

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 125  
5 Information theory 185  
6 Optimization 221

## II Inference 301

7 Inference algorithms: an overview 303  
8 State-space inference 315  
9 Message passing inference 369  
10 Variational inference 401  
11 Monte Carlo inference 451  
12 Markov Chain Monte Carlo inference 467  
13 Sequential Monte Carlo inference 517

## III Prediction 553

14 Predictive models: an overview 555  
15 Generalized linear models 571  
16 Deep neural networks 603  
17 Bayesian neural networks 625  
18 Gaussian processes 659  
19 Structured prediction 717  
20 Beyond the iid assumption 739

<u>1</u>	<b>IV Generation</b>	<b>775</b>
<u>2</u>	21 Generative models: an overview	777
<u>3</u>	22 Variational autoencoders	791
<u>4</u>	23 Auto-regressive models	835
<u>5</u>	24 Normalizing Flows	843
<u>6</u>	25 Energy-based models	863
<u>7</u>	26 Denoising diffusion models	883
<u>8</u>	27 Generative adversarial networks	891
<u>9</u>		
<u>10</u>		
<u>11</u>		
<u>12</u>		
<u>13</u>	<b>V Discovery</b>	<b>923</b>
<u>14</u>	28 Discovery methods: an overview	925
<u>15</u>	29 Latent variable models	927
<u>16</u>	30 Hidden Markov models	967
<u>17</u>	31 State-space models	997
<u>18</u>	32 Graph learning	1023
<u>19</u>	33 Non-parametric Bayesian models	1033
<u>20</u>	34 Representation learning ( <b>Unfinished</b> )	1065
<u>21</u>	35 Interpretability	1067
<u>22</u>		
<u>23</u>		
<u>24</u>		
<u>25</u>	<b>VI Decision making</b>	<b>1101</b>
<u>26</u>	36 Multi-step decision problems	1103
<u>27</u>	37 Reinforcement learning	1129
<u>28</u>	38 Causality	1167
<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

<b>Preface</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
<b>I Fundamentals</b>	<b>3</b>
<b>2 Probability</b>	<b>5</b>
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	43
11	2.7.1	Invertible transformations (bijections)	44
12	2.7.2	Monte Carlo approximation	44
13	2.7.3	Probability integral transform	44
14	2.8	Markov chains	46
15	2.8.1	Parameterization	46
16	2.8.2	Application: Language modeling	48
17	2.8.3	Parameter estimation	49
18	2.8.4	Stationary distribution of a Markov chain	51
19	2.9	Divergence measures between probability distributions	54
20	2.9.1	f-divergence	55
21	2.9.2	Integral probability metrics	56
22	2.9.3	Maximum mean discrepancy (MMD)	57
23	2.9.4	Total variation distance	60
24	2.9.5	Comparing distributions using binary classifiers	60
25	<b>3 Statistics</b>	<b>63</b>	
26	3.1	Introduction	63
27	3.1.1	Frequentist statistics	63
28	3.1.2	Bayesian statistics	63
29	3.1.3	Arguments for the Bayesian approach	64
30	3.1.4	Arguments against the Bayesian approach	65
31	3.1.5	Why not just use MAP estimation?	65
32	3.2	Closed-form analysis using conjugate priors	70
33	3.2.1	The binomial model	70
34	3.2.2	The multinomial model	78
35	3.2.3	The univariate Gaussian model	80
36	3.2.4	The multivariate Gaussian model	85
37	3.2.5	Conjugate-exponential models	90
38	3.3	Beyond conjugate priors	93
39	3.3.1	Robust (heavy-tailed) priors	93
40	3.3.2	Priors for variance parameters	93
41	3.4	Noninformative priors	95
42	3.4.1	Maximum entropy priors	95
43	3.4.2	Jeffreys priors	96
44	3.4.3	Invariant priors	99

