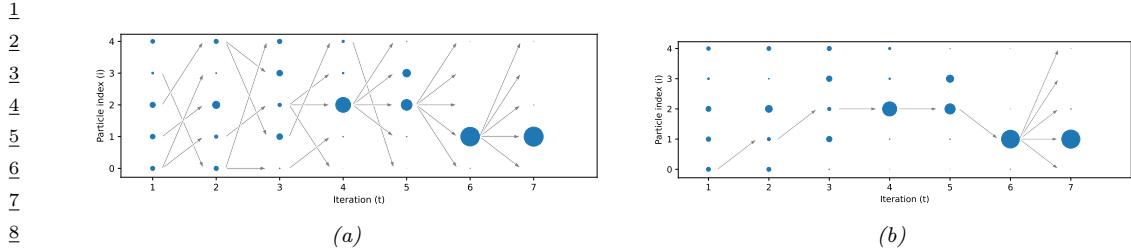


Figure 13.3: (a) Illustration of diversity of samples in SMC applied to the model in Equation (13.6). (b) Illustration of the path degeneracy problem. Generated by `/sis_vs_smc_demo.py`. Adapted from Figure 3 of [NLS19].

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)
```

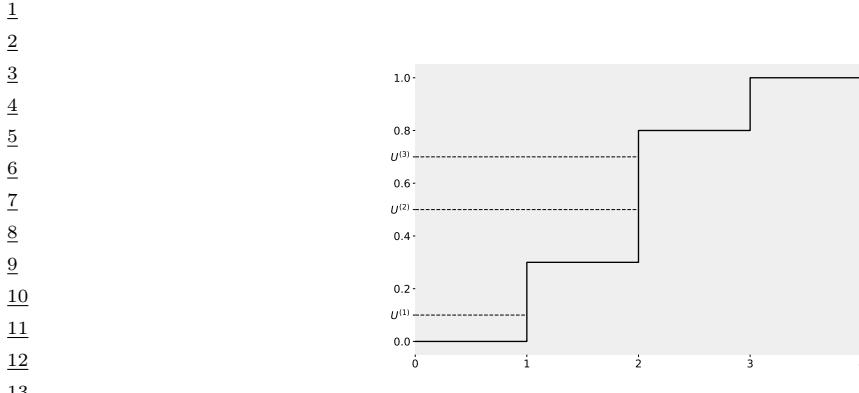


Figure 13.4: Illustration of how to sample from the empirical CDF $P(x) = \sum_{n=1}^N W^n \mathbb{I}(x \geq n)$ shown in black. The height of step n is W_n . If U^m picks step n , then we set the ancestor of m to be n , i.e., $A^m = n$. In this example, $A^{1:3} = (1, 2, 2)$. Adapted from Figure 9.3 of [CP20b].

```

14
15 bins = np.cumsum(w)
16 ancestors = np.digitize(u, bins)
17 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

```

41 def stratified_resampling(w, x):
42     N = w.shape[0]
43     u = (np.arange(N) + np.random.rand(N))/N
44     bins = np.cumsum(w)
45     ancestors = np.digitize(u, bins)
46     return x[ancestors]
```

1 **13.2.4.3 Systematic resampling**

2 We can further reduce the variance by forcing all the samples from $\hat{\pi}_{t-1}$ to be deterministically
3 generated from a shared random source, $u^i \sim \text{Unif}(0, 1)$, by computing
4

5

$$\frac{u^i}{N_s} = \frac{i - 1}{N_s} + \frac{u}{N_s} \quad (13.28)$$

6

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.)

9 *Listing 13.3: Systematic resampling*

```
10    def systematic_resampling(w, x):
11        N = w.shape[0]
12        u = (np.arange(N) + np.random.rand()) / N
13        bins = np.cumsum(w)
14        ancestors = np.digitize(u, bins)
15        return x[ancestors]
```

16

17

18 **13.2.4.4 Comparison**

19 It can be proved that all of the above methods are unbiased. It can also be proved that stratified
20 resampling is lower variance than multinomial resampling. Empirically it seems that systematic
21 resampling is lower variance than other methods [HSG06], although it can fail to converge depending
22 on the order of the input samples [GCW19]. A more complex resampling scheme, that is guaranteed
23 to converge and which is also low variance, is described in [GCW19].

24

25 **13.2.5 Adaptive resampling**

26 The resampling step can result in loss of diversity, since each ancestor may generate multiple children,
27 and some may generate no children, since the ancestor indices A_t^n are sampled independently; this
28 is called path degeneracy. On the other hand, if we never resample, we end up with SIS, which
29 suffers from weight degeneracy (particles with negligible weight). A compromise is to use **adaptive**
30 **resampling**, in which we resample whenever the **effective sample size** or **ESS** drops below some
31 minimum, such as $N/2$.

32 A common way to define the ESS is as follows:

33

$$\text{ESS}(W^{1:N}) = \frac{1}{\sum_{n=1}^N (W^n)^2} = \frac{\left(\sum_{n=1}^N \tilde{w}^n\right)^2}{\sum_{n=1}^N (\tilde{w}^n)^2} \quad (13.29)$$

34

35 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
36 ESS is between 1 and N .

37 The pseudocode for SISR with adaptive resampling is given in Algorithm 20. We see that the
38 expression for the weights of the particles are different when no resampling occurs. In particular,
39 instead of using the equally weighted samples after resampling, we use the weighted samples from the
40 previous step without resampling. We then need to add the term $\frac{W_{t-1}^i}{1/N_s}$ as an importance sampling
41 correction, where the factor $1/N_s$ corresponds to the proposal distribution of not resampling [NLS19,
42

¹ p30]. Although these extra factors will cancel out when we compute the normalized weights, W_t^i , we
² need to keep track of them for the estimate of Z_T in Equation (13.26) to be valid. (An alternative
³ approach is to compute Z_T in a different way, by keeping track of which steps involved resampling,
⁴ as in [CP20b, p134].)

⁵

⁶

⁷

Algorithm 20: SISR with adaptive resampling

```

9 1 Initialization:  $\mathbf{z}_1^i \sim q_1(\mathbf{z}_1)$ ,  $\tilde{w}_1^i = \frac{\tilde{\gamma}_1(\mathbf{z}_1^i)}{q_1(\mathbf{z}_1^i)}$ ,  $W_1^i = \frac{\tilde{w}_1^i}{\sum_j \tilde{w}_1^j}$ ,  $\hat{\pi}_1(\mathbf{z}_1) = \sum_{i=1}^{N_s} W_1^i \delta(\mathbf{z}_1 - \mathbf{z}_1^i)$ 
10 2 for  $t = 2 : T$  do
11 3   if  $ESS(W_{t-1}^{1:N}) < ESS_{\min}$  then
12 4     Resample  $\mathbf{z}_{t-1}^{1:N_s} = \text{resample}(\hat{\pi}_{t-1})$ 
13 5     for  $i = 1 : N_s$  do
14 6       Move  $\mathbf{z}_t^i \sim q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1}^i)$ 
15 7       Unnormalized weights  $\tilde{w}_t^i = \frac{\tilde{\gamma}_t(\mathbf{z}_{1:t}^i)}{\tilde{\gamma}_{t-1}(\mathbf{z}_{1:t-1}^i) q_t(\mathbf{z}_t^i | \mathbf{z}_{1:t-1}^i)}$ 
16 8     else
17 9       for  $i = 1 : N_s$  do
20 10         Move  $\mathbf{z}_t^i \sim q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1}^i)$ 
21 11         Unnormalized weights  $\tilde{w}_t^i = \frac{W_{t-1}^i}{1/N_s} \frac{\tilde{\gamma}_t(\mathbf{z}_{1:t}^i)}{\tilde{\gamma}_{t-1}(\mathbf{z}_{1:t-1}^i) q_t(\mathbf{z}_t^i | \mathbf{z}_{1:t-1}^i)}$ 
23 12       Normalized weights  $W_t^i = \frac{\tilde{w}_t^i}{\sum_j \tilde{w}_t^j}$  for  $i = 1 : N_s$ 
25 13       Define  $\hat{\pi}_t(\mathbf{z}_{1:t}) = \sum_{i=1}^{N_s} W_t^i \delta(\mathbf{z}_{1:t} - \mathbf{z}_{1:t}^i)$ 

```

²⁶

²⁷

²⁸

²⁹

13.3 Some applications of particle filtering

³² In this section, we give some examples of particle filtering applied to some state estimation problems
³³ in different kinds of state-space models. We focus on using the simplest kind of SMC algorithm,
³⁴ namely the bootstrap filter (Section 13.2.3.1).

³⁵

³⁶

13.3.1 1d pendulum model with outliers

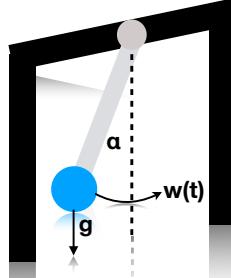
³⁹ Consider a simple pendulum of unit mass and length swinging from a fixed attachment, as in
⁴⁰ Figure 13.5. Such an object is in principle entirely deterministic in its behavior. However, in the real
⁴¹ world, there are often unknown forces at work (e.g., air turbulence, friction). We will model these by
⁴² a continuous time random Gaussian noise process $w(t)$. This gives rise to the following differential
⁴³ equation:

⁴⁴

$$\frac{d^2\alpha}{dt^2} = -g \sin(\alpha) + w(t) \quad (13.30)$$

⁴⁵

1
2
3
4
5
6
7
8
9
10



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

15 We can write this as a nonlinear SSM by defining the state to be $z_1(t) = \alpha(t)$ and $z_2(t) = d\alpha(t)/dt$.
16 Thus

17

$$\frac{d\mathbf{z}}{dt} = \begin{pmatrix} z_2 \\ -g \sin(z_1) \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} w(t) \quad (13.31)$$

20

21 If we discretize this step size Δ , we get the following formulation [Sar13, p74]:

22

$$\underbrace{\begin{pmatrix} z_{1,t} \\ z_{2,t} \end{pmatrix}}_{\mathbf{z}_t} = \underbrace{\begin{pmatrix} z_{1,t-1} + z_{2,t-1}\Delta \\ z_{2,t-1} - g \sin(z_{1,t-1})\Delta \end{pmatrix}}_{\mathbf{f}(\mathbf{z}_{t-1})} + \mathbf{q}_{t-1} \quad (13.32)$$

26

27 where $\mathbf{q}_{t-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ with

28

$$\mathbf{Q} = q^c \begin{pmatrix} \frac{\Delta^3}{3^2} & \frac{\Delta^2}{2} \\ \frac{\Delta^2}{2} & \Delta \end{pmatrix} \quad (13.33)$$

31

32 where q^c is the spectral density (continuous time variance) of the continuous-time noise process.

33 If we observe the angular position, we get the linear observation model

34

$$y_t = \alpha_t + r_t = h(\mathbf{z}_t) + r_t \quad (13.34)$$

36 where $h(\mathbf{z}_t) = z_{1,t}$ and r_t is the observation noise. If we only observe the horizontal position, we get
37 the nonlinear observation model

38

$$y_t = \sin(\alpha_t) + r_t = h(\mathbf{z}_t) + r_t \quad (13.35)$$

40

41 where $h(\mathbf{z}_t) = \sin(z_{1,t})$.

42 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)
43 and bootstrap filter (Section 13.2.3.1) all give very similar results (not shown). However, suppose
44 that some fraction $p = 0.4$ of the observations are outliers, coming from a $\text{Unif}(-2, 2)$ distribution.
45 (These could represent a faulty sensor, for example.) In this case, the bootstrap filter is more robust,
46 as shown in Figure 13.6.

47

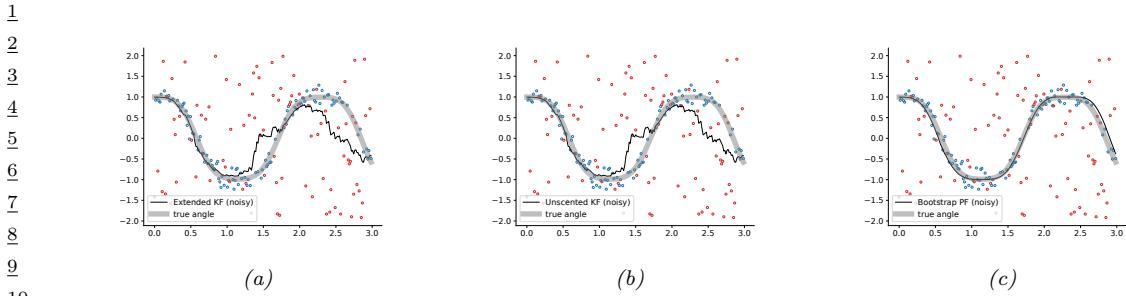


Figure 13.6: Filtering algorithms applied to the noisy pendulum model with 40% outliers (shown in red). (a) EKF. (b) UKF. (c) Bootstrap filter. Generated by `pendulum_1d_demo.py`.

13.3.2 Visual object tracking

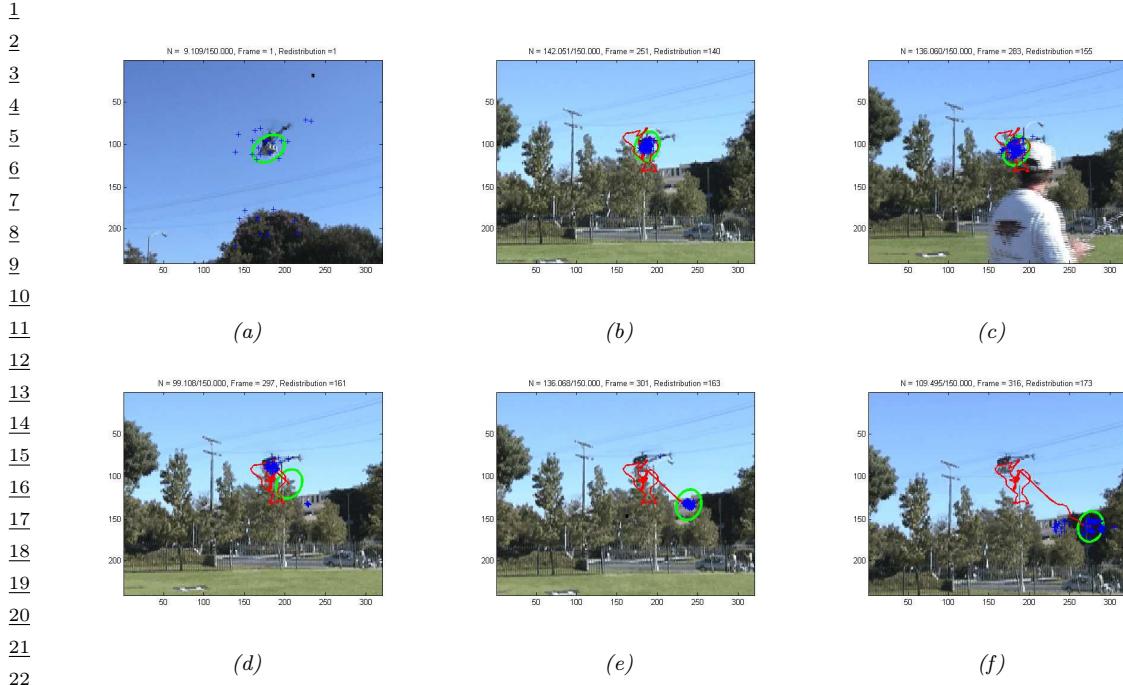
In Section 13.1.2, we tracked an object given noisy measurements of its location, as estimated by some kind of distance sensor. A harder problem is to track an object just given a sequence of frames from a video camera. This is called **visual tracking**. In this section we consider an example where 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**



23 *Figure 13.7: Example of particle filtering applied to visual object tracking, based on color histograms. Blue*
24 *dots are posterior samples, green ellipse is Gaussian approximation to posterior. (a-c) Successful tracking.*
25 *(d): Tracker gets distracted by an outlier gray patch in the background, and moves the posterior mean away from*
26 *the object. (e-f): Losing track. See text for details. Used with kind permission of Sébastien Paris.*

30 **filter** [JB16b]), since we are assuming the state space is discrete. However, since the number of states,
31 K , is often very large, the $O(K^2)$ time complexity per update is prohibitive. We can use a particle
32 filter as a sparse approximation to the belief state. This is known as **Monte Carlo localization**
33 [TBF06].

34 Figure 13.8 gives an example of the method in action. The robot uses a sonar range finder, so
35 it can only sense distance to obstacles. It starts out with a uniform prior, reflecting the fact that
36 the owner of the robot may have turned it on in an arbitrary location. (Figuring out where you are,
37 starting from a uniform prior, is called **global localization**.) After the first scan, which indicates
38 two walls on either side, the belief state is shown in (b). The posterior is still fairly broad, since the
39 robot could be in any location where the walls are fairly close by, such as a corridor or any of the
40 narrow rooms. After moving to location 2, the robot is pretty sure it must be in a corridor and not a
41 room, as shown in (c). After moving to location 3, the sensor is able to detect the end of the corridor.
42 However, due to symmetry, it is not sure if it is in location I (the true location) or location II. (This
43 is an example of **perceptual aliasing**, which refers to the fact that different things may look the
44 same.) After moving to locations 4 and 5, it is finally able to figure out precisely where it is (not
45 shown). The whole process is analogous to someone getting lost in an office building, and wandering
46 the corridors until they see a sign they recognize.

47

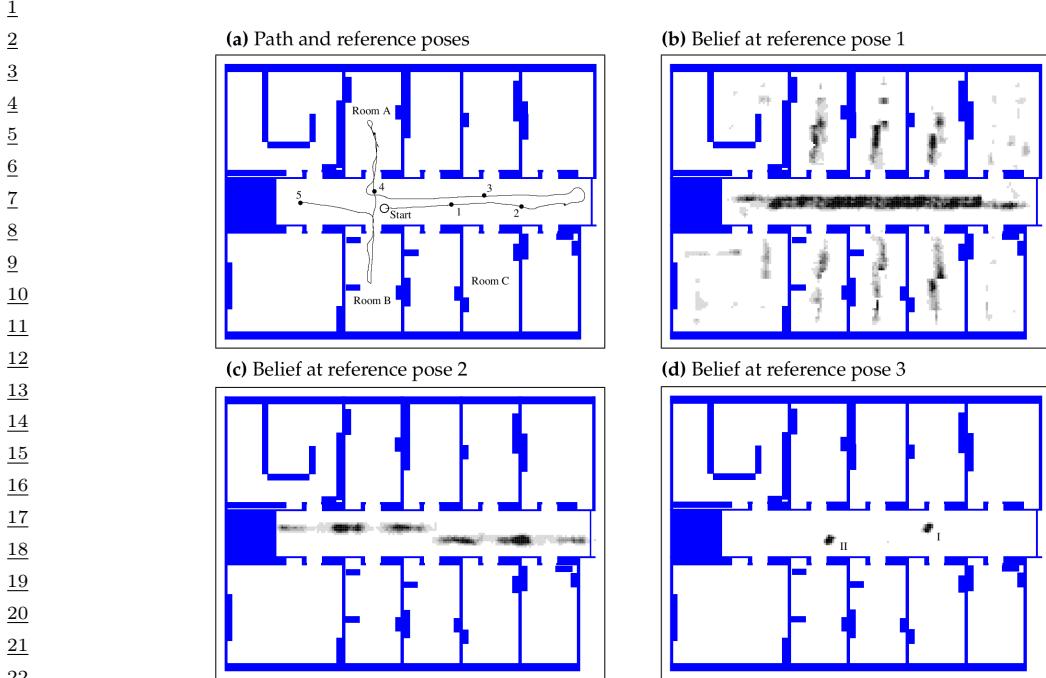


Figure 13.8: Illustration of Monte Carlo localization for a mobile robot in an office environment using a sonar sensor. From Figure 8.7 of [TBF06]. Used with kind permission of Sebastian Thrun.

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 **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

6 $J = D_{\text{KL}}(\pi_{t-1}(\mathbf{z}_{1:t-1})q_t^*(\mathbf{z}_t | \mathbf{z}_{1:t-1}) \| \pi_t(\mathbf{z}_{1:t}))$ (13.36)

7 This is given by

9 $q_t^*(\mathbf{z}_t | \mathbf{z}_{1:t-1}) = \pi_t(\mathbf{z}_t | \mathbf{z}_{1:t-1}) = \frac{\tilde{\gamma}_t(\mathbf{z}_{1:t})}{\tilde{\gamma}_t(\mathbf{z}_{1:t-1})}$ (13.37)

12 where $\tilde{\gamma}_t(\mathbf{z}_{1:t-1}) = \int \tilde{\gamma}_t(\mathbf{z}_{1:t}) d\mathbf{z}_t$.

13 To see this, note that we have the following (where const refers to terms that are constant wrt the
14 proposal):

16 $D_{\text{KL}}(\pi_{t-1}(\mathbf{z}_{1:t-1})q_t^*(\mathbf{z}_t | \mathbf{z}_{1:t-1}) \| \pi_t(\mathbf{z}_{1:t}))$ (13.38)

17 $= \mathbb{E}_{\pi_{t-1}q_t} [\log \{\pi_{t-1}(\mathbf{z}_{1:t-1})q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1})\} - \log \pi_t(\mathbf{z}_{1:t})]$ (13.39)

18 $= \mathbb{E}_{\pi_{t-1}q_t} [\log q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1}) - \log \pi_t(\mathbf{z}_t | \mathbf{z}_{1:t-1})] + \text{const}$ (13.40)

19 $= \mathbb{E}_{\pi_{t-1}q_t} [D_{\text{KL}}(q_t(\mathbf{z}_t | \mathbf{z}_{1:t-1}) \| \pi_t(\mathbf{z}_t | \mathbf{z}_{1:t-1}))] + \text{const}$ (13.41)

21 where the inner KL is minimized by choosing $q_t^*(\mathbf{z}_t | \mathbf{z}_{1:t-1}) = \pi_t(\mathbf{z}_t | \mathbf{z}_{1:t-1})$.

22 Note that the subscript t specifies the t 'th distribution, so in the context of SSMs, we have
23 $\pi_t(\mathbf{z}_t | \mathbf{z}_{1:t-1}) = p(\mathbf{z}_t | \mathbf{z}_{1:t-1}, \mathbf{y}_{1:t})$. Thus we see that when proposing \mathbf{z}_t , we should condition on all
24 the data, including the most recent observation, \mathbf{y}_t ; this is called a **guided particle filter**, and will
25 will be better than the bootstrap filter, which proposes from the prior.

26 In general, it is intractable to compute the locally optimal proposal, but it can be done in some
27 cases. For example consider the non-Markov, but Gaussian, SSM from Equation (13.6), with target

29 $\tilde{\gamma}_t(\mathbf{z}_{1:t}) = \tilde{\gamma}_{t-1}(\mathbf{z}_{1:t-1})p(z_t | z_{t-1})p(y_t | \mathbf{z}_{1:t})$ (13.42)

31 Since the joint distribution $p(\mathbf{z}_{1:t} | \mathbf{y}_{1:t})$ is multivariate Gaussian, we can compute the optimal proposal
32 as follows [NLS19, p35]:

34 $q_t^*(z_t | \mathbf{z}_{1:t-1}) \propto \frac{\tilde{\gamma}_t(\mathbf{z}_{1:t})}{\tilde{\gamma}_t(\mathbf{z}_{1:t-1})} \propto p(z_t | z_{t-1})p(y_t | \mathbf{z}_{1:t})$ (13.43)

36 $\propto \mathcal{N}\left(z_t \mid \frac{r\phi z_{t-1} + qy_t - q \sum_{s=1}^{t-1} \beta^{t-s} z_s}{q+r}, \frac{qr}{q+r}\right)$ (13.44)

39 **13.4.2 Proposals based on the Laplace approximation**

41 One way to approximate the optimal proposal is to use the Laplace approximation (Section 7.4.3), as
42 suggested in [DGA00]. In particular, consider an SSM with linear-Gaussian latent dynamics and a
43 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.,
44 using Newton-Raphson), and then approximate the likelihood using

46 $p(\mathbf{y}_t | \mathbf{z}_t) \approx \mathcal{N}(\mathbf{z}_t | \mathbf{z}_t^*, -\mathbf{H}_t^*)$ (13.45)

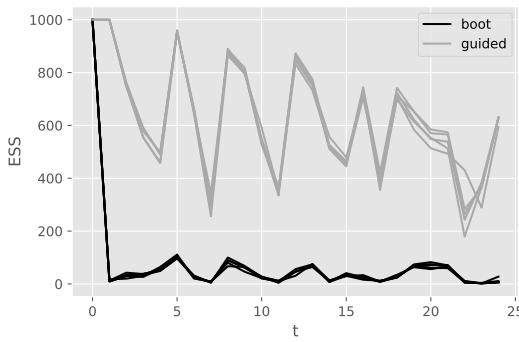


Figure 13.9: Effective sample size at each step for the bootstrap particle filter and a guided particle filter for a Gaussian SSM with Poisson likelihood. Adapted from Figure 10.4 of [CP20b]. Generated by `pf_guided_neural_decoding.ipynb`.

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”.)

¹ To illustrate this, we sample random parameters α and β , and then sample data from the model for
² 25 time steps. We then apply particle filtering to the problem, using either the bootstrap filter (i.e.,
³ proposal is the random walk prior) or the guided filter (i.e., proposal is the Laplace approximation
⁴ mentioned above). In Figure 13.9, we see that the effective sample size of the guided filter is much
⁵ higher than for the bootstrap filter.
⁶

⁷ 13.4.3 Proposals based on the extended and unscented Kalman filter

⁸ An alternative way to create approximate proposal distributions is based on the extended Kalman
⁹ filter (Section 8.5.2) and unscented Kalman filter (Section 13.4.3). This combination is called the
¹⁰ **extended particle filter** [DGA00] and **unscented particle filter** [Mer+00]. To explain these
¹¹ methods, we follow the presentation of [NLS19, p36]. We assume the (non-linear and/or non-Gaussian)
¹² dynamical system can be written as follows:
¹³

$$\underline{z}_t = f(\underline{z}_{t-1}, \epsilon_t) \quad (13.50)$$

$$\underline{y}_t = h(\underline{z}_t, \eta_t) \quad (13.51)$$

¹⁴ where ϵ_t is the system noise and η_t is the observation noise. The EKF and UKF approximations
¹⁵ assume that the two-slice joint distribution is Gaussian:
¹⁶

$$\underline{p}(\underline{z}_t, \underline{y}_t | \underline{z}_{1:t-1}) \approx \mathcal{N}\left(\begin{pmatrix} \underline{z}_t \\ \underline{y}_t \end{pmatrix} | \hat{\mu}, \hat{\Sigma}\right) \quad (13.52)$$

²¹ where

$$\hat{\mu} = \begin{pmatrix} \hat{\mu}_z \\ \hat{\mu}_y \end{pmatrix}, \hat{\Sigma} = \begin{pmatrix} \hat{\Sigma}_{zz} & \hat{\Sigma}_{zy} \\ \hat{\Sigma}_{yz} & \hat{\Sigma}_{yy} \end{pmatrix} \quad (13.53)$$

²² The EKF and UKF compute μ and Σ differently. In the EKF, we linearize f and h , and assume
²³ the noise terms are Gaussian. We then compute $p(\underline{z}_t, \underline{y}_t | \underline{z}_{1:t-1})$ exactly for this linearized model. In
²⁴ the UKF, we propagate sigma points through f and h , and approximate the resulting means and
²⁵ covariances (see Section 8.6.1). This avoids the need to compute the Jacobian term, and does not
²⁶ rely on a Gaussian assumption. For the details, see [NLS19, p56].
²⁷

²⁸ Once we have computed μ and Σ , we can use standard rules for Gaussian conditioning to compute
²⁹ the approximate proposal as follows:
³⁰

$$\underline{p}(\underline{z}_t | \underline{z}_{1:t-1}, \underline{y}_t) \approx \mathcal{N}(\underline{z}_t | \mu_t, \Sigma_t) \quad (13.54)$$

$$\mu_t = \hat{\mu}_z + \hat{\Sigma}_{zy} \hat{\Sigma}_{yy}^{-1} (\underline{y}_t - \hat{\mu}_y) \quad (13.55)$$

$$\Sigma_t = \hat{\Sigma}_{zz} - \hat{\Sigma}_{zy} \hat{\Sigma}_{yy}^{-1} \hat{\Sigma}_{yz} \quad (13.56)$$

⁴¹ 13.4.4 Proposals based on SMC

⁴² It is possible to use importance sampling, or SMC, to approximate the proposal $q_t(\underline{z}_t | \underline{z}_{1:t-1}^i)$ for
⁴³ each particle i . This is called **nested SMC** [NLS15; NLS19]. This can be considered an “exact
⁴⁴ approximation”, since the method can approach the locally optimal proposal arbitrarily well, since it
⁴⁵ does not make any limiting parametric assumptions. However, the method can be slow. In particular,
⁴⁶

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

$$\underline{3} \quad \mathcal{L} = \int p(\mathbf{y}) D_{\text{KL}}(p(\mathbf{z}_{1:T}|\mathbf{y}) \| q_T(\mathbf{z}_{1:T}; \boldsymbol{\lambda}_\phi(\mathbf{y}_{1:T}))) d\mathbf{y} \quad (13.61)$$

$$\underline{4} \quad = \int p(\mathbf{y}) \int p(\mathbf{z}|\mathbf{y}) \log \left[\frac{p(\mathbf{z}|\mathbf{y})}{q_T(\mathbf{z}_{1:T}; \boldsymbol{\lambda}_\phi(\mathbf{y}_{1:T}))} \right] dz d\mathbf{y} \quad (13.62)$$

$$\underline{5} \quad = -\mathbb{E}_{p(\mathbf{z}_{1:T}, \mathbf{y}_{1:T})} [\log q_T(\mathbf{z}_{1:T}; \boldsymbol{\lambda}_\phi(\mathbf{y}_{1:T}))] + \text{const} \quad (13.63)$$

6 where $\boldsymbol{\lambda}_\phi(\mathbf{y}) = f_\phi^{\text{inf}}(\mathbf{y})$ are the parameters of the proposal computed by an inference network.
7 We
8 can approximate the gradient of this using
9

$$\underline{10} \quad \nabla \mathcal{L}^k = \nabla_{\boldsymbol{\lambda}} \log q_T(\tilde{\mathbf{z}}_{1:T}; \boldsymbol{\lambda}_\phi(\tilde{\mathbf{y}}_{1:T})) \quad (13.64)$$

11 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
12 estimate, and does not need to use SMC in the inner loop. However, the inference network needs to
13 be trained offline on simulated data, so the method may not be optimal for any particular observed
14 sequence $\mathbf{y}_{1:T}$.

15 In many problems, the model has repeated structure, which lets us learn a single inference network
16 that can be reused multiple times.

17

18 13.4.7 Variational SMC

19 Instead of trying to learn good local proposals, we can instead try to learn proposals such that the
20 resulting joint distribution arising from the entire SMC process, $\hat{\pi}_T$, is close to the target, π_T . Let
21 $\mathbb{E}[\hat{\pi}_T]$ be the marginal distribution of a single sample of the empirical joint from SMC. One can show
22 [Nae+18] the KL divergence of this to the true distribution can be bounded as follows:

$$\underline{23} \quad D_{\text{KL}}(\mathbb{E}[\hat{\pi}_T(\mathbf{z}_{1:T}; \boldsymbol{\lambda})] \| \pi_T(\mathbf{z}_{1:T})) \leq -\mathbb{E} \left[\log \frac{\hat{Z}_T}{Z_T} \right] \quad (13.65)$$

24 where

$$\underline{25} \quad \mathbb{E} \left[\log \hat{Z}_T \right] = \mathbb{E} \left[\sum_{t=1}^T \log \left(\frac{1}{N_s} \sum_{i=1}^{N_s} \tilde{w}_t^i(\mathbf{z}_{1:t}^i; \boldsymbol{\lambda}) \right) \right] \quad (13.66)$$

26 Since the KL divergence is non-negative, we have

$$\underline{27} \quad \mathbb{E} \left[\log \hat{Z}_T \right] \leq \mathbb{E}[Z_T] \quad (13.67)$$

28 Thus Equation (13.66) is an evidence lower bound (Section 10.1.2). Maximizing this results in a
29 technique called **variational SMC** [Nae+18; Le+18; Mad+17].

30 We can approximate \hat{Z}_T using SMC. If we assume the proposal distribution is reparameterizable
31 (Section 6.6.4), and if we ignore the gradient from the resampling step, we can approximate the
32 gradient of the ELBO using

$$\underline{33} \quad \nabla_{\boldsymbol{\lambda}} \mathbb{E} \left[\log \hat{Z}_T \right] \approx \mathbb{E} \left[\sum_{t=1}^T \sum_{i=1}^{N_s} W_t^i \nabla_{\boldsymbol{\lambda}} \log \tilde{w}_t^i(\mathbf{z}_{1:t}^i; \boldsymbol{\lambda}) \right] \quad (13.68)$$

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

$$q(\mathbf{c}_{1:t} | \mathbf{y}_{1:t}) = q(\mathbf{c}_t | \mathbf{c}_{1:t-1}, \mathbf{y}_{1:t}) q(\mathbf{c}_{1:t-1} | \mathbf{y}_{1:t}) \quad (13.77)$$

3

4 Then we get the unnormalized importance weights

5

$$\tilde{w}_t^n \propto \frac{p(\mathbf{y}_t | c_t^n, \mathbf{c}_{1:t-1}^n, \mathbf{y}_{1:t-1}) p(c_t^n | c_{t-1}^n)}{q(c_t^n | \mathbf{c}_{1:t-1}^n, \mathbf{y}_{1:t})} \tilde{w}_{t-1}^n \quad (13.78)$$

6

7 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

8 weight update becomes

9

$$\tilde{w}_t^n \propto \tilde{w}_{t-1}^n p(\mathbf{y}_t | c_t^n = k, \mathbf{c}_{1:t-1}^n, \mathbf{y}_{1:t-1}) = \tilde{w}_{t-1}^n L_{tk}^n \quad (13.79)$$

10

11 where

12

$$L_{tk}^n = p(\mathbf{y}_t | c_t = k, \mathbf{c}_{1:t-1}^n, \mathbf{y}_{1:t-1}) = \int p(\mathbf{y}_t | c_t = k, \mathbf{z}_t) p(\mathbf{z}_t | c_t = k, \mathbf{y}_{1:t-1}, \mathbf{c}_{1:t-1}^n) d\mathbf{z}_t \quad (13.80)$$

13

14 The quantity L_{tk}^n is the predictive density for the new observation \mathbf{y}_t conditioned on $c_t = k$ and

15 the history of previous latents, $\mathbf{c}_{1:t-1}^n$. In the case of SLDS models, this can be computed using

16 the normalization constant of the Kalman filter, Equation (8.98). The resulting algorithm is shown

17 in Algorithm 21. The step marked ‘‘KFupdate’’ refers to the Kalman filter update equations in

18 Section 8.4.1, and is applied to each particle separately.

19

20 **Algorithm 21:** One step of RBPF for SLDS using prior as proposal

21 1 **for** $n = 1 : N$ **do**

22 2 $k \sim p(c_t | c_{t-1}^n)$;

23 3 $c_t^n := k$;

24 4 $(\boldsymbol{\mu}_t^n, \boldsymbol{\Sigma}_t^n, L_{tk}^n) = \text{KFupdate}(\boldsymbol{\mu}_{t-1}^n, \boldsymbol{\Sigma}_{t-1}^n, \mathbf{y}_t, \boldsymbol{\theta}_k)$;

25 5 $\tilde{w}_t^n = \tilde{w}_{t-1}^n L_{tk}^n$;

26 6 Normalize weights: $W_t^n = \frac{\tilde{w}_t^n}{\sum_{n'} \tilde{w}_t^{n'}}$;

27 7 Compute ESS = $\frac{1}{\sum_{n=1}^N (W_t^n)^2}$;

28 8 **if** $ESS < ESS_{\min}$ **then**

29 9 $A_t^{1:N} = \text{Resample}(W_t^{1:N})$

30 10 $c_t^{1:N} = \mathbf{c}_t^{\mathbf{A}_t}, \boldsymbol{\mu}_t^{1:N} = \boldsymbol{\mu}_t^{\mathbf{A}_t}, \boldsymbol{\Sigma}_t^{1:N} = \boldsymbol{\Sigma}_t^{\mathbf{A}_t}$;

31 11 $W_t^n = 1/N$;

32

33 An improved version of the algorothm can be developed based on the fact that we are sampling a

34 discrete state space. At each step, we propagate each of the N old particles through all K possible

35 transition models. We then compute the weight for all NK new particles, and sample from this to

36 get the final set of N particles. This latter step can be done using the **optimal resampling** method

37 of [FC03], which will stochastically select the particles with the largest weight, while also ensuring

38 the result is an unbiased approximation. In addition, this approach ensures that we do not have

39 duplicate particles, which is wasteful and unnecessary when the state space is discrete.

40

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 $p(\mathbf{z}_t | \mathbf{z}_{t-1}, c_t = k) = \mathcal{N}(\mathbf{z}_t | \mathbf{F}\mathbf{z}_{t-1} + \mathbf{B}_k \mathbf{u}_t, \mathbf{Q})$ (13.81)

5 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

6
$$\mathbf{F} = \begin{pmatrix} 1 & \Delta & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta \\ 0 & 0 & 0 & 1 \end{pmatrix}$$
 (13.82)

7 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
8 vectors are $\mathbf{b}_1 = (0, 0, 0, 0)$, $\mathbf{b}_2 = (-1.225, -0.35, 1.225, 0.35)$ and $\mathbf{b}_3 = (1.225, 0.35, -1.225, -0.35)$.
9 Thus the system will turn in different directions depending on the discrete state. We set the
10 observation model to $\mathbf{H} = \mathbf{I}$, and $\mathbf{R} = 10\text{diag}(2, 1, 2, 1)$. The discrete state transition matrix is given
11 by

12
$$\mathbf{A} = \begin{pmatrix} 0.8 & 0.1 & 0.1 \\ 0.1 & 0.8 & 0.1 \\ 0.1 & 0.1 & 0.8 \end{pmatrix}$$
 (13.83)

13 Figure 13.10a shows some observations, and the true state of the system, from a sample run, for
14 100 steps. The colors denote the discrete state, and the location of the symbol denotes the (x, y)
15 location. The small dots represent noisy observations. Figure 13.10b shows the estimate of the state
16 computed using RBPF with the optimal proposal with 1000 particles. In Figure 13.10c, we show the
17 analogous estimate using the bootstrap filter; we see that performance is worse.

18 In Figure 13.11a and Figure 13.11b, we show the posterior marginals of the (x, y) locations over
19 time. Figure 13.12a shows the true discrete state, and Figure 13.12b shows the approximate MAP
20 distribution over states; this has a classification error rate of 29%, but occasionally misclassifying
21 isolated time steps does not significantly hurt estimation of the continuous states, as we can see
22 from Figure 13.10b.

23 **13.5.2 FastSLAM**

24 We discussed the problem of simultaneous localization and mapping or SLAM in Section 31.3.2, where
25 we pointed out that the exact inference, even in the linear-Gaussian case, can be intractable for large
26 numbers of landmarks, due to the $K \times K$ covariance matrix. However, conditional on knowing the
27 robot's path, $\mathbf{r}_{1:t}$, the landmark locations are independent, i.e., $p(\mathbf{l}_t | \mathbf{r}_{1:t}, \mathbf{y}_{1:t}) = \prod_{k=1}^K p(\mathbf{l}_t^k | \mathbf{r}_{1:t}, \mathbf{y}_{1:t})$.
28 This can be seen by looking at the DGM in Figure 31.9. We can therefore sample the trajectory
29 using particle filtering, and apply Kalman filtering to each landmark independently.

30 In more detail, we sampling the trajectory, $\mathbf{r}_{1:t}$, assuming the map is known, as in Monte Carlo
31 localization described in Section 13.3.3. Given each sampled trajectory, we can apply the Kalman
32 filter to the remaining continuous latent variables, \mathbf{l}_t ; since these are conditionally independent given
33 $\mathbf{r}_{1:t}$, the joint posterior over the K landmarks factorizes into a product of K 2d Gaussians. Thus we

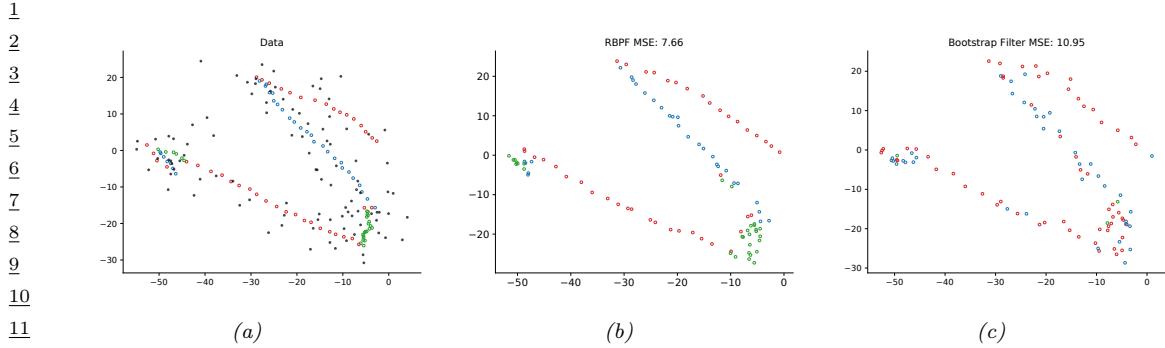


Figure 13.10: Illustration of state estimation for a switching linear model. (a) Black dots are observations, hollow circles are the true location, colors represent the discrete state. (b) Estimate from RBPF. Generated by `rpbf_maneuver_demo.py`. (c) Estimate from bootstrap filter. Generated by `bootstrap_filter_maneuver_demo.py`.

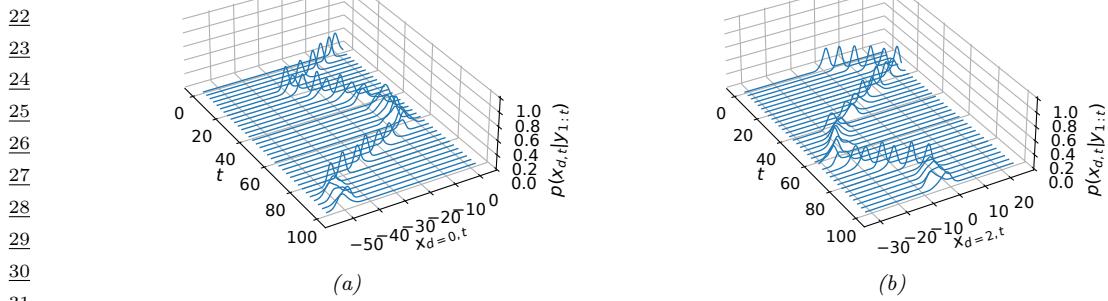


Figure 13.11: Posterior marginals of the location of the object over time, derived from the mixture of Gaussian representation. (a) x location (dimension 0). (b) y location (dimension 2). Generated by `rpbf_maneuver_demo.py`.

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

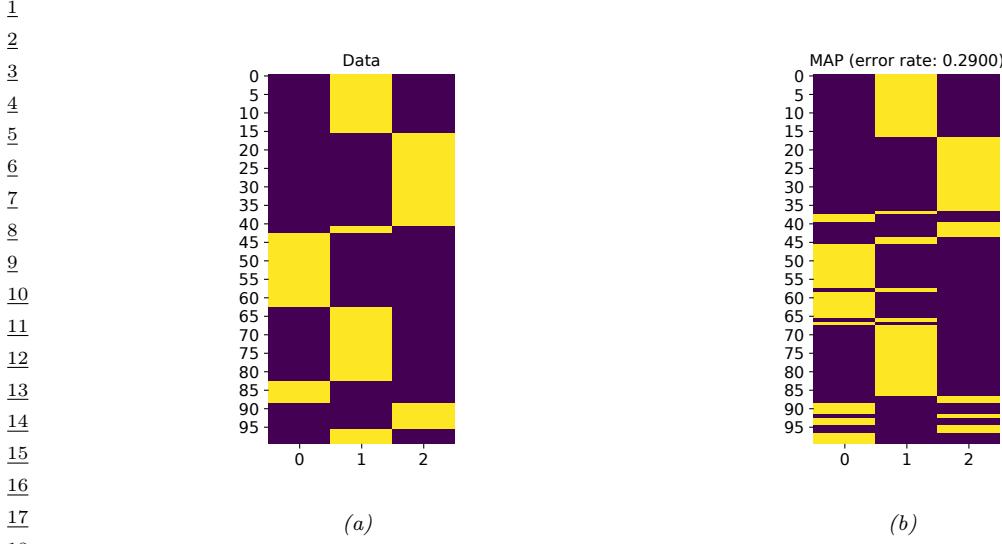


Figure 13.12: (a) Ground truth discrete state vs time, (b) Posterior distribution for the discrete state, derived from the particle representation. Generated by `rbspf_maneuver_demo.py`.

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

$$\tilde{w}_t = \frac{\pi_t(\mathbf{z}_{1:t})}{\bar{\pi}_{t-1}(\mathbf{z}_{1:t-1})M_t(\mathbf{z}_t|\mathbf{z}_{t-1})} \propto \frac{\tilde{\gamma}_t(\mathbf{z}_t)}{\tilde{\gamma}_{t-1}(\mathbf{z}_{t-1})} \frac{L_{t-1}(\mathbf{z}_{t-1}|\mathbf{z}_t)}{M_t(\mathbf{z}_t|\mathbf{z}_{t-1})} \quad (13.85)$$

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 $\tilde{\gamma}(\boldsymbol{\theta}) = \pi_0(\boldsymbol{\theta}) p(\mathcal{D}|\boldsymbol{\theta})$ 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   $W_t^n = w_t^n / (\sum_{m=1}^N w_t^m)$ 