1	3.4.4	Reference priors	100
2	3.5	Hierarchical priors	100
3	3.5.1	A hierarchical binomial model	101
4	3.5.2	A hierarchical Gaussian model	103
5	3.6	Empirical Bayes	106
6	3.6.1	A hierarchical binomial model	107
7	3.6.2	A hierarchical Gaussian model	108
8	3.6.3	Hierarchical Bayes for n-gram smoothing	109
9	3.7	Model selection and evaluation	111
10	3.7.1	Bayesian model selection	111
11	3.7.2	Estimating the marginal likelihood	112
12	3.7.3	Connection between cross validation and marginal likelihood	113
13	3.7.4	Pareto-Smoothed Importance Sampling LOO estimate	114
14	3.7.5	Information criteria	115
15	3.7.6	Posterior predictive checks	117
16	3.7.7	Bayesian p-values	118
17	3.8	Bayesian decision theory	120
18	3.8.1	Basics	121
19	3.8.2	Example: COVID-19	121
20	3.8.3	One-shot decision problems	122
21	3.8.4	Multi-stage decision problems	123
22	<b>4</b>	<b>Probabilistic graphical models</b>	<b>125</b>
23	4.1	Introduction	125
24	4.2	Directed graphical models (Bayes nets)	125
25	4.2.1	Representing the joint distribution	125
26	4.2.2	Examples	126
27	4.2.3	Conditional independence properties	131
28	4.2.4	Generation (sampling)	136
29	4.2.5	Inference	136
30	4.2.6	Learning	138
31	4.2.7	Plate notation	143
32	4.3	Undirected graphical models (Markov random fields)	146
33	4.3.1	Representing the joint distribution	146
34	4.3.2	Examples	148
35	4.3.3	Conditional independence properties	155
36	4.3.4	Generation (sampling)	157
37	4.3.5	Inference	157
38	4.3.6	Learning	158
39	4.4	Comparing directed and undirected PGMs	162
40	4.4.1	CI properties	162
41	4.4.2	Converting between a directed and undirected model	164
42	4.4.3	Combining directed and undirected graphs	165
43	4.4.4	Comparing directed and undirected Gaussian PGMs	167
44	4.4.5	Factor graphs	169

1	4.5	Extensions of Bayes nets	172
2	4.5.1	Probabilistic circuits	172
3	4.5.2	Relational probability models	173
4	4.5.3	Open-universe probability models	175
5	4.5.4	Programs as probability models	177
6	4.6	Structural causal models	177
7	4.6.1	Example: causal impact of education on wealth	178
8	4.6.2	Structural equation models	179
9	4.6.3	Do operator and augmented DAGs	179
10	4.6.4	Estimating average treatment effect using path analysis	180
11	4.6.5	Counterfactuals	181
12	<b>5</b>	<b>Information theory</b>	<b>185</b>
13	5.1	KL divergence	185
14	5.1.1	Desiderata	186
15	5.1.2	The KL divergence uniquely satisfies the desiderata	187
16	5.1.3	Thinking about KL	190
17	5.1.4	Properties of KL	192
18	5.1.5	KL divergence and MLE	194
19	5.1.6	KL divergence and Bayesian Inference	195
20	5.1.7	KL divergence and Exponential Families	196
21	5.2	Entropy	197
22	5.2.1	Definition	197
23	5.2.2	Differential entropy for continuous random variables	198
24	5.2.3	Typical sets	199
25	5.2.4	Cross entropy and perplexity	201
26	5.3	Mutual information	202
27	5.3.1	Definition	202
28	5.3.2	Interpretation	202
29	5.3.3	Data processing inequality	203
30	5.3.4	Sufficient Statistics	204
31	5.3.5	Multivariate mutual information	204
32	5.3.6	Variational bounds on mutual information	207
33	5.4	Data compression (source coding)	210
34	5.4.1	Lossless compression	210
35	5.4.2	Lossy compression and the rate-distortion tradeoff	210
36	5.4.3	Bits back coding	213
37	5.5	Error-correcting codes (channel coding)	213
38	5.6	The information bottleneck	215
39	5.6.1	Vanilla IB	215
40	5.6.2	Variational IB	216
41	5.6.3	Conditional entropy bottleneck	217
42	<b>6</b>	<b>Optimization</b>	<b>221</b>
43	6.1	Introduction	221