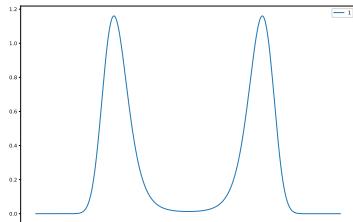
```

13.6.2.1 Example: sampling from a 1d bimodal distribution

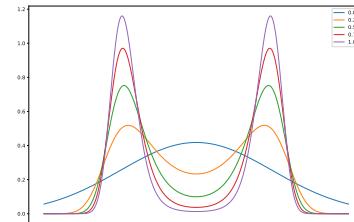
Consider the simple distribution

$$p(\boldsymbol{\theta}) \propto \mathcal{N}(\boldsymbol{\theta} | \mathbf{0}, \mathbf{I}) \exp(-\mathcal{E}(\boldsymbol{\theta})) \quad (13.96)$$

1
2
3
4
5
6
7
8
9
10
11
12
13



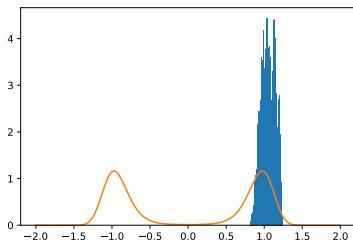
(a)



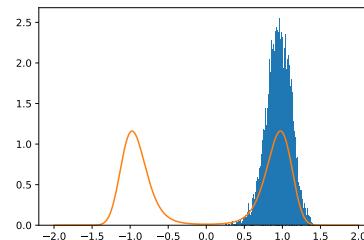
(b)

14
15 *Figure 13.13: (a) Illustration of a bimodal target distribution. (b) Tempered versions of the target at different
16 inverse temperatures, from $\lambda_T = 1$ down to $\lambda_1 = 0$. Generated by [smc_tempered_1d_bimodal.ipynb](#).*

17
18
19
20

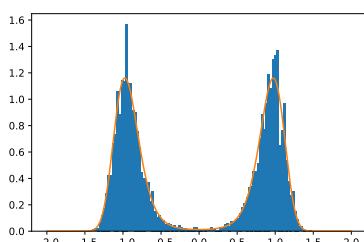


(a)

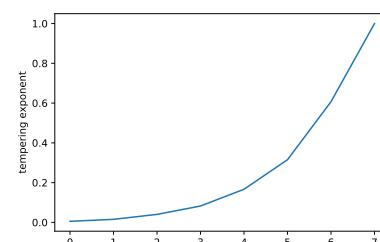


(b)

30
31
32
33
34
35
36
37
38
39
40
41



(c)



(d)

42 *Figure 13.14: Sampling from the bimodal distribution in Figure 13.13a. (a) HMC. (b) NUTS. (c) Tem-
43 pered SMC with HMC kernel (single step). (d) Adaptive inverse temperature schedule. Generated by
44 [smc_tempered_1d_bimodal.ipynb](#).*

45
46
47

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 **Algorithm 23:** IBIS (SMC for static parameters)

2 1 $\theta_0^n \sim \pi_0(\theta_0)$
3 2 $w_0^n = p(y_0 | \theta_0^n)$
4 3 $W_0^n = w_0^n / \sum_{m=1}^N w_0^m$
5 4 **for** $t = 1 : T$ **do**
6 5 **if** $ESS(W_{t-1}^{1:N}) < ESS_{\min}$ **then**
7 6 $A_t^{1:N} \sim \text{resample}(W_{t-1}^{1:N})$
8 7 $\hat{w}_{t-1}^n = 1$
9 8 $\theta_t^n \sim M_t(\theta_{t-1}^{A_t^n}, \cdot)$
10 9 **else**
11 10 $A_t^n = n$
12 11 $\hat{w}_{t-1}^n = w_{t-1}^n$
13 12 $\theta_t^n = \theta_{t-1}^n$
14 13 $w_t^n = \hat{w}_{t-1}^n p(y_t | y_{0:t-1}, \theta_t^n)$
15 14 $W_t^n = w_t^n / \sum_{m=1}^N w_t^m$

16

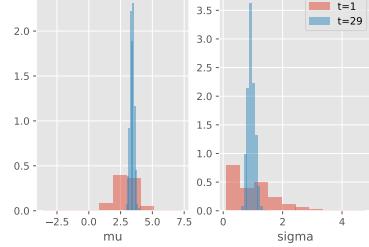
17

18

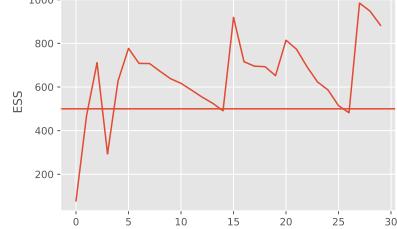
19

20

21



(a)



(b)

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

Figure 13.15: Illustration of IBIS applied to 30 samples from $\mathcal{N}(\mu = 3.14, \sigma = 1)$. (a) Posterior approximation after $t = 1$ and $t = 29$ observations. (b) Effective sample size over time. The sudden jumps up occur whenever resampling is triggered, which happens when the ESS drops below 500. Generated by [smc_ibis_1d.ipynb](#).

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
2 thresholds:

$$\pi_t(\boldsymbol{\theta}) = \frac{1}{Z_t} \mathbb{I}(S(\boldsymbol{\theta}) \geq \lambda_t) \pi_0(\boldsymbol{\theta}) \quad (13.99)$$

3
4
5
6 with $\lambda_0 < \dots < \lambda_T = \lambda^*$. We can tackle this using likelihood tempering, where the “likelihood” is
7 the function

$$G_t(\boldsymbol{\theta}_t) = \mathbb{I}(S(\boldsymbol{\theta}_t) \geq \lambda_t) \quad (13.100)$$

8
9 We can use SMC to generate samples from the final distribution π_T . We may also be interested in
10 estimating

$$Z_T = p(S(\boldsymbol{\theta}) \geq \lambda_T) \quad (13.101)$$

11
12 where the probability is taken wrt $\pi_0(\boldsymbol{\theta})$.

13
14 We can adaptively set the thresholds λ_t as follows: at each step, sort the samples by their fitness,
15 and set λ_t to the α ’th quantile. For example, if we set $\alpha = 0.5$, we keep the top 50% fittest particles.
16 This ensures the ESS equals the minimum threshold at each step. For details, see [Cér+12].

17
18 Note that this method is very similar to the **cross-entropy method** (see supplementary material).
19 The difference is that CEM fits a parametric distribution (e.g., a Gaussian) to the particles at each
20 step and samples from that, rather than using a Markov kernel.

21 13.6.5 SMC-ABC and likelihood-free inference

22
23 The term **likelihood-free inference** refers to estimating the parameters $\boldsymbol{\theta}$ of a blackbox from which
24 we can sample data, $\mathbf{y} \sim p(\mathbf{y}|\boldsymbol{\theta})$, but where we cannot compute the probability of that sample.
25 Such models are called simulators, so this approach to inference is also called **simulation-based**
26 **inference** (see e.g., [Nea+08; CBL20; Gou+96]). These models are also called implicit models (see
27 Section 27.1).

28
29 If we want to approximate the posterior of a model with no known likelihood, we can use
30 **Approximate Bayesian Computation** or **ABC** (see e.g., [Bea19; SFB18; Gut+14; Pes+21]).
31 In this setting, we sample both parameters $\boldsymbol{\theta}$ and synthetic data \mathbf{y} such that the synthetic data
32 (generated from $\boldsymbol{\theta}$) is sufficiently close to the observed data \mathbf{y}^* , as judged by some distance score,
33 $d(\mathbf{y}, \mathbf{y}^*) < \epsilon$. (For high dimensional problems, we typically require $d(\mathbf{s}(\mathbf{y}), \mathbf{s}(\mathbf{y}^*)) < \epsilon$, where $\mathbf{s}(\mathbf{y})$ is
34 a low-dimensional summary statistic of the data.)

35 In **SMC-ABC**, we gradually decrease the discrepancy ϵ to get a series of distributions as follows:

$$\pi_t(\boldsymbol{\theta}, \mathbf{y}) = \frac{1}{Z_t} \pi_0(\boldsymbol{\theta}) p(\mathbf{y}|\boldsymbol{\theta}) \mathbb{I}(d(\mathbf{y}, \mathbf{y}^*) < \epsilon_t) \quad (13.102)$$

36
37 where $\epsilon_0 > \epsilon_1 > \dots$. This is similar to the rare event SMC samplers in Section 13.6.4, except that
38 we can’t directly evaluate the quality of a candidate $\boldsymbol{\theta}$, instead we must first convert it to data space
39 and make the comparison there. For details, see [DMDJ12].

40
41 Although SMC-ABC is popular in some fields, such as genetics and epidemiology, this method
42 is quite slow and does not scale to high dimensional problems. In such settings, a more efficient
43 approach is to train a generative model to **emulate** the simulator; if this model is parametric with a
44 tractable likelihood (e.g., a flow model), we can use the usual methods for posterior inference of its
45 parameters (including gradient based methods like HMC). See e.g., [Bre+20a] for details.

¹ ² 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 $\frac{\gamma_t(\theta_t)}{\gamma_{t-1}(\theta_t)}$. 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 $\theta_t^{1:N_\theta}$, and we use an inner PF on the m ’th such particle to estimate
⁸ $p(\mathbf{y}_{0:t} | \theta_t^m)$. For details, see [CP20b, Ch. 18].

⁹ Unfortunately, SMC² is not a recursive algorithm, so it cannot be used for online parameter
¹⁰ estimation. An online extension of this method, called **recursive nested particle filter**, was
¹¹ proposed in [CM18].

¹² ¹³ 13.7 Particle MCMC methods

¹⁴ In this section, we discuss some other sampling techniques that leverage the fact that SMC can give an
¹⁵ unbiased estimate of the normalization constant Z for the target distribution. This can be useful for
¹⁶ sampling with models where the exact likelihood is intractable. These are called **pseudo-marginal**
¹⁷ **methods** [AR09].

¹⁸ To be more precise, note that the SMC algorithm can be seen as mapping a stream of random
¹⁹ 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
²⁰ states that are sampled at each step (using the inverse CDF of the proposal), and random numbers
²¹ $\mathbf{u}_{a,1:T-1}^{1:N_s}$ to control the ancestor indices that are chosen (using the resampling algorithm), where each
²² $u_{z,t}^i, u_{a,t}^i \sim \text{Unif}(0, 1)$. The normalization constant is also a function of these random numbers, so we
²³ denote it $\hat{Z}_t(\mathbf{u})$, where

$$\hat{Z}_t(\mathbf{u}) = \prod_{s=1}^t \frac{1}{N_s} \sum_{n=1}^{N_s} \tilde{w}_t^n(\mathbf{u}) \quad (13.103)$$

³⁰ One can show (see e.g., [NLS19, p80]) that

$$\mathbb{E} [\hat{Z}_t(\mathbf{u})] = Z_t \quad (13.104)$$

³¹ where the expectation is wrt the distribution of \mathbf{u} , denoted $\tau(\mathbf{u})$. (Note that \mathbf{u} can be represented
³² by a random seed.) This allows us to plug SMC inside other MCMC algorithms, as we show below.
³³ Such methods are often used by **probabilistic programming systems** (see e.g., [Zho+20]),
³⁴ since PPLs often define models with many latent variable models defined implicitly (via sampling
³⁵ statements), as discussed in Section 4.5.4.

³⁶

³⁷ 13.7.1 Particle Marginal Metropolis Hastings

³⁸ 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
³⁹ 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
⁴⁰ a SSM). We can use Metropolis Hastings (Section 12.2) to avoid having to compute the partition
⁴¹ function $p(\mathbf{y})$. However, in many cases it is intractable to compute the likelihood $p(\mathbf{y}|\boldsymbol{\theta})$ itself, due
⁴²

1 to the need to integrate over \mathbf{h} . This makes it hard to compute the MH acceptance probability
2

$$\frac{3}{4} A = \min \left(1, \frac{p(\mathbf{y}|\boldsymbol{\theta}') p(\boldsymbol{\theta}') q(\boldsymbol{\theta}^{j-1}|\boldsymbol{\theta}')}{p(\mathbf{y}|\boldsymbol{\theta}^{j-1}) p(\boldsymbol{\theta}^{j-1}) q(\boldsymbol{\theta}'|\boldsymbol{\theta}^{j-1})} \right) \quad (13.105)$$

5 where $\boldsymbol{\theta}^{j-1}$ is the parameter vector at iteration $j - 1$, and we are proposing $\boldsymbol{\theta}'$ from $q(\boldsymbol{\theta}'|\boldsymbol{\theta}^{j-1})$.
7 However, we can use SMC to compute $\hat{Z}(\boldsymbol{\theta})$ as an unbiased approximation to $p(\mathbf{y}|\boldsymbol{\theta})$, which can be
8 used to evaluate the MH acceptance ratio:
9

$$\frac{10}{11} A = \min \left(1, \frac{\hat{Z}(\mathbf{u}', \boldsymbol{\theta}') p(\boldsymbol{\theta}') q(\boldsymbol{\theta}^{j-1}|\boldsymbol{\theta}')}{\hat{Z}(\mathbf{u}^{j-1}, \boldsymbol{\theta}^{j-1}) p(\boldsymbol{\theta}^{j-1}) q(\boldsymbol{\theta}'|\boldsymbol{\theta}^{j-1})} \right) \quad (13.106)$$

13 More precisely, we apply MH to an extended space, where we sample both the parameters $\boldsymbol{\theta}$ and the
14 randomness \mathbf{u} for SMC.

15 We can generalize the above to return samples of the latent states as well as the latent parameters,
16 by sampling a single trajectory from
17

$$\frac{18}{19} p(\mathbf{h}_{1:T}|\boldsymbol{\theta}, \mathbf{y}) \approx \hat{p}(\mathbf{h}|\boldsymbol{\theta}, \mathbf{y}, \mathbf{u}) = \sum_{i=1}^{N_s} W_T^i \delta(\mathbf{h}_{1:T} - \mathbf{h}_{1:T}^i) \quad (13.107)$$

21 by using the internal samples generated by SMC. Thus we can sample $\boldsymbol{\theta}$ and \mathbf{h} jointly. This is called
22 the **particle marginal Metropolis Hastings** (PMMH) algorithm [ADH10]. See Algorithm 24
23 for the pseudocode. (In practice, we don't need to keep $\boldsymbol{\theta}$ and the random vector \mathbf{u} , we can just keep
24 $\boldsymbol{\theta}$ and the scalar $\hat{Z}(\mathbf{u}, \boldsymbol{\theta})$.) See e.g. [DS15] for more practical details.
25

Algorithm 24: Particle Marginal Metropolis-Hastings