1	6.2	Automatic differentiation	221
2	6.2.1	Differentiation in functional form	221
3	6.2.2	Differentiating chains, circuits, and programs	226
4	6.3	Stochastic gradient descent	231
5	6.4	Natural gradient descent	232
6	6.4.1	Defining the natural gradient	232
7	6.4.2	Interpretations of NGD	233
8	6.4.3	Benefits of NGD	234
9	6.4.4	Approximating the natural gradient	235
10	6.4.5	Natural gradients for the exponential family	236
11	6.5	Mirror descent	238
12	6.5.1	Bregman divergence	239
13	6.5.2	Proximal point method	240
14	6.5.3	PPM using Bregman divergence	240
15	6.6	Gradients of stochastic functions	240
16	6.6.1	Minibatch approximation to finite-sum objectives	241
17	6.6.2	Optimizing parameters of a distribution	241
18	6.6.3	Score function estimator (likelihood ratio trick)	242
19	6.6.4	Reparameterization trick	243
20	6.6.5	The delta method	245
21	6.6.6	Gumbel softmax trick	245
22	6.6.7	Stochastic computation graphs	246
23	6.6.8	Straight-through estimator	246
24	6.7	Bound optimization (MM) algorithms	247
25	6.7.1	The general algorithm	247
26	6.7.2	Example: logistic regression	248
27	6.7.3	The EM algorithm	250
28	6.7.4	Example: EM for an MVN with missing data	252
29	6.7.5	Example: robust linear regression using Student- <i>t</i> likelihood	254
30	6.7.6	Extensions to EM	255
31	6.8	The Bayesian learning rule	257
32	6.8.1	Deriving inference algorithms from BLR	258
33	6.8.2	Deriving optimization algorithms from BLR	260
34	6.8.3	Variational optimization	263
35	6.9	Bayesian optimization	264
36	6.9.1	Sequential model-based optimization	265
37	6.9.2	Surrogate functions	265
38	6.9.3	Acquisition functions	267
39	6.9.4	Other issues	270
40	6.10	Optimal Transport	271
41	6.10.1	Warm-up: Matching optimally two families of points	271
42	6.10.2	From Optimal Matchings to Kantorovich and Monge formulations	272
43	6.10.3	Solving optimal transport	275
44	6.11	Submodular optimization	279
45	6.11.1	Intuition, Examples, and Background	280

1	6.11.2	Submodular Basic Definitions	282
2	6.11.3	Example Submodular Functions	283
3	6.11.4	Submodular Optimization	286
4	6.11.5	Applications of Submodularity in Machine Learning and AI	290
5	6.11.6	Sketching, CoreSets, Distillation, and Data Subset & Feature Selection	290
6	6.11.7	Combinatorial Information Functions	294
7	6.11.8	Clustering, Data Partitioning, and Parallel Machine Learning	295
8	6.11.9	Active and Semi-Supervised Learning	296
9	6.11.10	Probabilistic Modeling	297
10	6.11.11	Structured Norms and Loss Functions	298
11	6.11.12	Conclusions	299
12	6.12	Derivative free optimization	299
13			
14			
15			
16	<b>II Inference</b>	<b>301</b>	
17	<b>7 Inference algorithms: an overview</b>	<b>303</b>	
18	7.1	Introduction	303
19	7.2	Common inference patterns	303
20	7.2.1	Global latents	304
21	7.2.2	Local latents	304
22	7.2.3	Global and local latents	305
23	7.3	Exact inference algorithms	305
24	7.4	Approximate inference algorithms	306
25	7.4.1	MAP estimation	306
26	7.4.2	Grid approximation	306
27	7.4.3	Laplace (quadratic) approximation	307
28	7.4.4	Variational inference	308
29	7.4.5	Markov Chain Monte Carlo (MCMC)	310
30	7.4.6	Sequential Monte Carlo	311
31	7.5	Evaluating approximate inference algorithms	312
32			
33	<b>8 State-space inference</b>	<b>315</b>	
34	8.1	Introduction	315
35	8.1.1	State space models	315
36	8.1.2	Example: casino HMM	317
37	8.1.3	Example: linear-Gaussian SSM for tracking in 2d	318
38	8.1.4	Inferential goals	318
39	8.2	Bayesian filtering and smoothing	321
40	8.2.1	The filtering equations	322
41	8.2.2	The smoothing equations	322
42	8.3	Inference for discrete SSMs	323
43	8.3.1	Forwards filtering	323
44	8.3.2	Backwards smoothing	325
45	8.3.3	The forwards-backwards algorithm	325
46			
47			