```

28 1 for  $j = 1 : J$  do
29 2   Sample  $\boldsymbol{\theta}' \sim q(\boldsymbol{\theta}'|\boldsymbol{\theta}^{j-1})$ ,  $\mathbf{u}' \sim \tau(\mathbf{u}')$ ,  $\mathbf{h}' \sim \hat{p}(\mathbf{h}'|\boldsymbol{\theta}', \mathbf{y}, \mathbf{u}')$ 
30 3   Compute  $\hat{Z}(\mathbf{u}', \boldsymbol{\theta}')$  using SMC
31 4   Compute  $A$  using Equation (13.106)
32 5   Sample  $u \sim \text{Unif}(0, 1)$ 
33 6   if  $u < A$  then
34 7     Set  $\boldsymbol{\theta}^j = \boldsymbol{\theta}'$ ,  $\mathbf{u}^j = \mathbf{u}'$ ,  $\mathbf{h}^j = \mathbf{h}'$ 
35 8   else
36 9     Set  $\boldsymbol{\theta}^j = \boldsymbol{\theta}^{j-1}$ ,  $\mathbf{u}^j = \mathbf{u}^{j-1}$ ,  $\mathbf{h}^j = \mathbf{h}^{j-1}$ 

```

40 13.7.2 Particle Independent Metropolis Hastings

41 Now suppose we just want to sample the latent states \mathbf{h} , with the parameters $\boldsymbol{\theta}$ being fixed. In
42 this case we can simplify PMMH algorithm by not sampling $\boldsymbol{\theta}$. Since the latent states \mathbf{h} are now
43 sampled independently of the state of the Markov chain, this is called the **particle independent**
44 **MH** algorithm. The acceptance ratio term also simplifies, since we can drop all terms involving $\boldsymbol{\theta}$.
45 See Algorithm 25 for the pseudocode.
46

1 **Algorithm 25:** Particle Independent Metropolis-Hastings

2 1 **for** $j = 1 : J$ **do**

3 2 Sample $\mathbf{u}' \sim \tau(\mathbf{u}')$, $\mathbf{h}' \sim \hat{p}(\mathbf{h}' | \boldsymbol{\theta}, \mathbf{y}, \mathbf{u}')$

4 3 Compute $\hat{Z}(\mathbf{u}', \boldsymbol{\theta})$ using SMC

5 4 Compute $A = \min\left(1, \frac{\hat{Z}(\mathbf{u}', \boldsymbol{\theta})}{\hat{Z}(\mathbf{u}^{j-1}, \boldsymbol{\theta})}\right)$

6 5 Sample $u \sim \text{Unif}(0, 1)$

7 6 **if** $u < A$ **then**

8 7 Set $\mathbf{u}^j = \mathbf{u}'$, $\mathbf{h}^j = \mathbf{h}'$

9 8 **else**

10 9 Set $\mathbf{u}^j = \mathbf{u}^{j-1}$, $\mathbf{h}^j = \mathbf{h}^{j-1}$

11

12

13

14

15

16 One might wonder what the advantage of PIMH is over just using SMC. The answer is that PIMH
17 can return unbiased estimates of smoothing expectations, such as
18

19
$$\pi(\varphi) = \int \varphi(\mathbf{h}_{1:T}) \pi(\mathbf{h}_{1:T} | \boldsymbol{\theta}, \mathbf{y}) d\mathbf{h}_{1:T} \quad (13.108)$$

20

21

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].

24

25 13.7.3 Particle Gibbs

26 In PMMH, we define a transition kernel that, given $(\boldsymbol{\theta}^{(j-1)}, \mathbf{h}^{(j-1)})$, generates a sample $(\boldsymbol{\theta}^{(j)}, \mathbf{h}^{(j)})$,
27 while leaving the target distribution invariant. Another way to perform this task is to use **particle**
28 **Gibbs sampling**, which avoids needing to specify any proposal distributions. In this approach, we
29 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
30 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
31 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.

34

35

36

37

38

39

40

41

42

43

44

45

46

47

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.,

¹ $f(\mathbf{x}; \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \phi(\mathbf{x})$ for some *fixed* feature transformation ϕ), then the model is called a generalized
² linear model or GLM, which we discuss in Chapter 15. If f is a non-linear, but differentiable, function
³ 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
⁴ f using a neural network (Chapter 16). Other types of predictive model, such as decision trees and
⁵ random forests, are discussed in the prequel to this book, [Mur22].
⁶

⁸ 14.1.2 Model fitting using ERM, MLE and MAP

⁹ In this section, we briefly discuss some methods used for fitting (parametric) models. The most
¹⁰ common approach is to use **maximum likelihood estimation** or **MLE**, which amounts to solving
¹¹ the following optimization problem:
¹²

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta} \in \Theta}{\operatorname{argmax}} p(\mathcal{D}|\boldsymbol{\theta}) = \underset{\boldsymbol{\theta} \in \Theta}{\operatorname{argmax}} \log p(\mathcal{D}|\boldsymbol{\theta}) \quad (14.1)$$

¹⁶ If the dataset is N iid data samples, the likelihood decomposes into a product of terms, $p(\mathcal{D}|\boldsymbol{\theta}) =$
¹⁷ $\prod_{n=1}^N p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta})$. Thus we can instead minimize the following (scaled) **negative log likelihood**:
¹⁸

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta} \in \Theta}{\operatorname{argmin}} \frac{1}{N} \sum_{n=1}^N [-\log p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta})] \quad (14.2)$$

²³ 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
²⁴ loss function to get

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta} \in \Theta}{\operatorname{argmin}} r(\boldsymbol{\theta}) \quad (14.3)$$

²⁸ where $r(\boldsymbol{\theta})$ is the **empirical risk**
²⁹

$$r(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N \ell_n(\boldsymbol{\theta}) \quad (14.4)$$

³⁴ This approach is called **empirical risk minimization** or **ERM**.

³⁵ ERM can easily result in **overfitting**, so it is common to add a penalty or regularizer term to get

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta} \in \Theta}{\operatorname{argmin}} r(\boldsymbol{\theta}) + \lambda C(\boldsymbol{\theta}) \quad (14.5)$$

³⁹ where $\lambda \geq 0$ controls the degree of regularization, and $C(\boldsymbol{\theta})$ is some complexity measure. If we use
⁴⁰ 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 $\lambda = 1$,
⁴¹ we recover the **MAP estimate**
⁴²

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta} \in \Theta}{\operatorname{argmax}} \log p(\mathcal{D}|\boldsymbol{\theta}) + \log \pi_0(\boldsymbol{\theta}) \quad (14.6)$$

⁴³ This can be solved using standard optimization methods (see Chapter 6).
⁴⁴

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 ² 14.2 Evaluating predictive models

³ In this section we discuss how to evaluate the quality of a trained discriminative model.

⁴ ⁵ ⁶ 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

$$\underline{13} \quad S(p_{\theta}, p^*) = \int p^*(\mathbf{x})p^*(y|\mathbf{x})S(p_{\theta}, (y, \mathbf{x}))dyd\mathbf{x} \quad (14.14)$$

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:

$$\underline{21} \quad S(p_{\theta}, p^*) = \mathbb{E}_{p^*(\mathbf{x})p^*(y|\mathbf{x})} [\log p_{\theta}(y|\mathbf{x})] \leq \mathbb{E}_{p^*(\mathbf{x})p^*(y|\mathbf{x})} [\log p^*(y|\mathbf{x})] \quad (14.15)$$

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:

$$\underline{27} \quad S(p_{\theta}, (y, \mathbf{x})) \triangleq \frac{1}{C} \sum_{c=1}^C (p_{\theta}(y=c|\mathbf{x}) - \mathbb{I}(y=c))^2 \quad (14.16)$$

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.

³³ 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

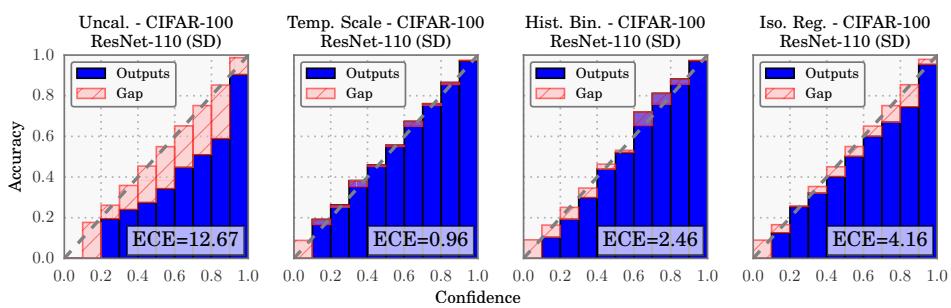


Figure 14.1: Reliability diagrams for the ResNet CNN image classifier [He+16b] applied to CIFAR-100 dataset. ECE is the expected calibration error, and measures the size of the red gap. Methods from left to right: original probabilities; after temperature scaling; after histogram binning; after isotonic 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 **14.2.2.2 Improving calibration**

7 In principle, training a classifier so it optimizes a proper scoring rule (such as NLL) should auto-
8 matically result in a well-calibrated classifier. In practice, however, unbalanced datasets can result
9 in poorly calibrated predictions. Below we discuss various ways for improving the calibration of
10 probabilistic classifiers, following [Guo+17].
11

12 **14.2.2.3 Platt scaling**

13 Let z be the log-odds, or logit, and $p = \sigma(z)$, produced by a probabilistic binary classifier. We wish
14 to convert this to a more calibrated value q . The simplest way to do this is known as **Platt scaling**,
15 and was proposed in [Pla00]. The idea is to compute $q = \sigma(az + b)$, where a and b are estimated via
16 maximum likelihood on a validation set.
17

18 In the multiclass case, we can extend Platt scaling by using matrix scaling: $\mathbf{q} = \mathcal{S}(\mathbf{W}\mathbf{z} + \mathbf{b})$, where
19 we estimate \mathbf{W} and \mathbf{b} via maximum likelihood on a validation set. Since \mathbf{W} has $K \times K$ parameters,
20 where K is the number of classes, this method can easily overfit, so in practice we restrict \mathbf{W} to be
21 diagonal.
22

23 **14.2.2.4 Nonparametric (histogram) methods**

24 Platt scaling makes a strong assumption about how the shape of the calibration curve. A more
25 flexible, nonparametric, method is to partition the predicted probabilities into bins, p_m , and to
26 estimate an empirical probability q_m for each such bin; we then replace p_m with q_m ; this is known
27 as **histogram binning** [ZE01a]. We can regularize this method by requiring that $q = f(p)$ be a
28 piecewise constant, monotonically non-decreasing function; this is known as **isotonic regression**
29 [ZE01a]. An alternative approach, known as the **scaling-binning calibrator**, is to apply a scaling
30 method (such as Platt scaling), and then to apply histogram binning to that. This has the advantage
31 of using the average of the scaled probabilities in each bin instead of the average of the observed
32 binary labels (see Figure 14.2). In [KLM19], they prove that this results in better calibration, due to
33 the lower variance of the estimator.
34

35 In the multiclass case, \mathbf{z} is the vector of logits, and $\mathbf{p} = \mathcal{S}(\mathbf{z})$ is the vector of probabilities. We
36 wish to convert this to a better calibrated version, \mathbf{q} . [ZE01b] propose to extend histogram binning
37 and isotonic regression to this case by applying the above binary method to each of the K one-vs-rest
38 problems, where K is the number of classes. However, this requires K separate calibration models,
39 and results in an unnormalized probability distribution.
40

41 **14.2.2.5 Temperature scaling**

42 In [Guo+17], they noticed empirically that the diagonal version of Platt scaling, when applied to
43 a variety of DNNs, often ended learning a vector of the form $\mathbf{w} = (c, c, \dots, c)$, for some constant c .
44 This suggests a simpler form of scaling, which they call **temperature scaling**: $\mathbf{q} = \mathcal{S}(\mathbf{z}/T)$, where
45

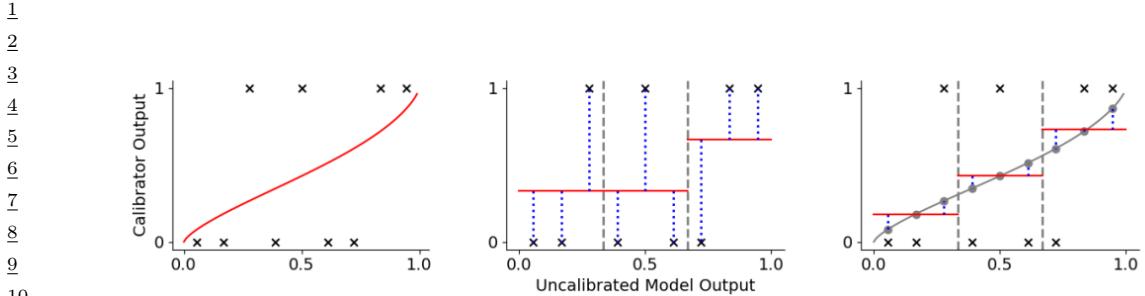


Figure 14.2: Visualization of 3 different approaches to calibrating a binary probabilistic classifier. Black crosses are the observed binary labels, red lines are the calibrated outputs. (a) Platt scaling. (b) Histogram binning with 3 bins. The output in each bin is the average of the binary labels in 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.

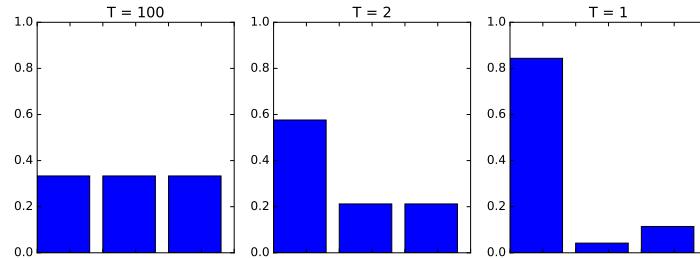


Figure 14.3: Softmax distribution $S(\mathbf{a}/T)$, where $\mathbf{a} = (3, 0, 1)$, at temperatures of $T = 100$, $T = 2$ and $T = 1$. When the temperature is high (left), the distribution is uniform, whereas when the temperature is low (right), the distribution is “spiky”, with most of its mass on 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 **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).

⁸ **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

$$\begin{aligned} \text{22} \quad p(Y_1^1 = 0, \dots, Y_\tau^1 = 0) &= \frac{1}{3^\tau} \end{aligned} \tag{14.22}$$

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

$$\begin{aligned} \text{27} \quad p(Y_1^2 = y_1, \dots, Y_\tau^2 = y_\tau) &= \begin{cases} 1/3 & \text{if } y_1 = \dots = y_\tau = 0 \\ 2/3 & \text{if } y_1 = \dots = y_\tau = 1 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{14.23}$$

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

$$\begin{aligned} \text{39} \quad P_T^B &= p(\mathbf{y}|\mathbf{x}, \mathcal{D}_T) \end{aligned} \tag{14.24}$$

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

⁴³

$$\begin{aligned} \text{44} \quad Q_T &= p(\mathbf{y}|\mathbf{x}, \mathcal{D}_T) \end{aligned} \tag{14.25}$$

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 $\boldsymbol{\theta}_m^j \sim Q(\boldsymbol{\theta}|\mathcal{D}_T^j)$ is a sample from the agent's posterior over the environment.

² The above assumes that $P(Y|X)$ is known; this will be the case if we use a synthetic data generator,
³ as in the the “neural testbed” in [Osb+21]. If we just have an J empirical distributions for $P^j(X, Y)$,
⁴ we can replace the KL with the cross entropy, which only differs by an additive constant:

$$\underline{d}_{\mathcal{E},Q}^{KL} = \mathbb{E}_{P(\mathbf{x}, \mathcal{D}_T, \mathcal{E})} [D_{\text{KL}}(P(\mathbf{y}|\mathbf{x}, \mathcal{E}) \| Q(\mathbf{y}|\mathbf{x}, \mathcal{D}_T))] \quad (14.36)$$

$$= \underbrace{\mathbb{E}_{P(\mathbf{x}, \mathbf{y}, \mathcal{E})} [\log P(\mathbf{y}|\mathbf{x}, \mathcal{E})]}_{\text{const}} - \underbrace{\mathbb{E}_{P(\mathbf{x}, \mathbf{y}, \mathcal{D}_T | \mathcal{E}) P(\mathcal{E})} [\log Q(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)]}_{d_{\mathcal{E},Q}^{CE}} \quad (14.37)$$

⁵

⁶ where the latter term is just the empirical negative log likelihood (NLL) of the agent on samples
⁷ from the environment. Hence if we rank agents in terms of their NLL or cross entropy $d_{\mathcal{E},Q}^{CE}$ we will
⁸ get the same results as ranking them by $d_{\mathcal{E},Q}^{KL}$, which will in turn give the same results as ranking
⁹ them by $d_{B,Q}^{KL}$.

¹⁰ In practice we can approximate the cross entropy as follows:

$$\hat{d}_{\mathcal{E},Q}^{CE} = -\frac{1}{JN} \sum_{j=1}^J \sum_{n=1}^N \log Q(\mathbf{y}_n^j | \mathbf{x}_n^j, \mathcal{D}_T^j) \quad (14.38)$$

¹¹ where $\mathcal{D}_T^j \sim P^j$, and $(\mathbf{x}_n^j, \mathbf{y}_n^j) \sim P^j$.

¹² An alternative to estimating the KL or NLL is to evaluate the joint predictive accuracy by using it
¹³ in a downstream task. In [Osb+21], they show that good predictive accuracy (for $\tau > 1$) correlates
¹⁴ with good performance on a bandit problem (see Section 36.4). In [WSG21] they show that good
¹⁵ predictive accuracy (for $\tau > 1$) results in good performance on a transductive active learning task.
¹⁶

¹⁷ 14.2.3.1 Proof of claim

¹⁸ We now prove Equation (14.29), based on [Lu+21]. First note that

$$\underline{d}_{\mathcal{E},Q}^{KL} = \mathbb{E}_{P(\mathbf{x}, \mathcal{D}_T, \mathcal{E}) P(\mathbf{y}|\mathbf{x}, \mathcal{E})} \left[\log \frac{P(\mathbf{y}|\mathbf{x}, \mathcal{E})}{Q(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)} \right] \quad (14.39)$$

$$= \mathbb{E} \left[\log \frac{P(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)}{Q(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)} \right] + \mathbb{E} \left[\log \frac{P(\mathbf{y}|\mathbf{x}, \mathcal{E})}{P(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)} \right] \quad (14.40)$$

¹⁹ For the first term in Equation (14.40) we have

$$\mathbb{E} \left[\log \frac{P(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)}{Q(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)} \right] = \sum P(\mathbf{x}, \mathbf{y}, \mathcal{D}_T) \log \frac{P(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)}{Q(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)} \quad (14.41)$$

$$= \sum P(\mathbf{x}, \mathcal{D}_T) \sum P(\mathbf{y}|\mathbf{x}, \mathcal{D}_T) \log \frac{P(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)}{Q(\mathbf{y}|\mathbf{x}, \mathcal{D}_T)} \quad (14.42)$$

$$= \mathbb{E}_{P(\mathbf{x}, \mathcal{D}_T)} [D_{\text{KL}}(P(\mathbf{y}|\mathbf{x}, \mathcal{D}_T) \| Q(\mathbf{y}|\mathbf{x}, \mathcal{D}_T))] = d_{B,Q}^{KL} \quad (14.43)$$

²⁰ We now show that the second term in Equation (14.40) reduces to the mutual information. We
²¹ exploit the fact that

$$P(\mathbf{y}|\mathbf{x}, \mathcal{E}) = P(\mathbf{y}|\mathcal{D}_T, \mathbf{x}, \mathcal{E}) = \frac{P(\mathcal{E}, \mathbf{y}|\mathcal{D}_T, \mathbf{x})}{P(\mathcal{E}|\mathcal{D}_T, \mathbf{x})} \quad (14.44)$$



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>.

¹ values of the score are less likely, so it is better to think of it as a non-conformity score.) Next we
² apply this score to a **calibration** set of n labeled examples, that was not used to train f , to get
³ $\mathcal{S} = \{s_i = s(\mathbf{x}_i, y_i) : i = 1 : n\}$. (In Section 14.3.4, we discuss what to do when we don't have a
⁴ calibration set.) The user specifies a desired confidence threshold α , say 0.1, and we then compute
⁵ 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
⁶ finite size of \mathcal{S} .) Finally, given a new test input, \mathbf{x}_{n+1} , we compute the prediction set to be
⁷

$$\mathcal{T}(\mathbf{x}_{n+1}) = \{y : s(\mathbf{x}_{n+1}, y) \leq \hat{q}\} \quad (14.49)$$

¹⁰ Intuitively, we include all the outputs y that are plausible given the input. See Figure 14.4 for an
¹¹ illustration.

¹² Remarkably, one can show the following general result

$$1 - \alpha \leq P^*(y^{n+1} \in \mathcal{T}(\mathbf{x}_{n+1})) \leq 1 - \alpha + \frac{1}{n+1} \quad (14.50)$$

¹⁷ where the probability is wrt the true distribution $P^*(\mathbf{x}_{n+1}, y_{n+1})$. We say that the prediction set
¹⁸ has a **coverage** level of $1 - \alpha$. This holds for any value of $n \geq 1$ and $\alpha \in [0, 1]$. The only assumption
¹⁹ is that the values (\mathbf{x}_i, y_i) are exchangeable, and hence the calibration scores s_i are also exchangeable.
²⁰ (We also assume the calibration set is drawn from P^* , although in Section 20.3.1.1, we discuss how
²¹ to handle covariate shift.)

²² 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
²³ assume the scores are distinct, for simplicity.) The score s_{n+1} is equally likely to fall in anywhere
²⁴ between the calibration points s_1, \dots, s_n , since the points are exchangeable. Hence

$$P^*(s_{n+1} \leq s_k) = \frac{k}{n+1} \quad (14.51)$$

²⁸ 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

$$P^*(y_{n+1} \in \mathcal{T}(\mathbf{x}_{n+1})) = P^*(s_{n+1} \leq \hat{q}) = \frac{\lceil(n+1)(1-\alpha)\rceil}{n+1} \geq 1 - \alpha \quad (14.52)$$

³³ For the proof of the upper bound, see [Lei+18].

³⁴ Although this result may seem like a “free lunch”, it is worth noting that we can always achieve
³⁵ a desired coverage level by defining the prediction set to be all possible labels. In this case, the
³⁶ prediction set will be independent of the input, but it will cover the true label $1 - \alpha$ of the time. To
³⁷ rule out some degenerate cases, we seek prediction sets that are as small as possible (although we
³⁸ allow for the set to be larger to harder examples), while meeting the coverage requirement. Achieving
³⁹ this goal requires that we define suitable conformal scores. Below we give some examples of how to
⁴⁰ compute conformal scores $s(\mathbf{x}, y)$ for different kinds of problem.

⁴¹

⁴² 14.3.1 Conformalizing classification

⁴⁴ The simplest way to apply conformal prediction to multiclass classification is to derive the conformal
⁴⁵ score from the softmax score assigned to the label using $s(\mathbf{x}, y) = 1 - f(\mathbf{x})_y$, so large values are
⁴⁶ considered less likely than small values. We compute the threshold \hat{q} as described above, and then we
⁴⁷

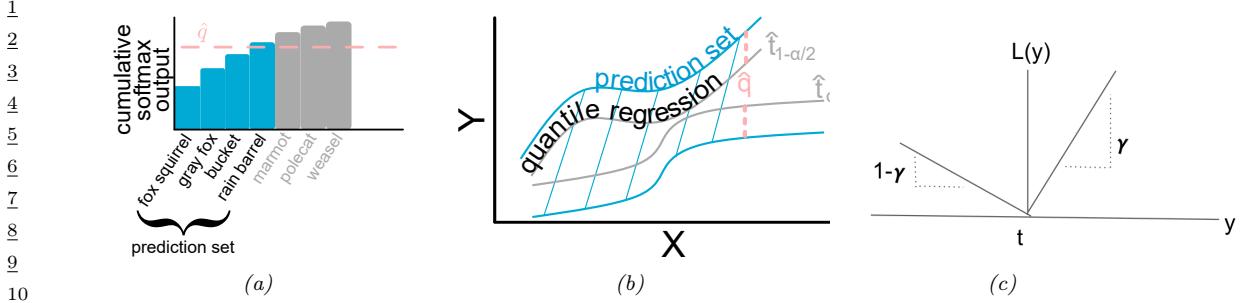


Figure 14.5: (a) Illustration of adaptive prediction set. From Figure 5 of [AB21]. Used with kind permission of Anastasios Angelopoulos. (b) Illustration of conformalized quantile regression. From Figure 6 of [AB21]. Used with kind permission of Anastasios Angelopoulos. (c) Illustration of pinball loss 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.

¹
² The regression quantiles are only approximately a 90% interval because the model may be
³ mismatched to the true distribution. However we can use conformal prediction to fix this. In
⁴ particular, let us define the conformal score to be

⁵
⁶ $s(\mathbf{x}, y) = \max(\hat{t}_{\alpha/2}(\mathbf{x}) - y, y - \hat{t}_{\alpha/2}(\mathbf{x}))$ (14.54)

⁷
⁸ In other words, $s(\mathbf{x}, y)$ is a positive measure of how far the value y is outside the prediction interval,
⁹ or is a negative measure if y is inside the prediction interval. We compute \hat{q} as before, and define the
¹⁰ conformal prediction interval to be

¹¹
¹² $\mathcal{T}(\mathbf{x}) = [\hat{t}_{\alpha/2}(\mathbf{x}) - \hat{q}, \hat{t}_{\alpha/2}(\mathbf{x}) + \hat{q}]$ (14.55)

¹³
¹⁴ This makes the quantile regression interval wider if \hat{q} is positive (if the base method was overconfident),
¹⁵ and narrower if \hat{q} is negative (if the base method was underconfident). See Figure 14.5b for an
¹⁶ illustration. This approach is called **conformalized quantile regression** or **CQR** [RPC19].

¹⁷ ¹⁸ 14.3.2.2 Conformalizing predicted variances

¹⁹
²⁰ There are many ways to define uncertainty scores $u(\mathbf{x})$, such as the predicted standard deviation,
²¹ from which we can derive a prediction interval using

²²
²³ $\mathcal{T}(\mathbf{x}) = [f(\mathbf{x}) - u(\mathbf{x})\hat{q}, f(\mathbf{x}) + u(\mathbf{x})\hat{q}]$ (14.56)

²⁴ Here \hat{q} is derived from the quantiles of the following conformal scores

²⁵
²⁶ $s(\mathbf{x}, y) = \frac{|y - f(\mathbf{x})|}{u(\mathbf{x})}$ (14.57)

²⁷
²⁸ The interval produced by this method tends to be wider than the one computed by CQR, since it
²⁹ extends an equal amount above and below the predicted value $f(\mathbf{x})$. In addition, the uncertainty
³⁰ measure $u(\mathbf{x})$ may not scale properly with α . Nevertheless, this is a simple post-hoc method that
³¹ can be applied to many regression methods without needing to retrain them.

³² ³³ 14.3.3 Conformalizing Bayes

³⁴
³⁵ Suppose we can compute the posterior predictive distribution $f(\mathbf{x})_y = p(y|\mathbf{x})$. If this is a perfect
³⁶ model, then the following prediction set would be optimal:

³⁷
³⁸ $\mathcal{S}(\mathbf{x}) = \{y : f(\mathbf{x})_y > t\}$, where t is chosen so $\int_{y \in \mathcal{S}(\mathbf{x})} f(\mathbf{x})_y dy = 1 - \alpha$ (14.58)

³⁹
⁴⁰ This set will not have the desired coverage if our modeling assumptions are wrong. However, we can
⁴¹ 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
⁴² all outputs above the chosen threshold. In [Hof21] they prove that this procedure has the smallest
⁴³ average size (Bayes risk) of any conformal procedure with $1 - \alpha$ coverage. Thus it is optimal in both
⁴⁴ the Bayesian and frequentist sense.

⁴⁵
⁴⁶

1 **14.3.4 What do we do if we don't have a calibration set?**

2 So far we have assumed access to a separate calibration set, which makes things simple. This is
3 called **split conformal prediction**. If we don't have enough data to adopt this splitting approach,
4 we can use **full conformal prediction** [VGS05], which requires fitting the model n times using a
5 leave-one-out type procedure. Alternatively we can use or the more efficient Bayesian add-one-in
6 importance sampling procedure of [FH21], or the **jackknife+** procedure of [Bar+19].
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

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:

$$\mu_n \triangleq \mathbb{E}[y_n | \mathbf{x}_n, \mathbf{w}, \sigma^2] = A'(\eta_n) \triangleq \ell^{-1}(\eta_n) \quad (15.2)$$

$$\text{V}[y_n | \mathbf{x}_n, \mathbf{w}, \sigma^2] = A''(\eta_n) \sigma^2 \quad (15.3)$$

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:

$$\log p(y_n | \mathbf{x}_n, \mathbf{w}, \sigma^2) = \frac{y_n \eta_n - \frac{\eta_n^2}{2}}{\sigma^2} - \frac{1}{2} \left(\frac{y_n^2}{\sigma^2} + \log(2\pi\sigma^2) \right) \quad (15.7)$$

6 We see that $A(\eta_n) = \eta_n^2/2$ and hence

$$\mathbb{E}[y_n] = \eta_n = \mathbf{w}^\top \mathbf{x}_n \quad (15.8)$$

$$\mathbb{V}[y_n] = \sigma^2 \quad (15.9)$$

11 See Section 15.2 for details on linear regression.

13 15.1.1.2 Binomial regression

14 If the response variable is the number of successes in N_n trials, $y_n \in \{0, \dots, N_n\}$, we can use
15 **binomial regression**, which is defined by

$$p(y_n | \mathbf{x}_n, N_n, \mathbf{w}) = \text{Bin}(y_n | \sigma(\mathbf{w}^\top \mathbf{x}_n), N_n) \quad (15.10)$$

18 We see that binary logistic regression is the special case when $N_n = 1$.

19 The log pdf is given by

$$\log p(y_n | \mathbf{x}_n, N_n, \mathbf{w}) = y_n \log \mu_n + (N_n - y_n) \log(1 - \mu_n) + \log \binom{N_n}{y_n} \quad (15.11)$$

$$= y_n \log\left(\frac{\mu_n}{1 - \mu_n}\right) + N_n \log(1 - \mu_n) + \log \binom{N_n}{y_n} \quad (15.12)$$

26 where $\mu_n = \sigma(\eta_n)$. To rewrite this in GLM form, let us define

$$\eta_n \triangleq \log \left[\frac{\mu_n}{(1 - \mu_n)} \right] = \log \left[\frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}_n}} \frac{1 + e^{-\mathbf{w}^\top \mathbf{x}_n}}{e^{-\mathbf{w}^\top \mathbf{x}_n}} \right] = \log \frac{1}{e^{-\mathbf{w}^\top \mathbf{x}_n}} = \mathbf{w}^\top \mathbf{x}_n \quad (15.13)$$

31 Hence we can write binomial regression in GLM form as follows

$$\log p(y_n | \mathbf{x}_n, N_n, \mathbf{w}) = y_n \eta_n - A(\eta_n) + h(y_n) \quad (15.14)$$

34 where $h(y_n) = \log \binom{N_n}{y_n}$ and

$$A(\eta_n) = -N_n \log(1 - \mu_n) = N_n \log(1 + e^{\eta_n}) \quad (15.15)$$

38 Hence

$$\mathbb{E}[y_n] = \frac{dA}{d\eta_n} = \frac{N_n e^{\eta_n}}{1 + e^{\eta_n}} = \frac{N_n}{1 + e^{-\eta_n}} = N_n \mu_n \quad (15.16)$$

42 and

$$\mathbb{V}[y_n] = \frac{d^2 A}{d\eta_n^2} = N_n \mu_n (1 - \mu_n) \quad (15.17)$$

46 See the supplementary material for an example of binomial regression.

47

15.1.1.3 Poisson regression

If the response variable is an integer count, $y_n \in \{0, 1, \dots\}$, we can use **Poisson regression**, which is defined by

$$p(y_n | \mathbf{x}_n, \mathbf{w}) = \text{Poi}(y_n | \exp(\mathbf{w}^\top \mathbf{x}_n)) \quad (15.18)$$

where

$$\text{Poi}(y | \mu) = e^{-\mu} \frac{\mu^y}{y!} \quad (15.19)$$

is the Poisson distribution. Poisson regression is widely used in bio-statistical applications, where y_n might represent the number of diseases of a given person or place, or the number of reads at a genomic location in a high-throughput sequencing context (see e.g., [Kua+09]).

The log pdf is given by

$$\log p(y_n | \mathbf{x}_n, \mathbf{w}) = y_n \log \mu_n - \mu_n - \log(y_n!) \quad (15.20)$$

where $\mu_n = \exp(\mathbf{w}^\top \mathbf{x}_n)$. Hence in GLM form we have

$$\log p(y_n | \mathbf{x}_n, \mathbf{w}) = y_n \eta_n - A(\eta_n) + h(y_n) \quad (15.21)$$

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

$$\mathbb{E}[y_n] = \frac{dA}{d\eta_n} = e^{\eta_n} = \mu_n \quad (15.22)$$

and

$$\mathbb{V}[y_n] = \frac{d^2A}{d\eta_n^2} = e^{2\eta_n} = \mu_n^2 \quad (15.23)$$

15.1.1.4 Zero-inflated Poisson regression

In many forms of count data, the number of observed 0s is larger than what a model might expect, even after taking into account the predictors. Intuitively, this is because there may be many ways to produce no outcome. For example, consider predicting sales data for a product. If the sales are 0, does it mean the product is unpopular (so the demand is very low), or was it simply sold out (implying the demand is high, but exceed supply)? Similar problems arise in genomics, epidemiology, etc.

To handle such situations, it is common to use a **zero-inflated Poisson** or **ZIP** model. The likelihood for this model is a mixture of two distributions: a spike at 0, and a standard Poisson. Formally, we define

$$\text{ZIP}(y | \rho, \lambda) = \begin{cases} \rho + (1 - \rho) \exp(-\lambda) & \text{if } y = 0 \\ (1 - \rho) \frac{\lambda^y \exp(-\lambda)}{y!} & \text{if } y > 0 \end{cases} \quad (15.24)$$

Here ρ is the prior probability of picking the spike, and λ is the rate of the Poisson. We see that there are two “mechanisms” for generating a 0: either (with probability ρ) we chose the spike, or (with probability $1 - \rho$) we simply generate a zero count just because the rate of the Poisson is so low. (This latter event has probability $\lambda^0 e^{-\lambda} / 0! = e^{-\lambda}$.)

15.1.2 GLMs with non-canonical link functions

We have seen how the mean parameters of the output distribution are given by $\mu = \ell^{-1}(\eta)$, where the function ℓ is the link function. There are several choices for this function, as we now discuss.

The **canonical link function** ℓ satisfies the property that $\theta = \ell(\mu)$, where θ are the canonical (natural) parameters. Hence

$$\theta = \ell(\mu) = \ell(\ell^{-1}(\eta)) = \eta \quad (15.25)$$

This is what we have assumed so far. For example, for the Bernoulli distribution, the canonical parameter is the log-odds $\eta = \log(\mu/(1-\mu))$, which is given by the logit transform

$$\eta = \ell(\mu) = \text{logit}(\mu) = \log\left(\frac{\mu}{1-\mu}\right) \quad (15.26)$$

The inverse of this is the sigmoid or logistic function

$$\mu = \ell^{-1}(\eta) = \sigma(\eta) = 1/(1 + e^{-\eta}) \quad (15.27)$$

However, we are free to use other kinds of link function. For example, in Section 15.4 we use

$$\eta = \ell(\mu) = \Phi^{-1}(\mu) \quad (15.28)$$

$$\mu = \ell^{-1}(\eta) = \Phi(\eta) \quad (15.29)$$

This is known as the **probit link function**.

Another link function that is sometimes used for binary responses is the **complementary log-log function**

$$\eta = \ell(\mu) = \log(-\log(1-\mu)) \quad (15.30)$$

This is used in applications where we either observe 0 events (denoted by $y = 0$) or one or more (denoted by $y = 1$), where events are assumed to be governed by a Poisson distribution with rate λ .

Let E be the number of events. The Poisson assumption means $p(E = 0) = \exp(-\lambda)$ and hence

$$p(y = 0) = (1 - \mu) = p(E = 0) = \exp(-\lambda) \quad (15.31)$$

Thus $\lambda = -\log(1 - \mu)$. When λ is a function of covariates, we need to ensure it is positive, so we use $\lambda = e^\eta$, and hence

$$\eta = \log(\lambda) = \log(-\log(1 - \mu)) \quad (15.32)$$

15.1.3 Maximum likelihood estimation

GLMs can be fit using similar methods to those that we used to fit logistic regression. In particular, the negative log-likelihood has the following form (ignoring constant terms):

$$\text{NLL}(\mathbf{w}) = -\log p(\mathcal{D}|\mathbf{w}) = -\frac{1}{\sigma^2} \sum_{n=1}^N \ell_n \quad (15.33)$$

1
2 where

3
4 $\ell_n \triangleq \eta_n y_n - A(\eta_n)$ (15.34)

5
6 where $\eta_n = \mathbf{w}^\top \mathbf{x}_n$. For notational simplicity, we will assume $\sigma^2 = 1$.

7 We can compute the gradient for a single term as follows:

8
9 $\mathbf{g}_n \triangleq \frac{\partial \ell_n}{\partial \mathbf{w}} = \frac{\partial \ell_n}{\partial \eta_n} \frac{\partial \eta_n}{\partial \mathbf{w}} = (y_n - A'(\eta_n)) \mathbf{x}_n = (y_n - \mu_n) \mathbf{x}_n$ (15.35)

10
11 where $\mu_n = f(\mathbf{w}^\top \mathbf{x}_n)$, and f is the inverse link function that maps from canonical parameters to
12 mean parameters. (For example, in the case of logistic regression, we have $\mu_n = \sigma(\mathbf{w}^\top \mathbf{x})$.) This
13 gradient expression can be used inside SGD, or some other gradient method, in the obvious way.

14 The Hessian is given by

15
16 $\mathbf{H} = \frac{\partial^2}{\partial \mathbf{w} \partial \mathbf{w}^\top} \text{NLL}(\mathbf{w}) = - \sum_{n=1}^N \frac{\partial \mathbf{g}_n}{\partial \mathbf{w}^\top}$ (15.36)

19 where

20
21 $\frac{\partial \mathbf{g}_n}{\partial \mathbf{w}^\top} = \frac{\partial \mathbf{g}_n}{\partial \mu_n} \frac{\partial \mu_n}{\partial \mathbf{w}^\top} = -\mathbf{x}_n f'(\mathbf{w}^\top \mathbf{x}_n) \mathbf{x}_n^\top$ (15.37)

23 Hence

25
26 $\mathbf{H} = \sum_{n=1}^N f'(\eta_n) \mathbf{x}_n \mathbf{x}_n^\top$ (15.38)

28
29 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
30 general, we see that the Hessian is positive definite, since $f'(\eta_n) > 0$; hence the negative log likelihood
31 is convex, so the MLE for a GLM is unique (assuming $f(\eta_n) > 0$ for all n).

32 15.1.4 Bayesian inference

34 To perform Bayesian inference of the parameters, we first need to specify a prior. Choosing a suitable
35 prior depends on the form of link function. For example, a “flat” or “uninformative” prior on the
36 offset term $\alpha \in \mathbb{R}$ will not translate to an uninformative prior on the probability scale if we pass α
37 through a sigmoid, as we discuss in Section 15.3.3.

38 Once we have chosen the prior, we can compute the posterior using a variety of approximate
39 inference methods. For small sample sizes, HMC (Section 12.5) is the easiest to use, since you just
40 need to write down the log likelihood and log prior, and use autograd to compute derivatives and pass
41 them to the HMC engine.¹ For large datasets, stochastic variational inference (Section 10.3.2) is often
42 more scalable. Of course, many other inference methods are possible, such as Laplace approximation
43 (Section 7.4.3), SMC (Section 13.6), etc.

45 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 ² 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 ⁷ 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 ¹² 15.2.1.1 Noise variance is known

¹³ The conjugate prior for linear regression has the following form:
¹⁴

$$\begin{aligned} \text{¹⁵ } p(\mathbf{w}) &= \mathcal{N}(\mathbf{w} | \tilde{\mathbf{w}}, \breve{\Sigma}) \\ \text{¹⁶} \end{aligned} \tag{15.39}$$

¹⁷ We often use $\tilde{\mathbf{w}} = \mathbf{0}$ as the prior mean and $\breve{\Sigma} = \tau^2 \mathbf{I}_D$ 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