1			
2	8.3.4	Two-slice smoothed marginals	327
3	8.3.5	Time and space complexity	328
4	8.3.6	The Viterbi algorithm	329
5	8.3.7	Forwards filtering, backwards sampling	332
6	8.3.8	Application to discretized state spaces	332
7	8.4	Inference for linear-Gaussian SSMs	333
8	8.4.1	The Kalman filter	333
9	8.4.2	Kalman filtering for linear regression (recursive least squares)	338
10	8.4.3	Predictive coding as Kalman filtering	340
11	8.4.4	The Kalman (RTS) smoother	342
12	8.5	Inference based on local linearization	343
13	8.5.1	Taylor series expansion	343
14	8.5.2	The extended Kalman filter (EKF)	346
15	8.5.3	The extended Kalman smoother	349
16	8.5.4	Exponential-family EKF	349
17	8.6	Inference based on the unscented transform	351
18	8.6.1	The unscented transform	352
19	8.6.2	The unscented Kalman filter (UKF)	353
20	8.6.3	The unscented Kalman smoother	355
21	8.7	Other variants of the Kalman filter	356
22	8.7.1	Ensemble Kalman filter	356
23	8.7.2	Robust Kalman filters	357
24	8.7.3	Gaussian filtering	357
25	8.8	Assumed density filtering	360
26	8.8.1	The ADF algorithm	360
27	8.8.2	Connection with Gaussian filtering	361
28	8.8.3	The Gaussian sum filter for switching SSMs	361
29	8.8.4	ADF for training logistic regression	364
30	9	<b>Message passing inference</b>	<b>369</b>
31	9.1	Introduction	369
32	9.2	Belief propagation on trees	370
33	9.2.1	BP for polytrees	370
34	9.2.2	BP for undirected graphs with pairwise potentials	373
35	9.2.3	BP for factor graphs	374
36	9.2.4	Max product belief propagation	375
37	9.2.5	Gaussian and non-Gaussian belief propagation	377
38	9.3	Loopy belief propagation	377
39	9.3.1	Convergence	378
40	9.3.2	Accuracy	380
41	9.3.3	Connection with variational inference	381
42	9.3.4	Generalized belief propagation	381
43	9.3.5	Application: error correcting codes	381
44	9.3.6	Application: Affinity propagation	383
45	9.3.7	Emulating BP with graph neural nets	384
46			
47			

1	9.4	The variable elimination (VE) algorithm	385
2	9.4.1	Derivation of the algorithm	385
3	9.4.2	Computational complexity of VE	386
4	9.4.3	Computational complexity of exact inference	388
5	9.4.4	Drawbacks of VE	389
6	9.5	The junction tree algorithm (JTA)	390
7	9.5.1	Creating a junction tree	390
8	9.5.2	Running belief propagation on a junction tree	395
9	9.5.3	The generalized distributive law	396
10	9.5.4	Other applications of the JTA	397
11	9.6	Inference as backpropagation	397
12	<b>10</b>	<b>Variational inference</b>	<b>401</b>
13	10.1	Introduction	401
14	10.1.1	Variational free energy	401
15	10.1.2	Evidence lower bound (ELBO)	402
16	10.2	Mean field VI	403
17	10.2.1	Coordinate ascent variational inference (CAVI)	403
18	10.2.2	Example: CAVI for the Ising model	404
19	10.2.3	Variational Bayes	406
20	10.2.4	Example: VB for a univariate Gaussian	407
21	10.2.5	Variational Bayes EM	410
22	10.2.6	Example: VBEM for a GMM	411
23	10.2.7	Variational message passing (VMP)	417
24	10.2.8	Autoconj	418
25	10.3	Fixed-form VI	418
26	10.3.1	Black-box variational inference	418
27	10.3.2	Stochastic variational inference	420
28	10.3.3	Reparameterization VI	421
29	10.3.4	Gaussian VI	422
30	10.3.5	Automatic differentiation VI	426
31	10.3.6	Beyond Gaussian posteriors	427
32	10.3.7	Amortized inference	429
33	10.3.8	Exploiting partial conjugacy	430
34	10.3.9	Online variational inference	434
35	10.4	More accurate variational posteriors	437
36	10.4.1	Structured mean field	438
37	10.4.2	Hierarchical (auxiliary variable) posteriors	438
38	10.4.3	Normalizing flow posteriors	438
39	10.4.4	Implicit posteriors	440
40	10.4.5	Combining VI with MCMC inference	441
41	10.5	Lower bounds	441
42	10.5.1	Multi-sample ELBO (IWAE bound)	441
43	10.5.2	The thermodynamic variational objective (TVO)	442
44	10.6	Upper bounds	442