$$\begin{aligned} \text{²² } \ell(\mathbf{w}) &= p(\mathcal{D} | \mathbf{w}, \sigma^2) = \prod_{n=1}^N p(y_n | \mathbf{w}^\top \mathbf{x}, \sigma^2) = \mathcal{N}(\mathbf{y} | \mathbf{X}\mathbf{w}, \sigma^2 \mathbf{I}_N) \\ \text{²³} \end{aligned} \tag{15.40}$$

²⁵ ²⁶ where \mathbf{I}_N is the $N \times N$ identity matrix. We can then use Bayes rule for Gaussians (Equation (2.59))
²⁷ to derive the posterior, which is as follows:

$$\begin{aligned} \text{²⁸ } p(\mathbf{w} | \mathbf{X}, \mathbf{y}, \sigma^2) &\propto \mathcal{N}(\mathbf{w} | \tilde{\mathbf{w}}, \breve{\Sigma}) \mathcal{N}(\mathbf{y} | \mathbf{X}\mathbf{w}, \sigma^2 \mathbf{I}_N) = \mathcal{N}(\mathbf{w} | \hat{\mathbf{w}}, \hat{\Sigma}) \\ \text{²⁹} \end{aligned} \tag{15.41}$$

$$\begin{aligned} \text{³⁰ } \hat{\mathbf{w}} &\triangleq \hat{\Sigma}^{-1} (\breve{\Sigma}^{-1} \tilde{\mathbf{w}} + \frac{1}{\sigma^2} \mathbf{X}^\top \mathbf{y}) \\ \text{³¹} \end{aligned} \tag{15.42}$$

$$\begin{aligned} \text{³² } \hat{\Sigma} &\triangleq (\breve{\Sigma}^{-1} + \frac{1}{\sigma^2} \mathbf{X}^\top \mathbf{X})^{-1} \\ \text{³³} \end{aligned} \tag{15.43}$$

³⁴ ³⁵ where $\hat{\mathbf{w}}$ is the posterior mean, and $\hat{\Sigma}$ is the posterior covariance.

³⁶ ³⁷ 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 $\breve{\Sigma} = \tau^2 \mathbf{I}$. In this case, the posterior mean becomes
⁴⁴

$$\begin{aligned} \text{⁴⁵ } \hat{\mathbf{w}} &= \frac{1}{\sigma^2} \hat{\Sigma} \mathbf{X}^\top \mathbf{y} = (\frac{\sigma^2}{\tau^2} \mathbf{I} + \mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \\ \text{⁴⁶} \end{aligned} \tag{15.44}$$

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

$$p(\mathbf{w} | \sigma^2) = \mathcal{N}(\mathbf{w} | \check{\mathbf{w}}, \sigma^2 \check{\Sigma}) \quad (15.48)$$

For the noise variance σ^2 , the conjugate prior is based on the inverse Gamma distribution, which has the form

$$\text{IG}(\sigma^2 | \check{a}, \check{b}) = \frac{\check{b}^{\check{a}}}{\Gamma(\check{a})} (\sigma^2)^{-(\check{a}+1)} \exp(-\frac{\check{b}}{\sigma^2}) \quad (15.49)$$

(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:

$$\text{NIG}(\mathbf{w}, \sigma^2 | \check{\mathbf{w}}, \check{\Sigma}, \check{a}, \check{b}) \triangleq \mathcal{N}(\mathbf{w} | \check{\mathbf{w}}, \sigma^2 \check{\Sigma}) \text{IG}(\sigma^2 | \check{a}, \check{b}) \quad (15.50)$$

$$\begin{aligned} &= \frac{\check{b}^{\check{a}}}{(2\pi)^{D/2} |\check{\Sigma}|^{1/2} \Gamma(\check{a})} (\sigma^2)^{-(\check{a}+(D/2)+1)} \\ &\times \exp\left[-\frac{(\mathbf{w} - \check{\mathbf{w}})^\top \check{\Sigma}^{-1} (\mathbf{w} - \check{\mathbf{w}}) + 2\check{b}}{2\sigma^2}\right] \end{aligned} \quad (15.51)$$

This results in the following posterior:

$$p(\mathbf{w}, \sigma^2 | \mathcal{D}) = \text{NIG}(\mathbf{w}, \sigma^2 | \hat{\mathbf{w}}, \hat{\Sigma}, \hat{a}, \hat{b}) \quad (15.52)$$

$$\hat{\mathbf{w}} = \hat{\Sigma} (\check{\Sigma}^{-1} \check{\mathbf{w}} + \mathbf{X}^\top \mathbf{y}) \quad (15.53)$$

$$\hat{\Sigma} = (\check{\Sigma}^{-1} + \mathbf{X}^\top \mathbf{X})^{-1} \quad (15.54)$$

$$\hat{a} = \check{a} + N/2 \quad (15.55)$$

$$\hat{b} = \check{b} + \frac{1}{2} \left(\check{\mathbf{w}}^\top \check{\Sigma}^{-1} \check{\mathbf{w}} + \mathbf{y}^\top \mathbf{y} - \hat{\mathbf{w}}^\top \hat{\Sigma}^{-1} \hat{\mathbf{w}} \right) \quad (15.56)$$

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, \bar{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

$$\frac{7}{8} p(\sigma^2 | \mathcal{D}) = \int p(\mathbf{w} | \sigma^2, \mathcal{D}) p(\sigma^2 | \mathcal{D}) d\mathbf{w} = \text{IG}(\sigma^2 | \hat{a}, \hat{b}) \quad (15.57)$$

⁹ For the regression weights, it can be shown that

$$\frac{10}{12} p(\mathbf{w} | \mathcal{D}) = \int p(\mathbf{w} | \sigma^2, \mathcal{D}) p(\sigma^2 | \mathcal{D}) d\sigma^2 = \mathcal{T}(\mathbf{w} | \hat{\mathbf{w}}, \frac{\hat{b}}{\hat{a}} \hat{\Sigma}, 2 \hat{a}) \quad (15.58)$$

¹⁵ 15.2.1.3 Posterior predictive distribution

¹⁶ In machine learning we usually care more about uncertainty (and accuracy) of our predictions, not
¹⁷ our parameter estimates. Fortunately, one can derive the posterior predictive distribution in closed
¹⁸ form. In particular, one can show that, given N' new test inputs $\tilde{\mathbf{X}}$, we have

$$\frac{20}{21} p(\tilde{\mathbf{y}} | \tilde{\mathbf{X}}, \mathcal{D}) = \int \int p(\tilde{\mathbf{y}} | \tilde{\mathbf{X}}, \mathbf{w}, \sigma^2) p(\mathbf{w}, \sigma^2 | \mathcal{D}) d\mathbf{w} d\sigma^2 \quad (15.59)$$

$$\frac{23}{24} = \int \int \mathcal{N}(\tilde{\mathbf{y}} | \tilde{\mathbf{X}}\mathbf{w}, \sigma^2 \mathbf{I}_{N'}) \text{NIG}(\mathbf{w}, \sigma^2 | \hat{\mathbf{w}}, \hat{\Sigma}, \hat{a}, \hat{b}) d\mathbf{w} d\sigma^2 \quad (15.60)$$

$$\frac{25}{26} = \mathcal{T}(\tilde{\mathbf{y}} | \tilde{\mathbf{X}} \hat{\mathbf{w}}, \frac{\hat{b}}{\hat{a}} (\mathbf{I}_{N'} + \tilde{\mathbf{X}} \hat{\Sigma} \tilde{\mathbf{X}}^\top), 2 \hat{a}) \quad (15.61)$$

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

³⁴

³⁵ 15.2.2 Uninformative priors

³⁶ A common criticism of Bayesian inference is the need to use a prior. This is sometimes thought to
³⁷ “pollute” the inferences one makes from the data. We can minimize the effect of the prior by using an
³⁸ uninformative prior, as we discussed in Section 3.4. Below we discuss various uninformative priors
³⁹ for linear regression.

⁴¹

⁴² 15.2.2.1 Jeffreys prior

⁴³ From Section 3.4.3.1, we know that the Jeffreys prior for the location parameter has the form
⁴⁴ $p(\mathbf{w}) \propto 1$, and from Section 3.4.3.2, we know that the Jeffreys prior for the scale factor has the
⁴⁵ form $p(\sigma) \propto \sigma^{-1}$. We can emulate these priors using an improper NIG prior with $\tilde{\mathbf{w}} = \mathbf{0}$, $\tilde{\Sigma} = \infty \mathbf{I}$,
⁴⁶

$\check{a} = -D/2$ 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.

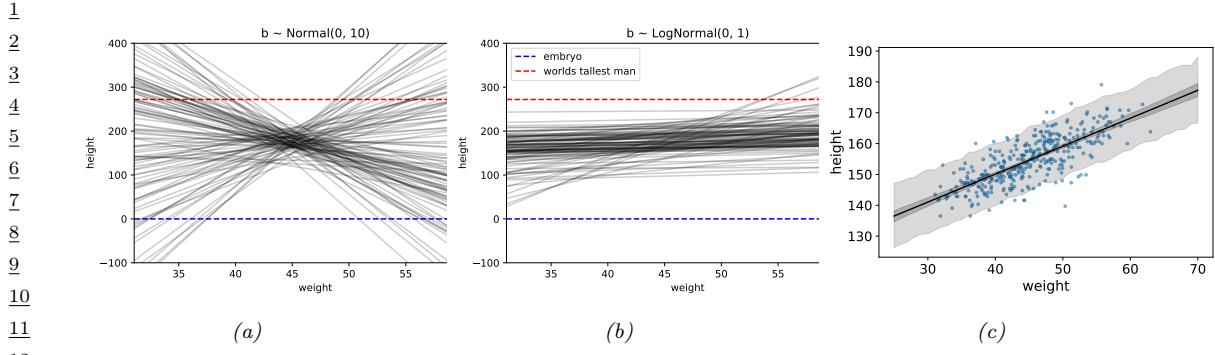


Figure 15.1: Linear regression for predicting height given weight, $y \sim \mathcal{N}(\alpha + \beta x, \sigma^2)$. (a) Prior predictive samples using a Gaussian prior for β . (b) Prior predictive samples using a Log-Gaussian prior for β . (c) Posterior predictive samples using the Log-Gaussian prior. 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 $\tilde{\alpha} = -D/2$, $\tilde{\beta} = 0$ (to ensure $p(\sigma^2) \propto 1$) and $\tilde{\mathbf{w}} = \mathbf{0}$ and $\tilde{\Sigma} = g(\mathbf{X}^T \mathbf{X})^{-1}$, 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]).

3 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
4 the inputs. The observation model is given by

$$\small{y_i \sim \mathcal{N}(\mu_i, \sigma)} \quad (15.78)$$

$$\small{\mu_i = \alpha + \beta(x_i - \bar{x})} \quad (15.79)$$

5 We see that the intercept α is the predicted output if $x_i = \bar{x}$, and the slope β is the predicted change
6 in height per unit change in weight above or below the average weight.

7 The question is: what priors should we use? To be truly Bayesian, we should set these before
8 looking at the data. A sensible prior for α is the height of a “typical person”, with some spread. We
9 use $\alpha \sim \mathcal{N}(178, 20)$, since the author of the “Rethinking Statistics” book from which this example is
10 taken is 178cm. By using a standard deviation of 20, the prior puts 95% probability on the broad
11 range of 178 ± 40 .

12 What about the prior for β ? It is tempting to use a **vague prior**, or **weak prior**, such as
13 $\beta \sim \mathcal{N}(0, 10)$, which is similar to a flat (uniform) prior, but more concentrated at 0 (a form of mild
14 regularization). To see if this is reasonable, we can compute samples from the **prior predictive**
15 **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,
16 for different samples $s = 1 : S$. The results are shown in Figure 15.1a. We see that this is not a very
17 sensible prior. For example, we see that it suggests that it is just as likely for the height to decrease
18 with weight as increase with weight, which is not plausible. In addition, it predicts heights which
19 are larger than the world’s tallest person (272 cm) and smaller than the world’s shortest person (an
20 embryo, of size 0).

21 We can encode the monotonically increasing relationship between weight and height by restricting
22 β to be positive. An easy way to do this is to use a log-normal or log-Gaussian prior. (If $\tilde{\beta} = \log(\beta)$
23 is Gaussian, then $e^{\tilde{\beta}}$ must be positive.) Specifically, we will assume $\beta \sim \mathcal{LN}(0, 1)$. Samples from this
24 prior are shown in Figure 15.1b. This is much more reasonable.

25 Finally we must choose a prior over σ . In [McE20] they use $\sigma \sim \text{Unif}(0, 50)$. This ensures that σ
26 is positive, and that the prior predictive distribution for the output is within 100cm of the average
27 height. However, it is usually easier to specify the expected value for σ than an upper bound. To
28 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
29 standard deviation that we expect. For example, we can use the empirical standard deviation of the
30 data.

31 Since these priors are no longer conjugate, we cannot compute the posterior in closed form. However,
32 we can use a variety of approximate inference methods. In this simple example, it suffices to use a
33 quadratic (Laplace) approximation (see Section 7.4.3). The results are shown in Figure 15.1c, and
34 look sensible.

35 So far, we have only considered a subset of the data, corresponding to adults over the age of 18. If
36 we include children, we find that the mapping from weight to height is nonlinear. This is illustrated
37 in Figure 15.2a. We can fix this problem by using **polynomial regression**. For example, consider a
38 quadratic expansion of the standardized features x_i :

$$\small{\mu_i = \alpha + \beta_1 x_i + \beta_2 x_i^2} \quad (15.80)$$

39 If we use a log-Gaussian prior for β_2 , we find that the model is too constrained, and it underfits.
40 This is illustrated in Figure 15.2b. The reason is that we need to use an inverted quadratic with
41

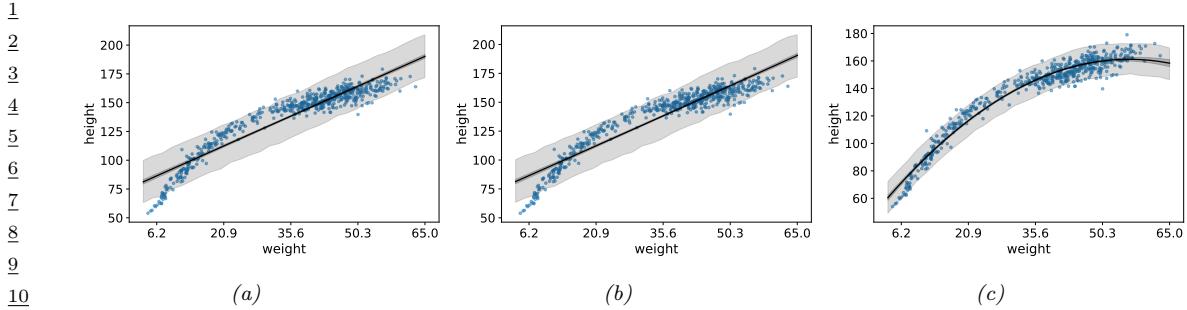


Figure 15.2: Linear regression for predicting height given weight for the full dataset (including children) using polynomial regression. (a) Posterior fit for linear model with log-Gaussian prior for β_1 . (b) Posterior fit for quadratic model with log-Gaussian prior for β_2 . (c) Posterior fit for quadratic 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

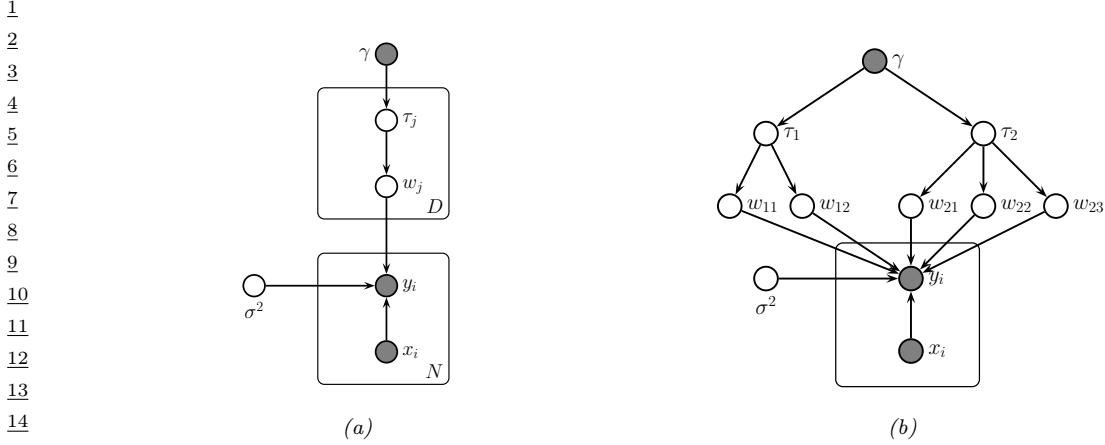


Figure 15.3: (a) Representing lasso using a Gaussian scale mixture prior. (b) Graphical model for group lasso with 2 groups, the first has size $G_1 = 2$, the second has size $G_2 = 3$.

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

$$p(\mathbf{y}, \mathbf{w}, \boldsymbol{\tau}, \sigma^2 | \mathbf{X}) = \mathcal{N}(\mathbf{y} | \mathbf{X}\mathbf{w}, \sigma^2 \mathbf{I}_N) \left[\prod_j \mathcal{N}(w_j | 0, \sigma^2 \tau_j^2) \text{Ga}(\tau_j^2 | 1, \lambda^2/2) \right] p(\sigma^2) \quad (15.87)$$

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

$$\mathbf{w}_g | \sigma^2, \tau_g^2 \sim \mathcal{N}(\mathbf{0}, \sigma^2 \tau_g^2 \mathbf{I}_{d_g}) \quad (15.88)$$

$$\tau_g^2 \sim \text{Ga}\left(\frac{d_g + 1}{2}, \frac{\lambda^2}{2}\right) \quad (15.89)$$

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 15.2.6 Horseshoe prior

²⁵ The Laplace prior is not suitable for sparse Bayesian models, because posterior samples are not
²⁶ sparse. The spike and slab prior does not have this problem but is often too slow to use (although see
²⁷ [BRG20]). Fortunately, it is possible to devise continuous priors (without discrete latent variables)
²⁸ that are both sparse and computationally efficient. One popular prior of this type is the **horseshoe**
²⁹ prior [CPS10], so-named because of the shape of its density function.

³⁰ In the horseshoe prior, instead of using a Laplace prior for each weight, we use the following
³¹ Gaussian scale mixture:
³²

$$w_j \sim \mathcal{N}(0, \lambda_j^2 \tau^2) \quad (15.90)$$

$$\lambda_j \sim \mathcal{C}_+(0, 1) \quad (15.91)$$

$$\tau^2 \sim \mathcal{C}_+(0, 1) \quad (15.92)$$

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:

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\alpha}, \beta) = \int \mathcal{N}(\mathbf{y}|\mathbf{X}\mathbf{w}, (1/\beta)\mathbf{I}_N) \mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{A}^{-1}) d\mathbf{w} \quad (15.93)$$

$$= \mathcal{N}(\mathbf{y}|\mathbf{0}, \beta^{-1}\mathbf{I}_N + \mathbf{X}\mathbf{A}^{-1}\mathbf{X}^\top) \quad (15.94)$$

$$= \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{C}_{\boldsymbol{\alpha}}) \quad (15.95)$$

where $\mathbf{C}_{\boldsymbol{\alpha}} \triangleq \beta^{-1}\mathbf{I}_N + \mathbf{X}\mathbf{A}^{-1}\mathbf{X}^\top$. 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

$$\ell(\boldsymbol{\alpha}, \beta) = -\frac{1}{2} \log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\alpha}, \beta) = \log |\mathbf{C}_{\boldsymbol{\alpha}}| + \mathbf{y}^\top \mathbf{C}_{\boldsymbol{\alpha}}^{-1} \mathbf{y} \quad (15.97)$$

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.

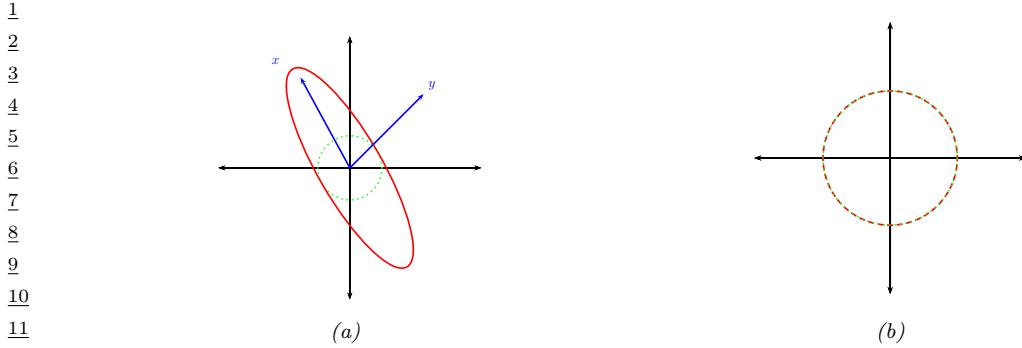


Figure 15.4: Illustration of why ARD results in sparsity. The vector of inputs \mathbf{x} does not point towards the vector of outputs \mathbf{y} , so the feature should be removed. (a) For finite α , the probability density is spread in directions away from \mathbf{y} . (b) When $\alpha = \infty$, the 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 $\hat{\mathbf{w}} = \hat{\beta}\hat{\Sigma}^{-1}\mathbf{X}^T\mathbf{y}$. If we have $\hat{\alpha}_d \approx \infty$, then $\hat{w}_d \approx 0$, 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

$$q(\mathbf{w}, \boldsymbol{\alpha}) = q(\mathbf{w})q(\boldsymbol{\alpha}) \approx \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \prod_d \text{Ga}(\alpha_d | \hat{a}_d, \hat{b}_d) \quad (15.98)$$

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

$$\frac{4}{5} p(\mathbf{w}|\mathcal{D}) = \int \mathcal{N}(\mathbf{w}|\mathbf{0}, \text{diag}(\boldsymbol{\alpha})^{-1}) \prod_d \text{Ga}(\alpha_d | \hat{a}_d, \hat{b}) d\boldsymbol{\alpha} \quad (15.99)$$

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, [HGXW17] 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.

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].

41 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 **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

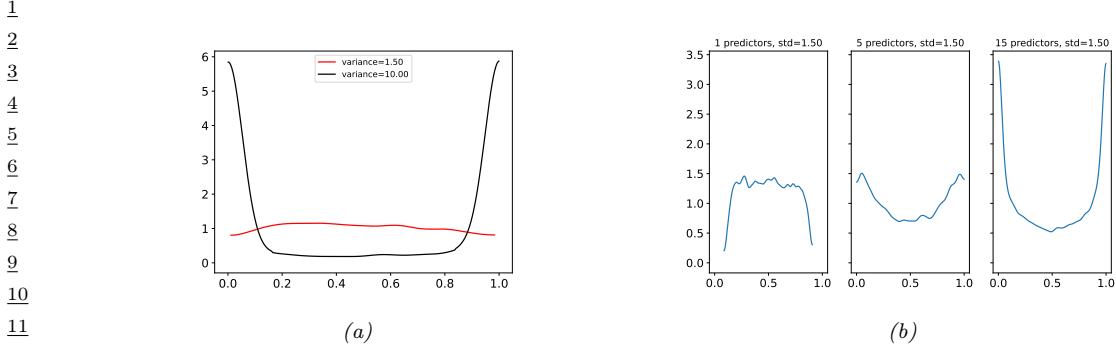


Figure 15.5: (a) Prior on logistic regression output when using $\mathcal{N}(0, \omega)$ prior for the offset term, for $\omega = 10$ or $\omega = 1.5$. Adapted from Figure 11.3 of [McE20]. Generated by `logreg_prior_offset.py`. (b) Distribution over the fraction of 1s we expect to see when using binary 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 15.3.4 Posteriors

⁷ Unfortunately, there is no tractable prior that is conjugate to the logistic likelihood. Hence we cannot
⁸ compute the posterior analytically, unlike with linear regression, even if we use a Gaussian prior.
⁹ (This mirrors the case with MLE, where we have a closed form solution for linear regression, but not
¹⁰ for logistic regression.) Fortunately, there are a range of approximate inference methods we can use,
¹¹ as we discuss in the sections below.
¹²

¹³ 15.3.5 Laplace approximation

¹⁴ As we discuss in Section 7.4.3, the Laplace approximation approximates the posterior using a Gaussian.
¹⁵ The mean of the Gaussian is equal to the MAP estimate $\hat{\mathbf{w}}$, and the covariance is equal to the inverse
¹⁶ 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
¹⁷ a standard optimization method, and we can then compute the Hessian at the mode analytically or
¹⁸ using automatic differentiation.
¹⁹

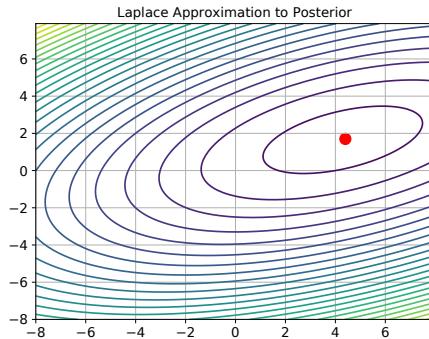
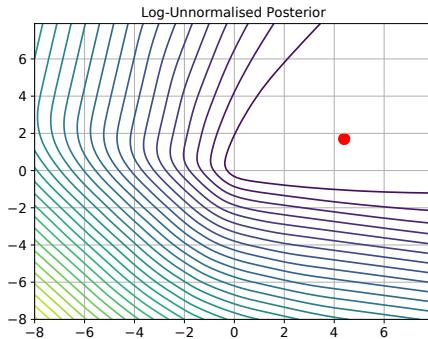
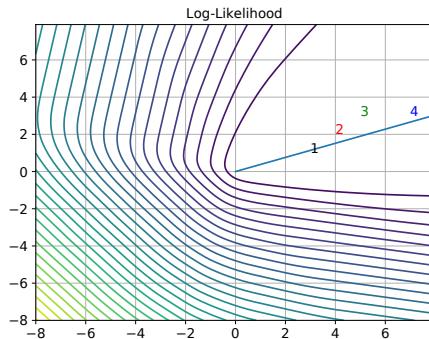
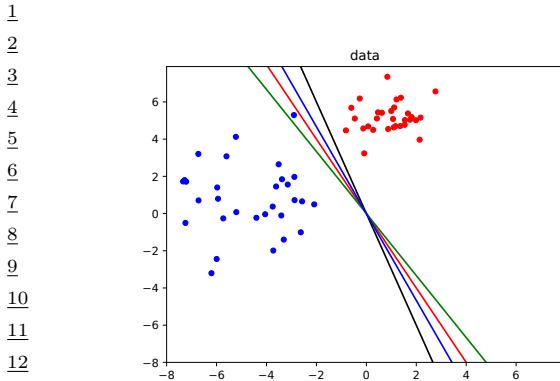
²⁰ As an example, consider the binary data illustrated in Figure 15.6(a). There are many parameter
²¹ settings that correspond to lines that perfectly separate the training data; we show 4 example lines.
²² For each decision boundary in Figure 15.6(a), we plot the corresponding parameter vector as point
²³ in the log likelihood surface in Figure 15.6(b). These parameters values are $\mathbf{w}_1 = (3, 1)$, $\mathbf{w}_2 = (4, 2)$,
²⁴ $\mathbf{w}_3 = (5, 3)$, and $\mathbf{w}_4 = (7, 3)$. These points all approximately satisfy $\mathbf{w}_i(1)/\mathbf{w}_i(2) \approx \hat{\mathbf{w}}_{\text{mle}}(1)/\hat{\mathbf{w}}_{\text{mle}}(2)$,