1	10.6.1	Minimizing the $\chi$ -divergence upper bound	443
2	10.6.2	Minimizing the evidence upper bound	444
3	10.7	Expectation propagation (EP)	445
4	10.7.1	Minimizing forwards vs reverse KL	445
5	10.7.2	EP as generalized ADF	447
6	10.7.3	Algorithm	447
7	10.7.4	Example	448
8	10.7.5	Optimization issues	448
9	10.7.6	Power EP and $\alpha$ -divergence	449
10	10.7.7	Stochastic EP	449
11	10.7.8	Applications	450
12	<b>11</b>	<b>Monte Carlo inference</b>	<b>451</b>
13	11.1	Introduction	451
14	11.2	Monte Carlo integration	451
15	11.2.1	Example: estimating $\pi$ by Monte Carlo integration	452
16	11.2.2	Accuracy of Monte Carlo integration	452
17	11.3	Generating random samples from simple distributions	454
18	11.3.1	Sampling using the inverse cdf	454
19	11.3.2	Sampling from a Gaussian (Box-Muller method)	455
20	11.4	Rejection sampling	455
21	11.4.1	Basic idea	456
22	11.4.2	Example	457
23	11.4.3	Adaptive rejection sampling	457
24	11.4.4	Rejection sampling in high dimensions	458
25	11.5	Importance sampling	458
26	11.5.1	Direct importance sampling	459
27	11.5.2	Self-normalized importance sampling	459
28	11.5.3	Choosing the proposal	460
29	11.5.4	Annealed importance sampling (AIS)	460
30	11.6	Controlling Monte Carlo variance	462
31	11.6.1	Rao-Blackwellisation	462
32	11.6.2	Control variates	463
33	11.6.3	Antithetic sampling	464
34	11.6.4	Quasi Monte Carlo (QMC)	465
35	<b>12</b>	<b>Markov Chain Monte Carlo inference</b>	<b>467</b>
36	12.1	Introduction	467
37	12.2	Metropolis Hastings algorithm	467
38	12.2.1	Basic idea	468
39	12.2.2	Why MH works	469
40	12.2.3	Proposal distributions	470
41	12.2.4	Initialization	473
42	12.2.5	Simulated annealing	473
43	12.3	Gibbs sampling	475

1	12.3.1	Basic idea	476
2	12.3.2	Gibbs sampling is a special case of MH	476
3	12.3.3	Example: Gibbs sampling for Ising models	477
4	12.3.4	Example: Gibbs sampling for Potts models	478
5	12.3.5	Example: Gibbs sampling for GMMs	479
6	12.3.6	Sampling from the full conditionals	481
7	12.3.7	Blocked Gibbs sampling	481
8	12.3.8	Collapsed Gibbs sampling	482
9	12.4	Auxiliary variable MCMC	484
10	12.4.1	Slice sampling	485
11	12.4.2	Swendsen Wang	487
12	12.5	Hamiltonian Monte Carlo (HMC)	488
13	12.5.1	Hamiltonian mechanics	488
14	12.5.2	Integrating Hamilton's equations	489
15	12.5.3	The HMC algorithm	491
16	12.5.4	Tuning HMC	492
17	12.5.5	Riemann Manifold HMC	493
18	12.5.6	Langevin Monte Carlo (MALA)	493
19	12.5.7	Connection between SGD and Langevin sampling	494
20	12.5.8	Applying HMC to constrained parameters	496
21	12.5.9	Speeding up HMC	497
22	12.6	MCMC convergence	497
23	12.6.1	Mixing rates of Markov chains	498
24	12.6.2	Practical convergence diagnostics	499
25	12.6.3	Improving speed of convergence	506
26	12.6.4	Non-centered parameterizations and Neal's funnel	506
27	12.7	Stochastic gradient MCMC	508
28	12.7.1	Stochastic Gradient Langevin Dynamics (SGLD)	508
29	12.7.2	Preconditioning	509
30	12.7.3	Reducing the variance of the gradient estimate	510
31	12.7.4	SG-HMC	511
32	12.7.5	Underdamped Langevin Dynamics	511
33	12.8	Reversible jump (trans-dimensional) MCMC	513
34	12.8.1	Basic idea	514
35	12.8.2	Example	515
36	12.8.3	Discussion	516
37	12.9	Annealing methods	516
38	12.9.1	Parallel tempering	516
39	13	Sequential Monte Carlo inference	517
40	13.1	Introduction	517
41	13.1.1	Problem statement	517
42	13.1.2	Particle filtering for state-space models	517
43	13.1.3	SMC samplers for static parameter estimation	519
44	13.2	Basics of SMC	519
45			
46			
47			

1	13.2.1	Importance sampling	519
2	13.2.2	Sequential importance sampling	520
3	13.2.3	Sequential importance sampling with resampling	521
4	13.2.4	Resampling methods	524
5	13.2.5	Adaptive resampling	526
6	13.3	Some applications of particle filtering	527
7	13.3.1	1d pendulum model with outliers	527
8	13.3.2	Visual object tracking	529
9	13.3.3	Robot localization	529
10	13.3.4	Online parameter estimation	531
11	13.4	Proposal distributions	531
12	13.4.1	Locally optimal proposal	532
13	13.4.2	Proposals based on the Laplace approximation	532
14	13.4.3	Proposals based on the extended and unscented Kalman filter	534
15	13.4.4	Proposals based on SMC	534
16	13.4.5	Neural adaptive SMC	535
17	13.4.6	Amortized adaptive SMC	535
18	13.4.7	Variational SMC	536
19	13.5	Rao-Blackwellised particle filtering (RBPF)	537
20	13.5.1	Mixture of Kalman filters	537
21	13.5.2	FastSLAM	539
22	13.6	SMC samplers	541
23	13.6.1	Ingredients of an SMC sampler	541
24	13.6.2	Likelihood tempering (geometric path)	543
25	13.6.3	Data tempering	545
26	13.6.4	Sampling rare events and extrema	547
27	13.6.5	SMC-ABC and likelihood-free inference	547
28	13.6.6	SMC <sup>2</sup>	548
29	13.7	Particle MCMC methods	548
30	13.7.1	Particle Marginal Metropolis Hastings	549
31	13.7.2	Particle Independent Metropolis Hastings	549
32	13.7.3	Particle Gibbs	550
33			
34			
35			
36	<b>III</b>	<b>Prediction</b>	<b>553</b>
37			
38	<b>14</b>	<b>Predictive models: an overview</b>	<b>555</b>
39	14.1	Introduction	555
40	14.1.1	Types of model	555
41	14.1.2	Model fitting using ERM, MLE and MAP	556
42	14.1.3	Model fitting using Bayes, VI and generalized Bayes	557
43	14.2	Evaluating predictive models	558
44	14.2.1	Proper scoring rules	558
45	14.2.2	Calibration	558
46	14.2.3	Beyond evaluating marginal probabilities	562
47			

1	14.3	Conformal prediction	565
2	14.3.1	Conformalizing classification	566
3	14.3.2	Conformalizing regression	567
4	14.3.3	Conformalizing Bayes	568
5	14.3.4	What do we do if we don't have a calibration set?	569
6	<b>15 Generalized linear models</b>	<b>571</b>	
7	15.1	Introduction	571
8	15.1.1	Examples	571
9	15.1.2	GLMs with non-canonical link functions	574
10	15.1.3	Maximum likelihood estimation	574
11	15.1.4	Bayesian inference	575
12	15.2	Linear regression	576
13	15.2.1	Conjugate priors	576
14	15.2.2	Uninformative priors	578
15	15.2.3	Informative priors	580
16	15.2.4	Spike and slab prior	582
17	15.2.5	Laplace prior (Bayesian lasso)	583
18	15.2.6	Horseshoe prior	584
19	15.2.7	Automatic relevancy determination	585
20	15.3	Logistic regression	587
21	15.3.1	Binary logistic regression	588
22	15.3.2	Multinomial logistic regression	588
23	15.3.3	Priors	589
24	15.3.4	Posteriors	590
25	15.3.5	Laplace approximation	590
26	15.3.6	MCMC inference	593
27	15.3.7	Variational inference	594
28	15.4	Probit regression	594
29	15.4.1	Latent variable interpretation	594
30	15.4.2	Maximum likelihood estimation	595
31	15.4.3	Bayesian inference	596
32	15.4.4	Ordinal probit regression	597
33	15.4.5	Multinomial probit models	598
34	15.5	Multi-level GLMs	598
35	15.5.1	Generalized linear mixed models (GLMMs)	598
36	15.5.2	Model fitting	599
37	15.5.3	Example: radon regression	599
38	<b>16 Deep neural networks</b>	<b>603</b>	
39	16.1	Introduction	603
40	16.2	Building blocks of differentiable circuits	603
41	16.2.1	Linear layers	604
42	16.2.2	Non-linearities	604
43	16.2.3	Convolutional layers	605