²⁵ and hence are close to the orientation of the maximum likelihood decision boundary. The points
²⁶ are ordered by increasing weight norm (3.16, 4.47, 5.83, and 7.62). The unconstrained MLE has
²⁷ $\|\mathbf{w}\| = \infty$, so is infinitely far to the top right.
²⁸

²⁹ To ensure a unique solution, we use a (spherical) Gaussian prior centered at the origin, $\mathcal{N}(\mathbf{w}|\mathbf{0}, \sigma^2 \mathbf{I})$.
³⁰ The value of σ^2 controls the strength of the prior. If we set $\sigma^2 = \infty$, we force the MAP estimate
³¹ to be $\mathbf{w} = \mathbf{0}$; this will result in maximally uncertain predictions, since all points \mathbf{x} will produce a
³² predictive distribution of the form $p(y=1|\mathbf{x}) = 0.5$. If we set $\sigma^2 = 0$, the MAP estimate becomes
³³ the MLE, resulting in minimally uncertain predictions. (In particular, all positively labeled points
³⁴ 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
³⁵ data is separable.) As a compromise (to make a nice illustration), we pick the value $\sigma^2 = 100$.
³⁶

³⁷ Multiplying this prior by the likelihood results in the unnormalized posterior shown in Figure 15.6(c).
³⁸ The MAP estimate is shown by the blue dot. The Laplace approximation to this posterior is shown
³⁹ in Figure 15.6(d). We see that it gets the mode correct (by construction), but the shape of the
⁴⁰ posterior is somewhat distorted. (The southwest-northeast orientation captures uncertainty about the
⁴¹ magnitude of \mathbf{w} , and the southeast-northwest orientation captures uncertainty about the orientation
⁴² of the decision boundary.)

⁴³ Next we need to convert the posterior over the parameters into a posterior over predictions, as
⁴⁴ follows:

$$\begin{aligned} p(y|\mathbf{x}, \mathcal{D}) &= \int p(y|\mathbf{x}, \mathbf{w})p(\mathbf{w}|\mathcal{D})d\mathbf{w} \end{aligned} \tag{15.112}$$



29 Figure 15.6: (a) Illustration of the data. (b) Log-likelihood for a logistic regression model. The line is drawn
30 from the origin in the direction of the MLE (which is at infinity). The numbers correspond to 4 points in
31 parameter space, corresponding to the lines in (a). (c) Unnormalised log posterior (assuming vague spherical
32 prior). (d) Laplace approximation to posterior. Adapted from a figure by Mark Girolami. Generated by
33 `logreg_laplace_demo.py`.

38 The simplest way to evaluate this integral is to use a Monte Carlo approximation:

41
42
$$p(y=1|\mathbf{x}, \mathcal{D}) \approx \frac{1}{S} \sum_{s=1}^S \sigma(\mathbf{w}_s^\top \mathbf{x}) \quad (15.113)$$

44 where $\mathbf{w}_s \sim p(\mathbf{w}|\mathcal{D})$.

45 Alternatively, we can use the deterministic **probit approximation** first suggested in [SL90]. If we
46

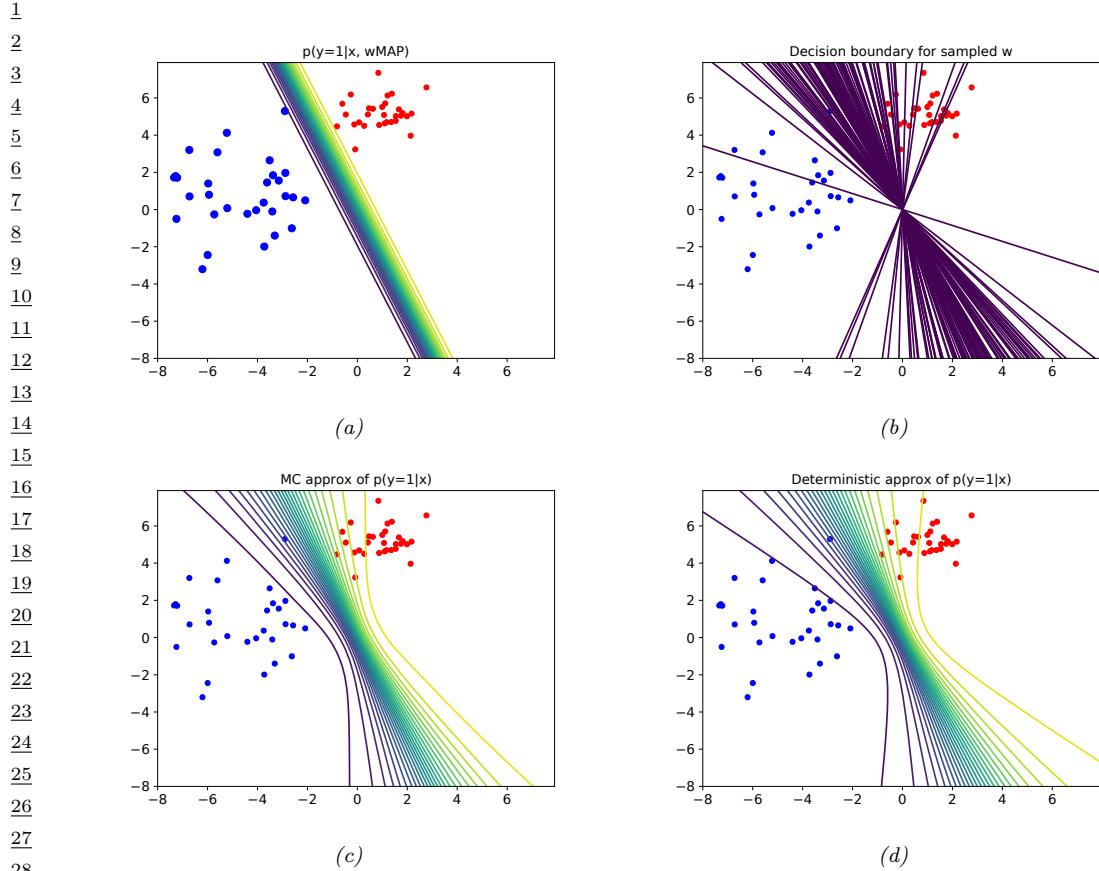


Figure 15.7: Posterior predictive distribution for a logistic regression model in 2d. (a): contours of $p(y = 1|\mathbf{x}, \hat{\mathbf{w}}_{map})$. (b): samples from the posterior predictive distribution. (c): Averaging over these samples. (d): moderated output (probit approximation). Adapted from a figure by Mark Girolami. Generated by [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

$$37 \quad p(y = 1|\mathbf{x}, \mathcal{D}) \approx \sigma(\kappa(v)m) \quad (15.114)$$

38

$$39 \quad \kappa(v) \triangleq (1 + \pi v/8)^{-\frac{1}{2}} \quad (15.115)$$

40

$$41 \quad v = \mathbb{V}[a] = \mathbb{V}[\mathbf{x}^\top \mathbf{w}] = \mathbf{x}^\top \boldsymbol{\Sigma} \mathbf{x} \quad (15.116)$$

42

$$43 \quad m = \mathbb{E}[a] = \mathbf{x}^\top \boldsymbol{\mu} \quad (15.117)$$

44

In Figure 15.7, we show contours of the posterior predictive distribution. Figure 15.7(a) shows the plugin approximation using the MAP estimate. We see that there is no uncertainty about the decision boundary, even though we are generating probabilistic predictions over the labels. Figure 15.7(b) shows what happens when we plug in samples from the Gaussian posterior. Now we see that there is

45

46

47

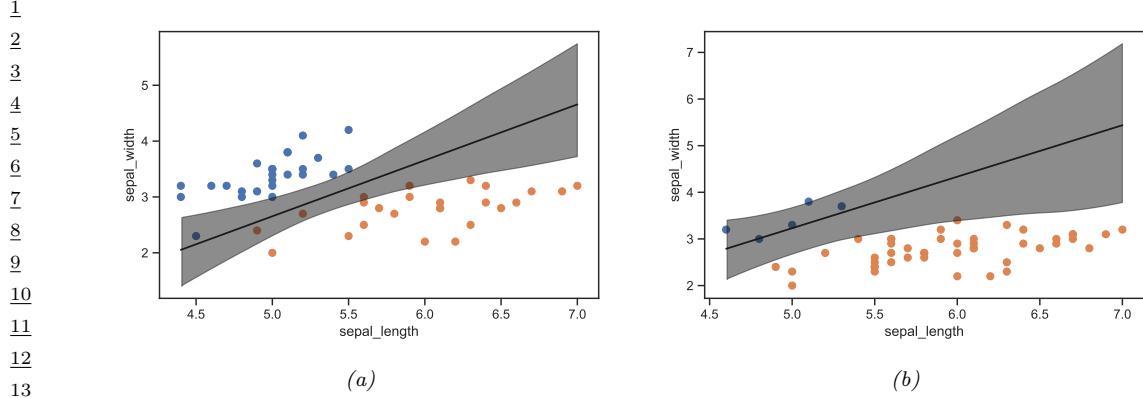


Figure 15.8: Illustration of the posterior over the decision boundary for classifying iris flowers (setosa vs versicolor) using 2 input features. (a) 25 examples per class. Adapted from Figure 4.5 of [Mar18]. (b) 5 examples of class 0, 45 examples of class 1. Adapted from Figure 4.8 of [Mar18]. Generated 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 As we discuss in Section 10.1, variational inference converts approximate inference into an optimization
¹⁵ problem. It does this by choosing an approximate distribution $q(\mathbf{w}; \boldsymbol{\psi})$ and optimising the variational
¹⁶ parameters $\boldsymbol{\psi}$ to maximize the evidence lower bound (ELBO). This has the effect of making
¹⁷ $q(\mathbf{w}; \boldsymbol{\psi}) \approx p(\mathbf{w} | \mathcal{D})$ in the sense that the KL divergence is small. There are several ways to tackle
¹⁸ this: use a stochastic estimate of the ELBO (see Section 10.3.3), use the conditionally conjugate VI
¹⁹ method of Section 10.3.8.2, or use a “local” VI method that creates a quadratic lower bound to the
²⁰ logistic function (see supplementary material).
²¹

²²

²³ 15.4 Probit regression

²⁴ In this section, we discuss **probit regression**, which is similar to binary logistic regression except
²⁵ 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
²⁶ 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
²⁷ inverse of the Gaussian cdf is known as the **probit function**.
²⁸

²⁹ The Gaussian cdf Φ is very similar to the logistic function, as shown in Figure 15.9. Thus probit
³⁰ regression and “regular” logistic regression behave very similarly. However, probit regression has some
³¹ advantages. In particular, it has a simple interpretation as a latent variable model (see Section 15.4.1),
³² which arises from the field of **choice theory** as studied in economics (see e.g., [Koo03]). This also
³³ simplifies the task of Bayesian parameter inference.
³⁴

³⁵

³⁶

³⁷ 15.4.1 Latent variable interpretation

³⁸ We can interpret $a_n = \mathbf{w}^\top \mathbf{x}_n$ as a factor that is proportional to how likely a person is respond
³⁹ positively (generate $y_n = 1$) given input \mathbf{x}_n . However, typically there are other unobserved factors that
⁴⁰ influence someone’s response. Let us model these hidden factors by Gaussian noise, $\epsilon_n \sim \mathcal{N}(0, 1)$. Let
⁴¹ the combined preference for positive outcomes be represented by the latent variable $z_n = \mathbf{w}^\top \mathbf{x}_n + \epsilon_n$.
⁴² We assume that the person will pick the positive label iff this latent factor is positive rather than
⁴³ negative, i.e.,
⁴⁴

⁴⁵

$$\text{y}_n = \mathbb{I}(z_n \geq 0) \tag{15.120}$$

⁴⁶

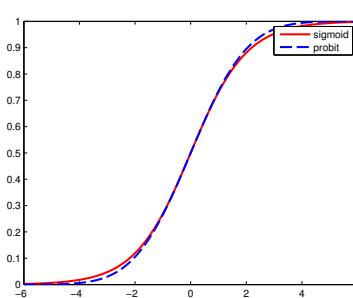


Figure 15.9: The logistic (sigmoid) function $\sigma(x)$ in solid red, with the Gaussian cdf function $\Phi(\lambda x)$ in dotted blue superimposed. Here $\lambda = \sqrt{\pi/8}$, which was chosen so that the derivatives of the two curves match at $x = 0$. Adapted from Figure 4.9 of [Bis06]. Generated 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

$$\mathbf{g}_n \triangleq \frac{d}{d\mathbf{w}} \log p(\tilde{y}_n | \mathbf{w}^\top \mathbf{x}_n) = \frac{d\mu_n}{d\mathbf{w}} \frac{d}{d\mu_n} \log p(\tilde{y}_n | \mathbf{w}^\top \mathbf{x}_n) = \mathbf{x}_n \frac{\tilde{y}_n \phi(\mu_n)}{\Phi(\tilde{y}_n \mu_n)} \quad (15.126)$$

¹ where ϕ is the standard normal pdf, and Φ is its cdf. Similarly, the Hessian for a single case is given
² by
³

$$\underline{4} \quad \underline{5} \quad \underline{6} \quad \mathbf{H}_n = \frac{d}{d\mathbf{w}^2} \log p(\tilde{y}_n | \mathbf{w}^\top \mathbf{x}_n) = -\mathbf{x}_n \left(\frac{\phi(\mu_n)^2}{\Phi(\tilde{y}_n \mu_n)^2} + \frac{\tilde{y}_n \mu_n \phi(\mu_n)}{\Phi(\tilde{y}_n \mu_n)} \right) \mathbf{x}_n^\top \quad (15.127)$$

⁷ This can be passed to any gradient-based optimizer.
⁸

⁹ 15.4.2.2 MLE using EM

¹¹ We can use the latent variable interpretation of probit regression to derive an elegant EM algorithm
¹² for fitting the model. The complete data log likelihood has the following form, assuming a $\mathcal{N}(\mathbf{0}, \mathbf{V}_0)$
¹³ prior on \mathbf{w} :

$$\underline{14} \quad \ell(\mathbf{z}, \mathbf{w} | \mathbf{V}_0) = \log p(\mathbf{y} | \mathbf{z}) + \log \mathcal{N}(\mathbf{z} | \mathbf{X}\mathbf{w}, \mathbf{I}) + \log \mathcal{N}(\mathbf{w} | \mathbf{0}, \mathbf{V}_0) \quad (15.128)$$

$$\underline{15} \quad = \sum_n \log p(y_n | z_n) - \frac{1}{2} (\mathbf{z} - \mathbf{X}\mathbf{w})^\top (\mathbf{z} - \mathbf{X}\mathbf{w}) - \frac{1}{2} \mathbf{w}^\top \mathbf{V}_0^{-1} \mathbf{w} \quad (15.129)$$

¹⁹ The posterior in the E step is a **truncated Gaussian**:
²⁰

$$\underline{21} \quad \underline{22} \quad p(z_n | y_n, \mathbf{x}_n, \mathbf{w}) = \begin{cases} \mathcal{N}(z_n | \mathbf{w}^\top \mathbf{x}_n, 1) \mathbb{I}(z_n > 0) & \text{if } y_n = 1 \\ \mathcal{N}(z_n | \mathbf{w}^\top \mathbf{x}_n, 1) \mathbb{I}(z_n < 0) & \text{if } y_n = 0 \end{cases} \quad (15.130)$$

²⁴ In Equation (15.129), we see that \mathbf{w} only depends linearly on \mathbf{z} , so we just need to compute
²⁵ $\mathbb{E}[z_n | y_n, \mathbf{x}_n, \mathbf{w}]$, so we just need to compute the posterior mean. One can show that this is given by
²⁶

$$\underline{27} \quad \underline{28} \quad \mathbb{E}[z_n | \mathbf{w}, \mathbf{x}_n] = \begin{cases} \mu_n + \frac{\phi(\mu_n)}{1 - \Phi(-\mu_n)} = \mu_n + \frac{\phi(\mu_n)}{\Phi(\mu_n)} & \text{if } y_n = 1 \\ \mu_n - \frac{\phi(\mu_n)}{\Phi(-\mu_n)} = \mu_n - \frac{\phi(\mu_n)}{1 - \Phi(\mu_n)} & \text{if } y_n = 0 \end{cases} \quad (15.131)$$

³⁰ where $\mu_n = \mathbf{w}^\top \mathbf{x}_n$.

³¹ In the M step, we estimate \mathbf{w} using ridge regression, where $\boldsymbol{\mu} = \mathbb{E}[\mathbf{z}]$ is the output we are trying
³² to predict. Specifically, we have
³³

$$\underline{34} \quad \hat{\mathbf{w}} = (\mathbf{V}_0^{-1} + \mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \boldsymbol{\mu} \quad (15.132)$$

³⁶ The EM algorithm is simple, but can be much slower than direct gradient methods, as illustrated
³⁷ in Figure 15.10. This is because the posterior entropy in the E step is quite high, since we only
³⁸ observe that z is positive or negative, but are given no information from the likelihood about its
³⁹ magnitude. Using a stronger regularizer can help speed convergence, because it constrains the range
⁴⁰ of plausible z values. In addition, one can use various speedup tricks, such as data augmentation
⁴¹ [DM01].
⁴²

⁴³ 15.4.3 Bayesian inference

⁴⁵ It is possible to use the latent variable formulation of probit regression in Section 15.4.2.2 to derive a
⁴⁶ simple Gibbs sampling algorithm for approximating the posterior $p(\mathbf{w} | \mathcal{D})$ (see e.g., [AC93; HH06]).
⁴⁷

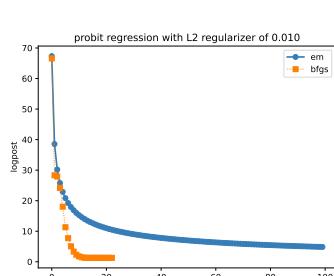


Figure 15.10: Fitting a probit regression model in 2d using a quasi-Newton method or EM. Generated by `probitRegDemo.py`.

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

$$y_n = j \quad \text{if } \gamma_{j-1} < z_n \leq \gamma_j \quad (15.137)$$

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 $\boldsymbol{\gamma}$, 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 15.4.5 Multinomial probit models

⁶ Now consider the case where the response variable can take on C unordered categorical values,
⁷ $y_n \in \{1, \dots, C\}$. The **multinomial probit** model is defined as follows:

$$\begin{aligned} \text{⁸ } z_{nc} &= \mathbf{w}_c^\top \mathbf{x}_{nc} + \epsilon_{nc} & (15.138) \\ \text{⁹ } \epsilon &\sim \mathcal{N}(\mathbf{0}, \mathbf{R}) & (15.139) \\ \text{¹⁰ } y_n &= \arg \max_c z_{nc} & (15.140) \end{aligned}$$

¹¹ See e.g., [DE04; GR06b; Sco09; FSF10] for more details on the model and its connection to multinomial
¹² logistic regression.

¹³ 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
¹⁴ **multivariate probit**, which is one way to model C correlated binary outcomes (see e.g., [TMD12]).

¹⁵ 15.5 Multi-level GLMs

¹⁶ Suppose we have a set of J related datasets, each of which contains a series of N_j datapoints
¹⁷ $\mathcal{D}_j = \{(\mathbf{x}_{nj}, \mathbf{y}_{nj}) : n = 1 : N_j\}$. There are 3 main ways to fit models in such a setting: we could fit J
¹⁸ 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
¹⁹ 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
²⁰ we can use a **hierarchical Bayesian model**, also called a **multilevel model** or **partially pooled**
²¹ **model**, in which we assume each group has its own parameters, $\boldsymbol{\theta}_j$, but that these have something
²² in common, as modeled by a shared global prior $p(\boldsymbol{\phi})$. (Note that each group could be a single
²³ individual.) The overall model has the form

$$\begin{aligned} \text{²⁴ } p(\boldsymbol{\phi}, \boldsymbol{\theta}, \mathcal{D}) &= p(\boldsymbol{\phi}) \prod_{j=1}^J p(\boldsymbol{\theta}_j | \boldsymbol{\phi}) \prod_{n=1}^{N_j} p(\mathbf{y}_{nj} | \mathbf{x}_{nj}, \boldsymbol{\theta}_j) & (15.141) \\ \text{²⁵ } \end{aligned}$$

²⁶ If the likelihood function is a GLM, this is called a **hierarchical generalized linear model** [LN96].

²⁷ 15.5.1 Generalized linear mixed models (GLMMs)

²⁸ If $p(\boldsymbol{\theta}_j | \boldsymbol{\phi}) = \mathcal{N}(\boldsymbol{\theta}_j | \boldsymbol{\phi}, \boldsymbol{\Sigma})$, and the likelihood function is a GLM, then the model can be written as
²⁹ follows:

$$\begin{aligned} \text{³⁰ } p(\mathbf{y}_{nj} | \mathbf{x}_{nj}, \boldsymbol{\theta}) &= p(\mathbf{y}_{nj} | \ell(\boldsymbol{\eta}_{nj})) & (15.142) \\ \text{³¹ } \boldsymbol{\eta}_{nj} &= (\phi_0 + \epsilon_{0j}) + \sum_{d=1}^D (\phi_d + \epsilon_{dj}) x_{njd} & (15.143) \\ \text{³² } \end{aligned}$$

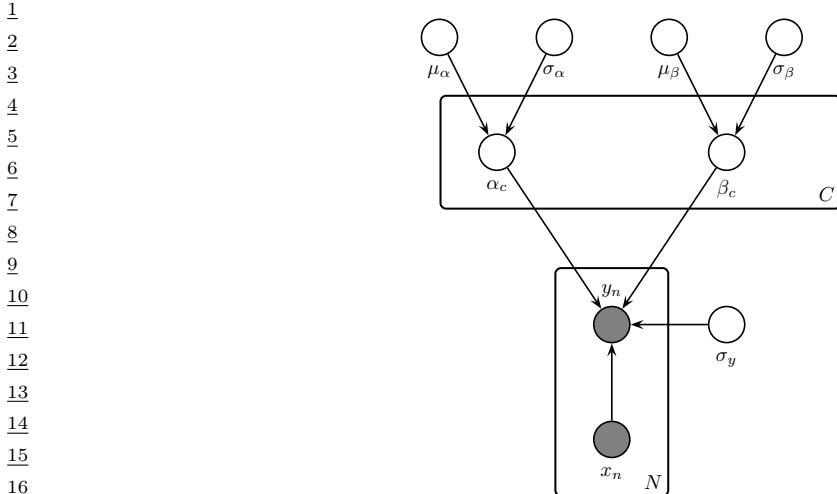


Figure 15.11: A hierarchical Bayesian linear regression model for the radon problem.

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

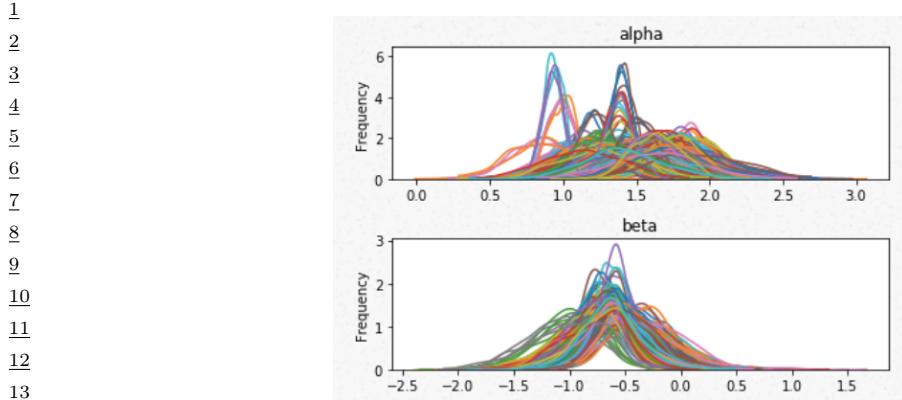
We can fit GLMMs, and hierarchical models more generally, using standard Bayesian inference methods. We can use a variety of algorithms, such as HMC (see e.g., [BG13]), variational Bayes (see e.g., [HOW11; TN13]), expectation propagation (see e.g., [KW18]), etc. In this section, we use HMC, since it 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.



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 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.

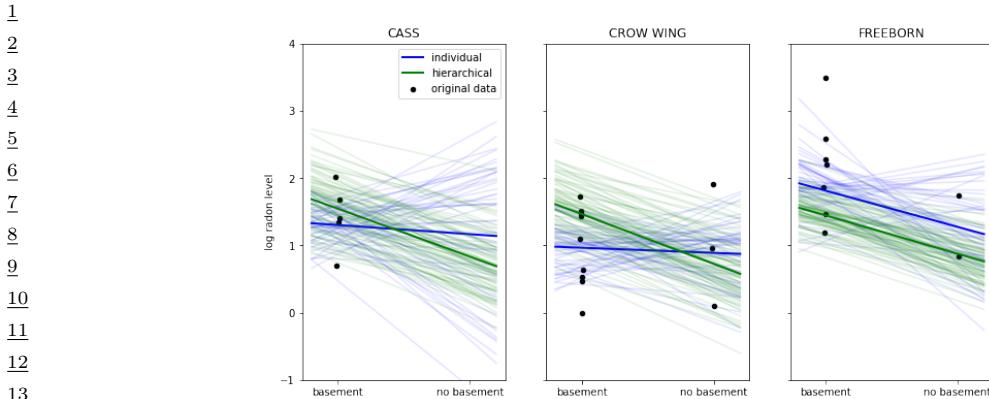


Figure 15.13: Predictions from the radon model for 3 different counties in Minnesota. Black dots are observed datapoints. Green represents results of hierarchical (shared) prior, blue represents results of non-hierarchical prior. Thick lines are the result of using the posterior mean, thin lines are the result of using posterior samples. Generated by `linreg_hierarchical_non_centered_blackjax.ipynb`.

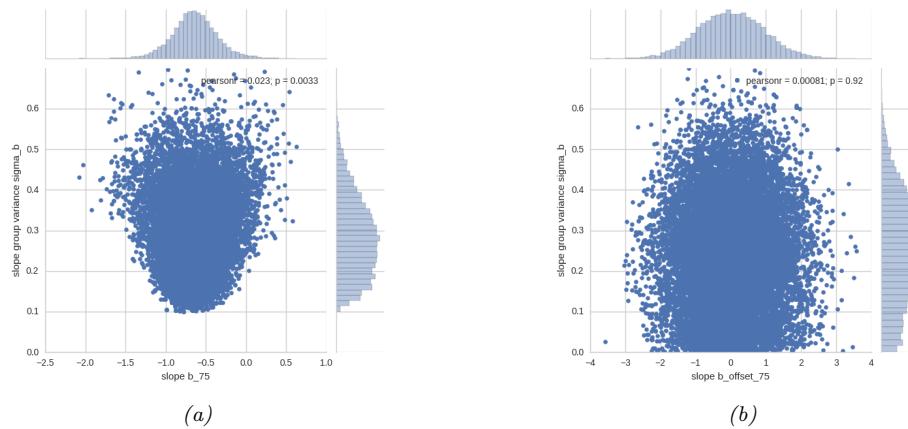


Figure 15.14: (a) Bivariate posterior $p(\beta_c, \sigma_\beta | \mathcal{D})$ for the hierarchical radon model for county $c = 75$ using centered parameterization. (b) Similar to (a) except we plot $p(\hat{\beta}_c, \sigma_\beta | \mathcal{D})$ for the non-centered parameterization. Generated by `linreg_hierarchical_non_centered_blackjax.ipynb`.

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 A simple solution to this is to use a **non-centered parameterization** [PR03]. That is, we replace
7 $\beta_c \sim \mathcal{N}(\mu_\beta, \sigma_\beta^2)$ 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,
8 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
9 for more details.
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

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.

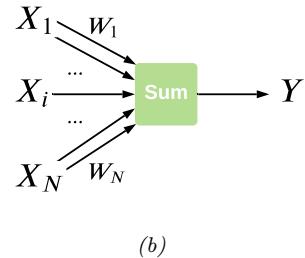
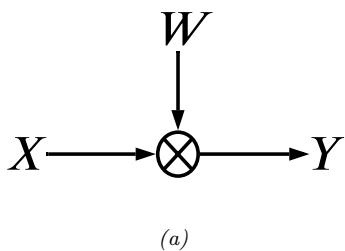
The other way we can combine DNNs and probabilistic models is to use DNNs to approximate the posterior distribution, i.e., we learn a function f to compute $q(\mathbf{h}|f(\mathcal{D}; \boldsymbol{\phi}))$, where \mathbf{h} are the hidden variables (latents and/or parameters), \mathcal{D} are the observed 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} .

1
2
3
4
5
6
7
8
9
10



11 *Figure 16.1: An artiticial “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

$$y = f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{w}^\top \mathbf{x} + b \quad (16.1)$$

22

23 where $\boldsymbol{\theta} = (\mathbf{w}, b)$ are the parameters for the function f . This is depicted in Figure 16.1. (The bias
24 term is omitted for clarity.)

25

26 It is common to group a set of neurons together into a **layer**. We can then represent the activations
27 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
28 an output vector \mathbf{y} by multiplying by a weight matrix \mathbf{W} , an adding an offset vector or bias term \mathbf{b}
29 to get

30

$$\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{W}\mathbf{x} + \mathbf{b} \quad (16.2)$$

31

32 where $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b})$ are the parameters for the function f . This is called a **linear layer**, or **fully
33 connected layer**.

34

35 It is common to prepend the bias vector onto the first column of the weight matrix, and to append
36 a 1 to the vecotr \mathbf{x} , so that we can write this more compactly as $\tilde{\mathbf{x}} = \tilde{\mathbf{W}}^\top \mathbf{x}$, where $\tilde{\mathbf{W}} = [\mathbf{W}, \mathbf{b}]$ and
37 $\tilde{\mathbf{x}} = [\mathbf{x}, 1]$. This allows us to ignore the bias term from our notation if we want to.

38

16.2.2 Non-linearities

39

40 A stack of linear layers is equivalent to a single linear layer where we multilpy together all the
41 weight matrices. To get more expressive power we can transform each layer by passing it elementwise
42 (pointwise) through a nonlinear function called an **activation function**. This is denoted by

43

$$\mathbf{y} = \varphi(\mathbf{x}) = [\varphi(x_1), \dots, \varphi(x_D)] \quad (16.3)$$

44

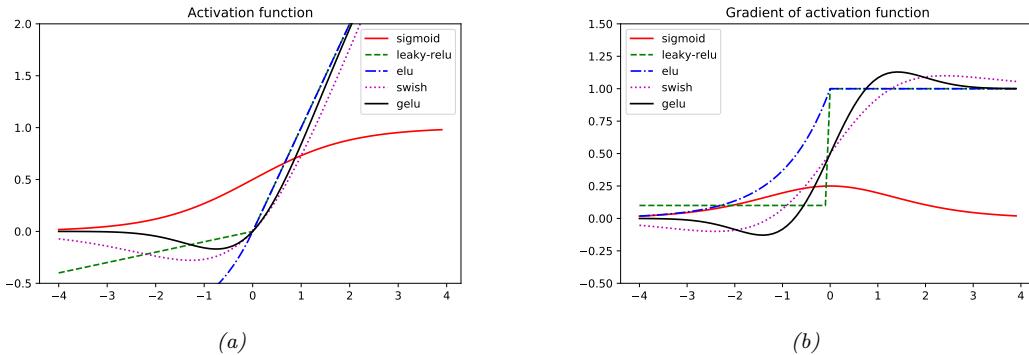
45 See Table 16.1 for a list of some common activation functions, and Figure 16.2 for a visualization.

46 For more details, see e.g., [Mur22, Sec 13.2.3].

47

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. Note that, technically speaking, we are using **cross correlation** rather than convolution. However, these terms are used interchangeably in deep learning.

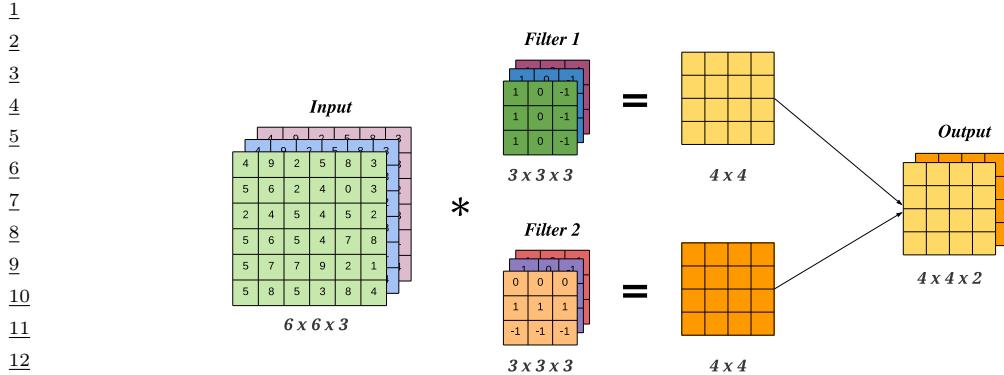


Figure 16.3: A 2d convolutional layer with 3 input channels and 2 output channels. The kernel has size 3×3 and we use stride 1 with 0 padding, so the the 6×6 input gets mapped to the 4×4 output.

*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]).

39

40 16.2.4 Residual (skip) connections

41

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,

47

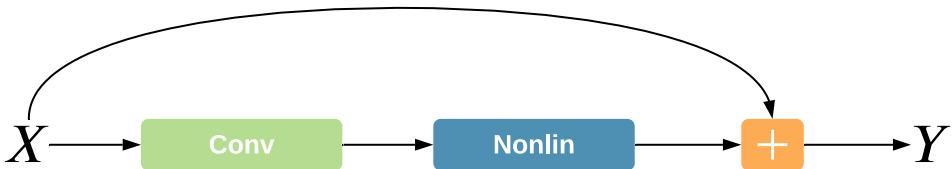


Figure 16.4: A residual connection around a convolutional layer.

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.

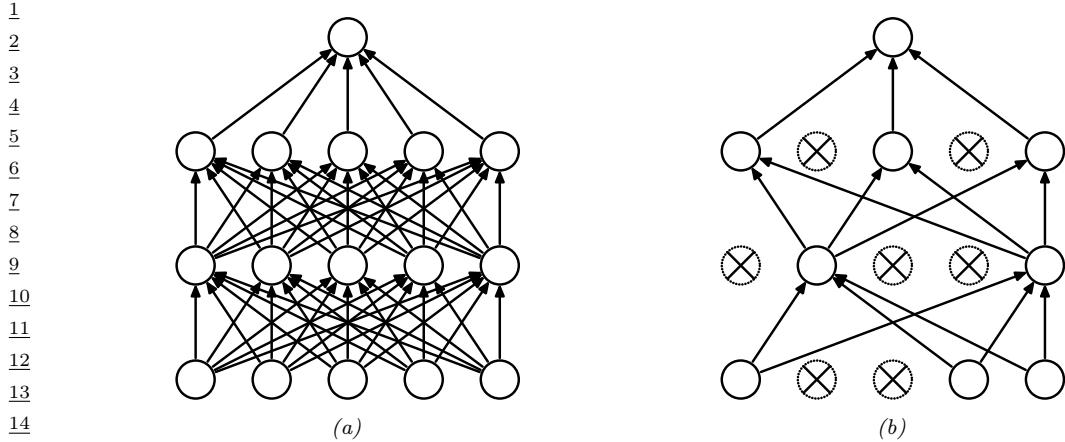


Figure 16.5: Illustration of dropout. (a) A standard neural net with 2 hidden layers. (b) An example of a thinned net produced by applying dropout with $p = 0.5$. Units that have been dropped out are marked with an x . From Figure 1 of [Sri+14]. Used with kind permission of Geoff Hinton.

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

$$36 \quad \hat{\mathbf{y}} = \sum_{i=1}^m \alpha_i \mathbf{y}_i \quad (16.7)$$

39

40 See Section 18.3.7 for details.

41 We can make a differentiable and parametric version of this as follows (see [Tsa+19] for details).

42 First we replace the stored examples matrix \mathbf{X} with a learned embedding, to create a set of stored
43 **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
44 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
45 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.

46 To ensure the output is a differentiable function of the input, we replace the fixed kernel function
47

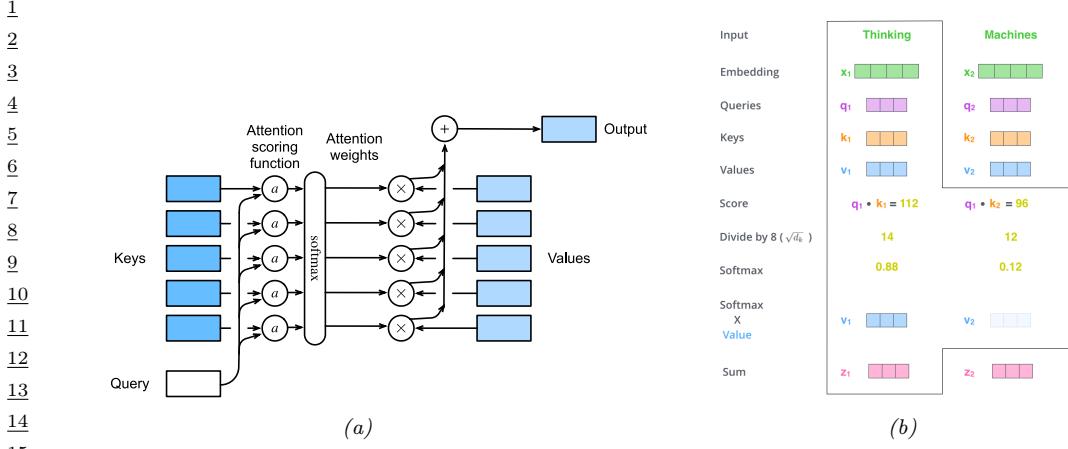


Figure 16.6: Attention layer. (a) Mapping a single query q to a single output, given a set of keys and values. From Figure 10.3.1 of [Zha+20a]. Used with kind permission of Aston Zhang. (b) Detailed visualization of attention. Here the $n = 2$ queries q_j are derived by embedding input vectors x_j corresponding 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 Alamar.

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

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

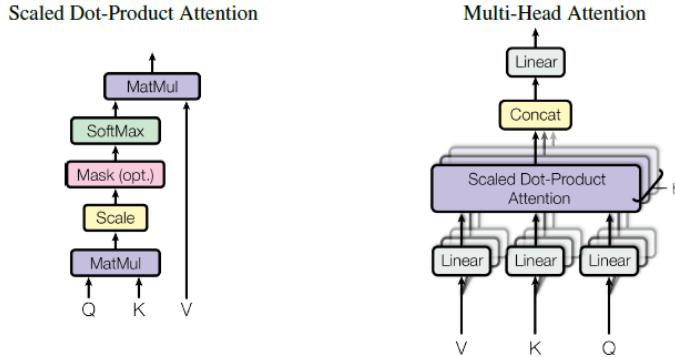


Figure 16.7: (a) Scaled dot-product attention in matrix form. (b) Multi-head attention. From Figure 2 of [Vas+17b]. Used with kind permission of Ashish Vaswani.

can implement this efficiently by setting the attention score for the masked entries to a large negative number, such as -10^6 , so that the corresponding softmax weights will be 0.

In practice, we usually deal with minibatches of n vectors at a time. Let the corresponding matrices 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

$$\mathbf{z}_j = \sum_{i=1}^m \alpha_i(\mathbf{q}_j, \mathbf{K}) \mathbf{v}_i \quad (16.11)$$

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 parallel using

$$\mathbf{Z} = \text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathcal{S}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V} \quad (16.12)$$

where the softmax function \mathcal{S} is applied row-wise. See Figure 16.7(left) for an illustration, and Figure 16.6b for a detailed worked example.

To increase the flexibility of the model, we often use a **multi-head attention** layer, as illustrated in Figure 16.7(right). Let the i 'th head be

$$\mathbf{h}_i = \text{Attn}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \quad (16.13)$$

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 output of the MHA layer to be

$$\mathbf{o} = \text{MHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\mathbf{h}_1, \dots, \mathbf{h}_h)\mathbf{W}^O \quad (16.14)$$

where h is the number of heads, and $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d}$. Having multiple heads can increase performance of the layer, in the event that some of the weight matrices are poorly initialized; after training, we can often remove all but one of the heads [MLN19].

When the output of one attention layer is used as input to another, the method is called **self-attention**. This is the basis of the transformer model, which we discuss in Section 16.3.4.

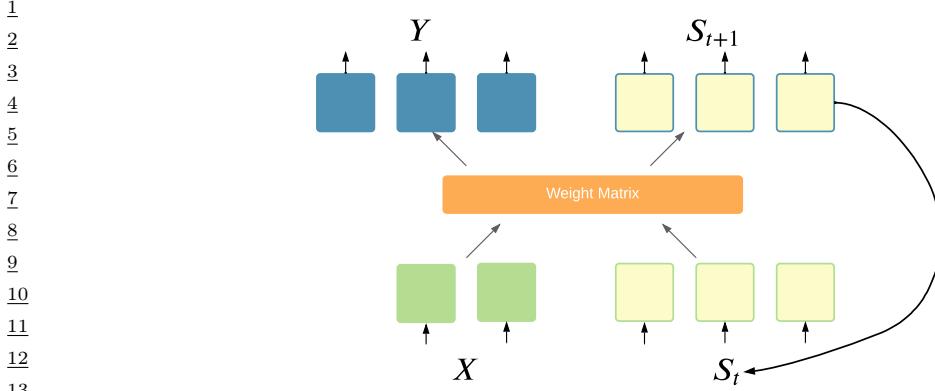


Figure 16.8: Recurrent layer.

16.2.8 Recurrent layers

We can make the model be **stateful** by augmenting the input \mathbf{x} with the current state \mathbf{s}_t , and then computing the output and the new state using some kind of function:

$$(\mathbf{y}, \mathbf{s}_{t+1}) = f(\mathbf{x}, \mathbf{s}_t) \quad (16.15)$$

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

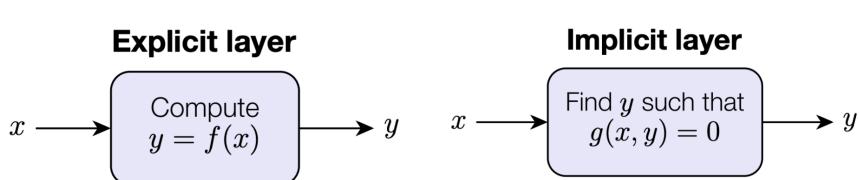


Figure 16.9: Explicit vs implicit layers.

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 $\mathbf{W}' = \boldsymbol{\sigma}(\mathbf{z}^\top \mathbf{W} + \mathbf{V}) = \text{diag}(a_1, \dots, a_n)$, 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.

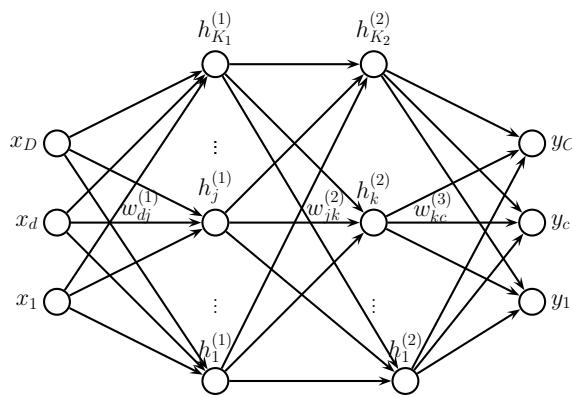


Figure 16.10: A feedforward neural network with D inputs, K_1 hidden units in layer 1, K_2 hidden units in layer 2, and C outputs. $w_{jk}^{(l)}$ is the weight of the connection from node j in layer $l - 1$ to node k in layer l .

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:

$$f(\mathbf{x}; \theta) = \mathbf{W}_L \varphi_L (\mathbf{W}_{L-1} \varphi_{L-1} (\cdots \varphi_1 (\mathbf{W}_1 \mathbf{x}) \cdots)) \quad (16.20)$$

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

$$h_k^{(l)} = \varphi_l \left(b_k^{(l)} + \sum_{j=1}^{K_{l-1}} w_{jk}^{(l)} h_j^{(l-1)} \right) \quad (16.21)$$

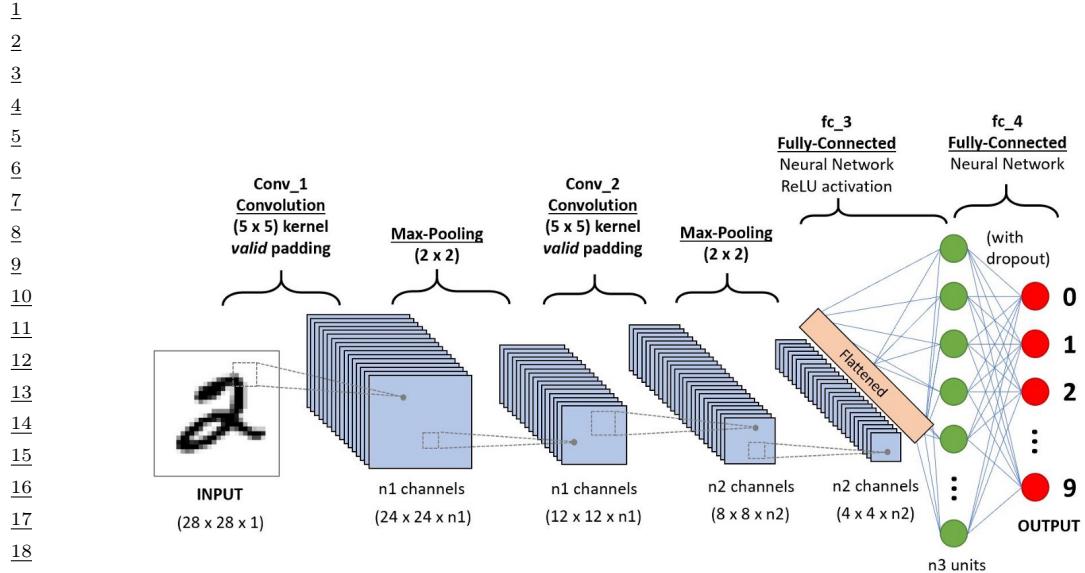
where φ_l is the nonlinear activation function at layer l . For a classification problem, the final nonlinearity is usually the softmax function.

16.3.2 Convolutional neural networks (CNN)

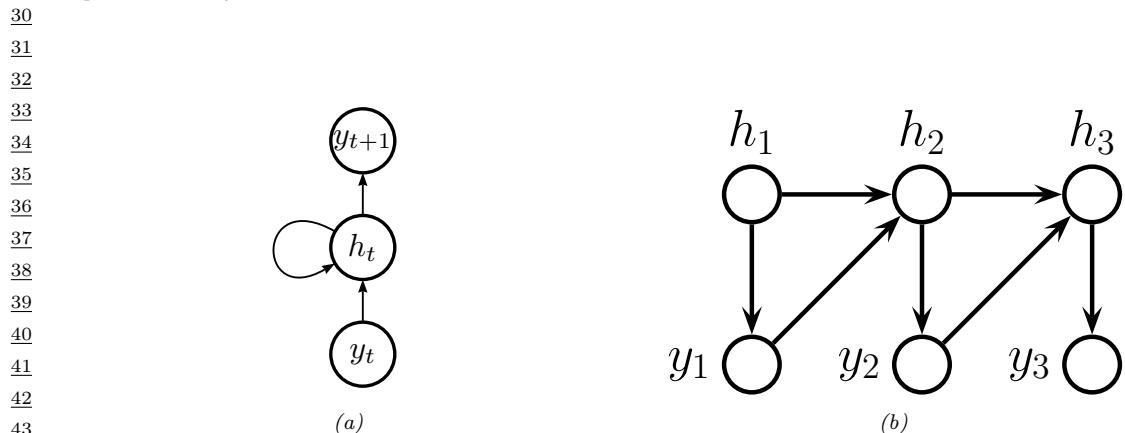
A vanilla **convolutional neural network** or **CNN** consists of a series of convolutional layers, pooling layers, linear layers, and nonlinearities. See Figure 16.11 for an example. More sophisticated architectures, such as the **ResNet** model [He+16a; He+16b], add skip (residual) connections, normalization 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



22 Figure 16.11: A simple CNN for classifying MNIST images. The model has 2 convolutional layers, and 2
 23 fully connected layers, with a softmax output. The dotted square boxes denote the receptive fields. The use of
 24 a 5×5 filter with “valid” convolution means we reduce the input size from 28 pixels per side to $28 - 5 + 1 = 24$.
 25 The use of 2×2 max-pooling with stride 2 reduces the 24 pixels to 12. The second convolution reduces this
 26 to $12 - 5 + 1 = 8$ pixels per side, and the second pooling layer reduces this to 4. As we reduce the spatial
 27 size of each layer, we typically increase the number of feature channels (e.g., $n_2 = 2n_1$). The first fully
 28 connected (linear) layer (fc_3) maps from n_2 features to n_3 . The second fully connected layer (fc_4) maps
 29 from n_3 features to $C = 10$ output logits; this final layer may optionally be regularized with dropout. From
<https://bit.ly/2YB9oOH>. Used with kind permission of Sumit Saha.



44 Figure 16.12: Illustration of a recurrent neural network (RNN). (a) With self-loop. (b) Unrolled in time.
 45

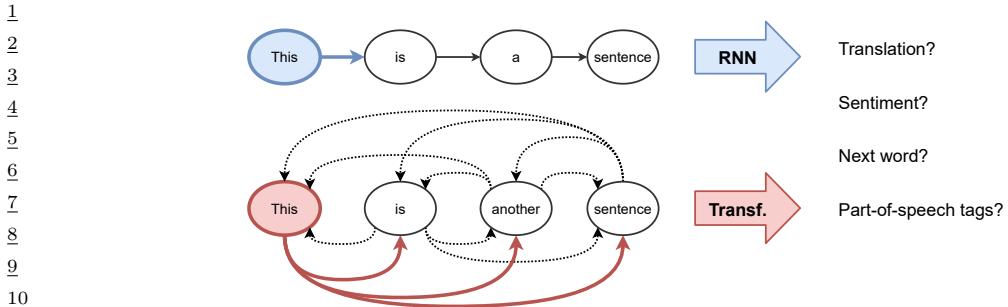


Figure 16.13: Visualizing the difference between an RNN and a transformer. From [JOS20]. Used with kind permission of Chaitanya Joshi.

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.

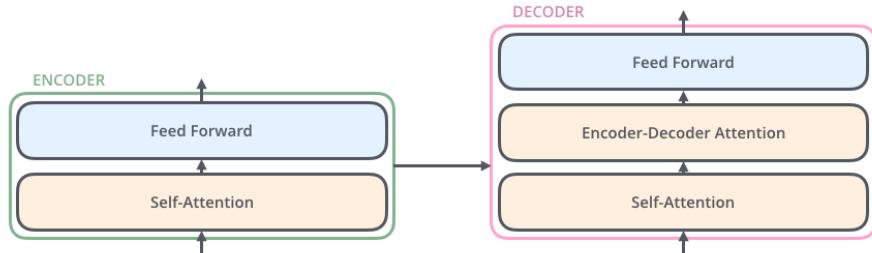
It is also possible to create a decoder-only transformer, in which each output \mathbf{y}_t only attends to all the previously generated outputs, $\mathbf{y}_{1:t-1}$. This can be implemented using masked attention, and is useful for generative language models, such as GPT (see Section 23.4.1).

We can combine the encoder and decoder to create a conditional sequence-to-sequence model,

1
2
3
4
5
6
7
8
9
10

11 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.
12
13

14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29

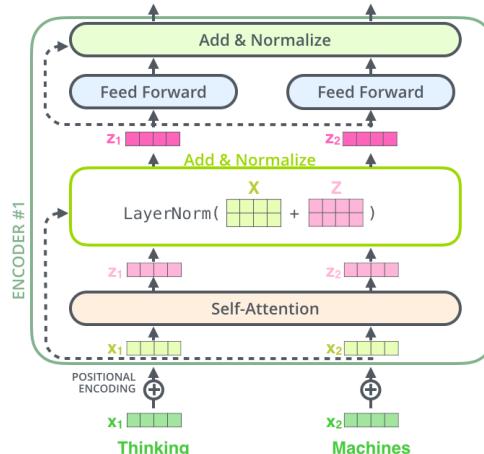


30 Figure 16.15: The encoder block of a transformer for two input tokens. From <https://jalammar.github.io/illustrated-transformer/>. Used with kind permission of Jay Alammar.
31
32
33

34 $p(\mathbf{y}_{1:T_y} | \mathbf{x}_{1:T_x})$, as proposed in the original transformer paper [Vas+17c]. The high level structure is
35 shown in Figure 16.14. We give the details below.
36

37 16.3.4.1 Encoder

38
39
40
41
42
43
44
45
46
47



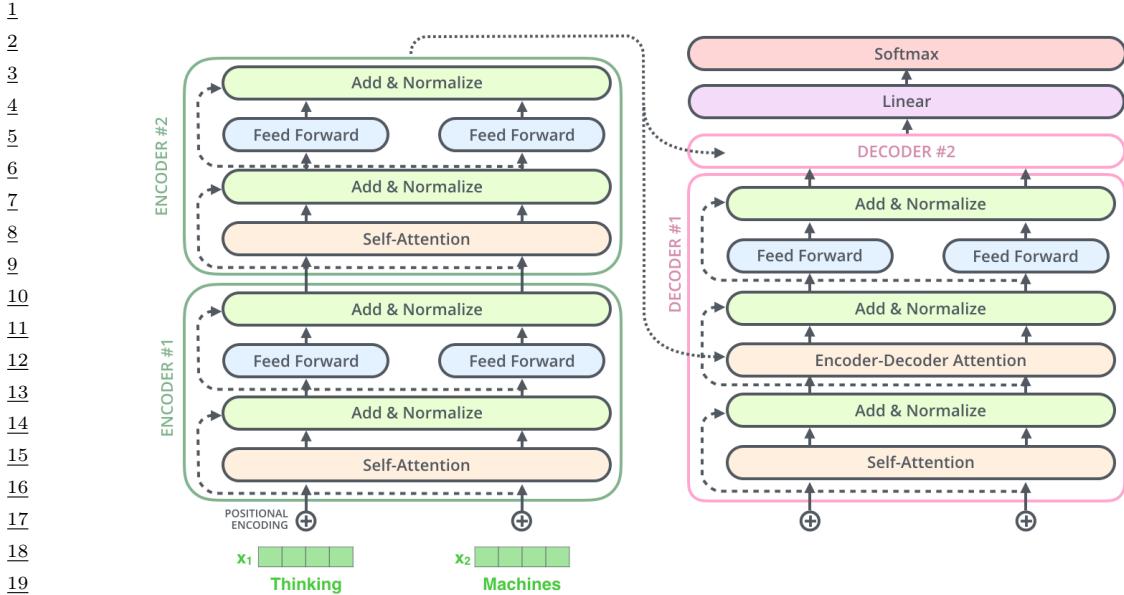


Figure 16.16: A transformer model where we use 2 encoder blocks and 2 decoder blocks. (The second decoder block is not expanded.) We assume there are 2 input and 2 output tokens. From <https://github.com/jalammar/illustrated-transformer/>. Used with kind permission of Jay Alammar.

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

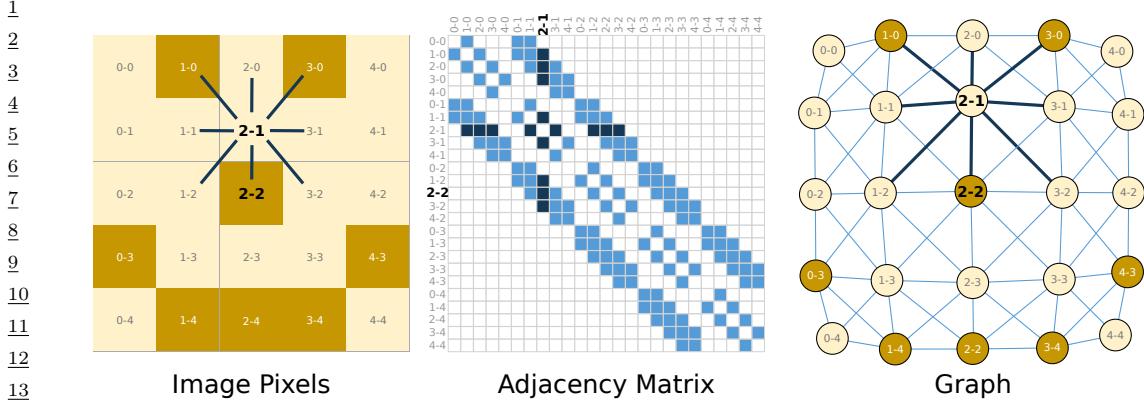


Figure 16.17: Left: Illustration of a 5×5 image, where each pixel is either off (light yellow) or on (dark yellow). Each non-border pixel has 8 nearest neighbors. We highlight the node at location $(2,1)$, where the top-left is $(0,0)$. Middle: The corresponding adjacency matrix, which is sparse and 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.

20

21

22

23 **positional encoding** vectors to the input tokens $\mathbf{x} \in \mathbb{R}^{T_x \times D}$. That is, we replace \mathbf{x} with $\mathbf{x} + \mathbf{u}$,
 24 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 \mathbf{x}_i comes
 25 from the i 'th location in the N -dimensional sequence.

26

27 16.3.4.4 Discussion

28

29 It has been found that large transformers are very flexible sequence-to-sequence function approxima-
 30 tors, if trained on enough data (see e.g., [Lin+21] for a review in the context of NLP, and [Kha+21;
 31 Han+20; Zan21] for reviews in the context of computer vision). The reasons why they work so well
 32 are still not very clear. However, some initial analysis can be found in e.g., [WGY21; Nel21; BP21].
 33 See also Section 16.3.5.5 where we discuss the connection with graph neural networks.

34

35 16.3.5 Graph neural networks (GNNs)

36

37 In this section, we discuss **graph neural networks** or **GNNs**. Our presentation is based on [SL+21],
 38 which in turn is a summary of the **message passing neural network** framework of [Gil+17] and
 39 the **Graph Nets** framework of [Bat+18].

40 We assume the graph is represent as a set of N nodes or vertices, each associated with a feature
 41 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
 42 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
 43 of the graph, such as its size. (We can think of \mathbf{u} as the features associated with a global or master
 44 node.) The topology of the graph can be represented as an $N \times N$ **adjacency matrix**, but since
 45 this is usually very sparse (see Figure 16.17 for an example), a more compact representation is just
 46 to store the list of edges in an **adjacency list** (see Figure 16.18 for an example).

47

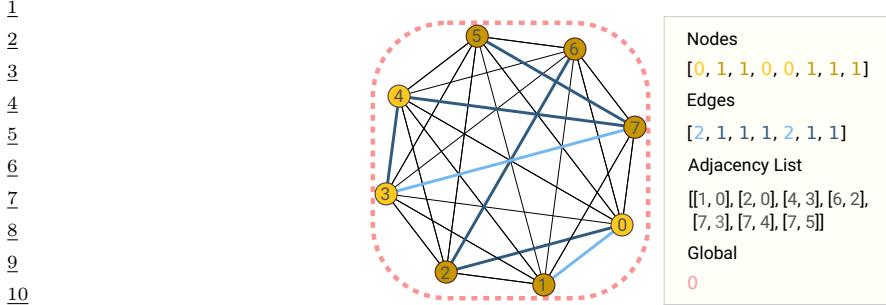


Figure 16.18: A simple graph where each node has 2 types (0=light yellow, 1=dark yellow), each edge has 2 types (1=gray, 2=blue), and the global feature vector is a constant (0=red). We represent the topology using an adjacency list. From [SL+21]. Used with kind permission of Benjamin Sanchez-Lengeling.

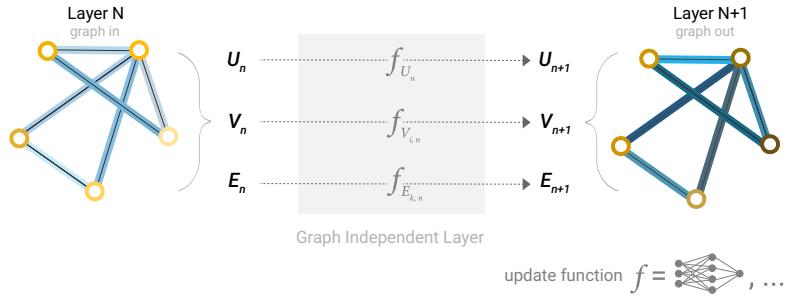


Figure 16.19: A basic GNN layer. We update the embedding vectors \mathbf{U} , \mathbf{V} and \mathbf{V} using the global, node and edge functions f . From [SL+21]. Used with kind permission of Benjamin Sanchez-Lengeling.

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.

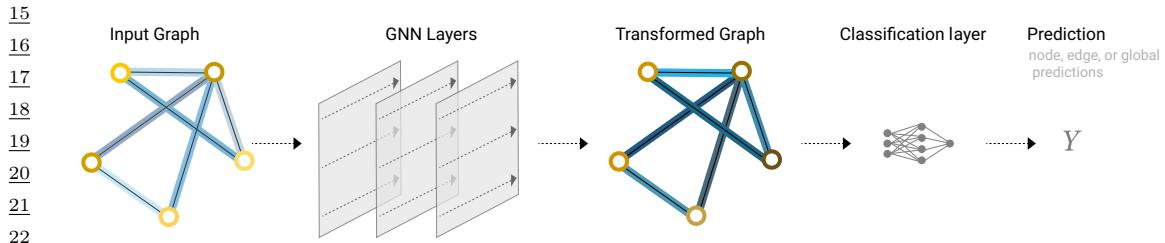


Figure 16.21: An end-to-end GNN classifier. From [SL+21]. Used with kind permission of Benjamin Sanchez-Lengeling.

25
26
27

28 16.3.5.2 Message passing

29
30 Instead of transforming each vector independently and then pooling, we can first pool the information
31 for each node (or edge) and then update its vector representation. That is, for node i , we **gather**
32 information from all neighboring nodes, $\{\mathbf{h}_j : j \in \text{nbr}(i)\}$; we **aggregate** these vectors with the
33 local vector using an operation such as sum; and then we compute the new state using an **update**
34 function, such as

$$36 \quad \mathbf{h}'_i = \text{ReLU}(\mathbf{U}\mathbf{h}_i + \sum_{j \in \text{nbr}(i)} \mathbf{V}\mathbf{h}_j) \quad (16.23)$$

38 See Figure 16.24a for a visualization.

40 The above operation can be viewed as a form of “**message passing**”, in which the values of
41 neighboring nodes \mathbf{h}_j are sent to node i and then combined. It is more general than belief propaga-
42 tion (Section 9.2), since the messages are not restricted to represent probability distributions (see
43 Section 9.3.7 for more discussion).

44 After K message passing layers, each node will have received information from neighbors which
45 are K steps away in the graph. This can be “short circuited” by sending messages through the global
46 node, which acts as a kind of bottleneck. See Figure 16.22 for an illustration.

47

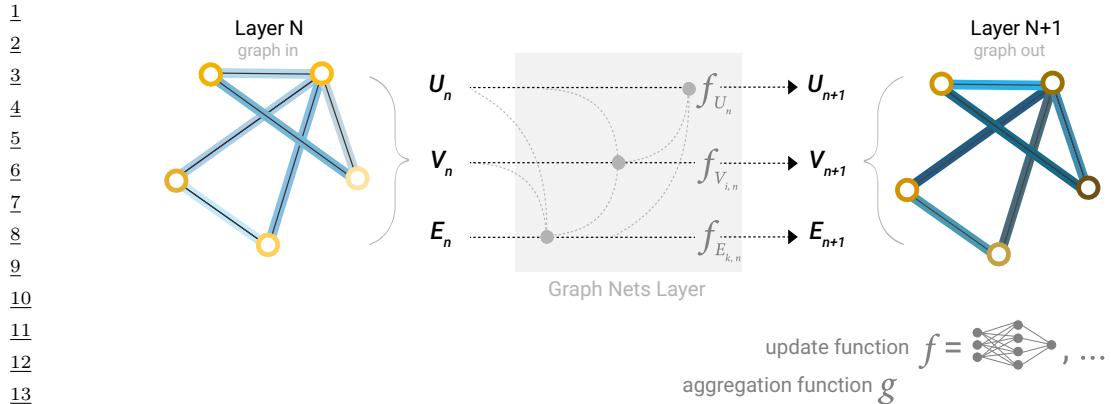


Figure 16.22: Message passing in one layer of a GNN. First the global node U_n and the local nodes V_n send messages to the edges E_n , which get updated to give E_{n+1} . Then the nodes get updated to give V_{n+1} . Finally the global node gets updated to give U_{n+1} . From [SL+21]. Used with kind permission of Benjamin Sanchez-Lengeling.

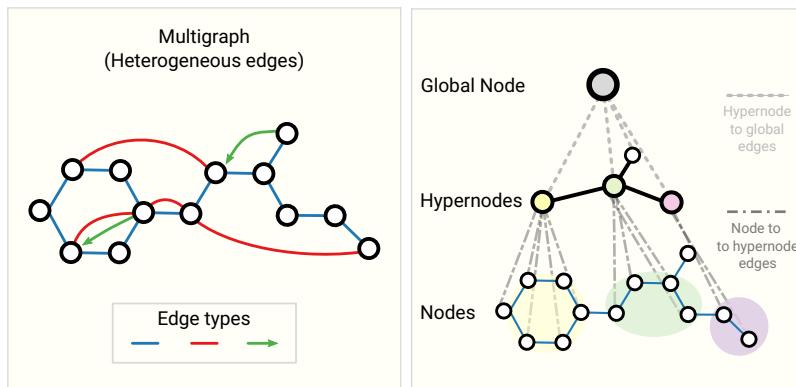


Figure 16.23: Left: a multigraph can have different edge types. Right: a hypergraph can have edges which connect multiple nodes. From [SL+21]. Used with kind permission of Benjamin Sanchez-Lengeling.

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).

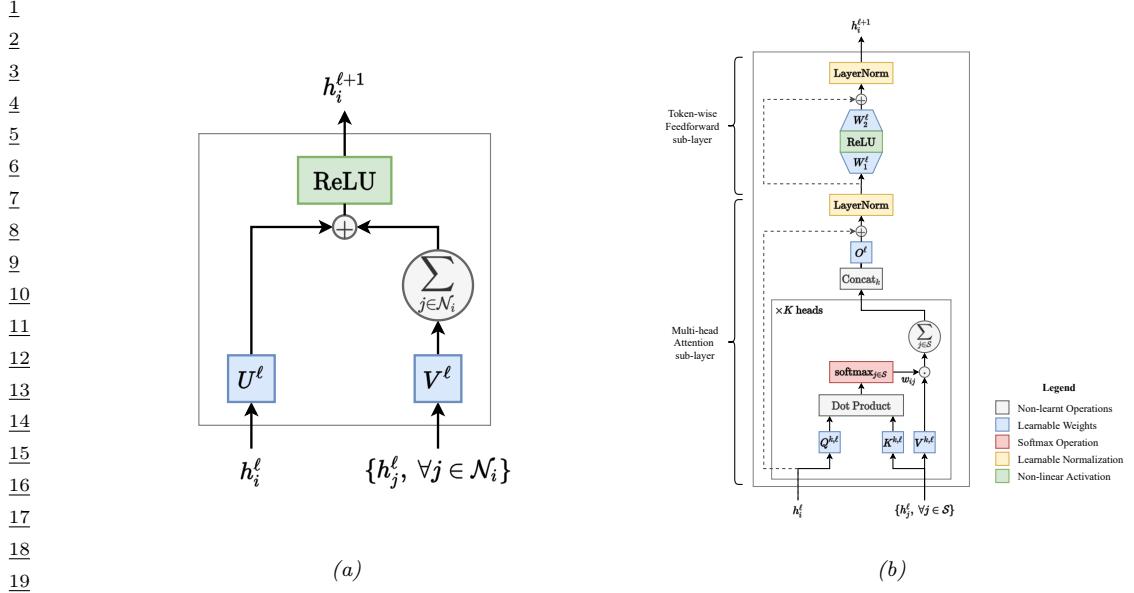


Figure 16.24: (a) A graph neural network aggregation block. Here h_i^ℓ is the hidden representation for node i in layer ℓ , and $\mathcal{N}(i)$ are i 's neighbors. The output is given by $h_i^{\ell+1} = \text{ReLU}(\mathbf{U}^\ell h_i + \sum_{j \in \text{nbr}(i)} \mathbf{V}^\ell h_j^\ell)$. (b) A transformer 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 $h_i^{\ell+1} = \text{Attn}(\mathbf{Q}^\ell h_i^\ell, \{\mathbf{K}^\ell h_j^\ell, \mathbf{V}^\ell h_j^\ell\})$. From [Jos20]. Used with kind permission of Chaitanya Joshi.

25

26

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.

33

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.

47

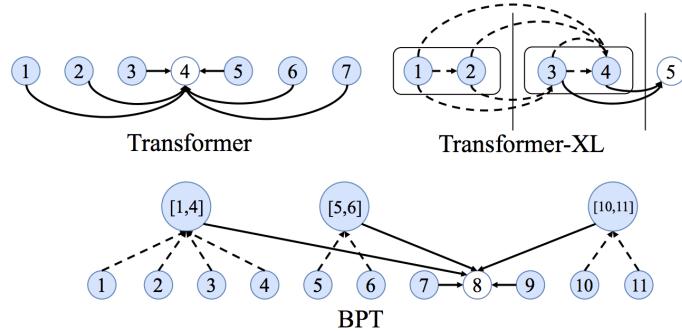


Figure 16.25: Graph connectivity for different types of transformer. Top left: in a vanilla Transformer, every node is connected to every other node. Top right: in Transformer-XL, nodes are grouped into blocks. Bottom: in BPT, we use a binary partitioning of the graph to create virtual node clusters. From <https://graphdeeplearning.github.io/post/transformers-are-gnns/>. Used with kind permission of Chaitanya Joshi.