1	16.2.4	Residual (skip) connections	606
2	16.2.5	Normalization layers	607
3	16.2.6	Dropout layers	607
4	16.2.7	Attention layers	608
5	16.2.8	Recurrent layers	611
6	16.2.9	Multiplicative layers	611
7	16.2.10	Implicit layers	612
8	16.3	Canonical examples of neural networks	612
9	16.3.1	Multi-layer perceptrons (MLP)	613
10	16.3.2	Convolutional neural networks (CNN)	613
11	16.3.3	Recurrent neural networks (RNN)	613
12	16.3.4	Transformers	615
13	16.3.5	Graph neural networks (GNNs)	618
14	<b>17 Bayesian neural networks</b>	<b>625</b>	
15	17.1	Introduction	625
16	17.2	Priors for BNNs	625
17	17.2.1	Gaussian priors	626
18	17.2.2	Sparsity-promoting priors	627
19	17.2.3	Learning the prior	628
20	17.2.4	Priors in function space	628
21	17.2.5	Architectural priors	628
22	17.3	Likelihoods for BNNs	629
23	17.4	Posterioris for BNNs	630
24	17.4.1	Laplace approximation	630
25	17.4.2	Variational inference	631
26	17.4.3	Expectation propagation	632
27	17.4.4	Last layer methods	632
28	17.4.5	Dropout	632
29	17.4.6	MCMC methods	633
30	17.4.7	Methods based on the SGD trajectory	633
31	17.4.8	Deep ensembles	634
32	17.4.9	Approximating the posterior predictive distibution	638
33	17.5	Generalization in Bayesian deep learning	639
34	17.5.1	Sharp vs flat minima	639
35	17.5.2	Effective dimensionality of a model	640
36	17.5.3	The hypothesis space of DNNs	642
37	17.5.4	Double descent	643
38	17.5.5	A Bayesian Resolution to Double Descent	644
39	17.5.6	PAC-Bayes	646
40	17.5.7	Out-of-Distribution Generalization for BNNs	647
41	17.6	Online inference	650
42	17.6.1	Extended Kalman Filtering for DNNs	650
43	17.6.2	Assumed Density Filtering for DNNs	652
44	17.6.3	Sequential Laplace for DNNs	654

1	17.6.4	Variational methods	654
2	17.7	Hierarchical Bayesian neural networks	654
3	17.7.1	Solving multiple related classification problems	655
4	<b>18 Gaussian processes</b>	<b>659</b>	
5	18.1	Introduction	659
6	18.2	Mercer kernels	661
7	18.2.1	Some popular Mercer kernels	662
8	18.2.2	Mercer's theorem	667
9	18.2.3	Kernels from Spectral Densities	668
10	18.3	GPs with Gaussian likelihoods	670
11	18.3.1	Predictions using noise-free observations	670
12	18.3.2	Predictions using noisy observations	671
13	18.3.3	Weight space vs function space	672
14	18.3.4	Semi-parametric GPs	673
15	18.3.5	Marginal likelihood	674
16	18.3.6	Computational and numerical issues	674
17	18.3.7	Kernel ridge regression	675
18	18.4	GPs with non-Gaussian likelihoods	678
19	18.4.1	Binary classification	678
20	18.4.2	Multi-class classification	680
21	18.4.3	GPs for Poisson regression (Cox process)	680
22	18.5	Scaling GP inference to large datasets	681
23	18.5.1	Subset of data	682
24	18.5.2	Nyström approximation	683
25	18.5.3	Inducing point methods	684
26	18.5.4	Sparse variational methods	687
27	18.5.5	Exploiting parallelization and structure via kernel matrix multiplies	690
28	18.5.6	Converting a GP to a SSM	693
29	18.6	Learning the kernel	693
30	18.6.1	Empirical Bayes for the kernel parameters	693
31	18.6.2	Bayesian inference for the kernel parameters	696
32	18.6.3	Multiple kernel learning for additive kernels	697
33	18.6.4	Automatic search for compositional kernels	699
34	18.6.5	Spectral mixture kernel learning	701
35	18.6.6	Deep kernel learning	703
36	18.6.7	Functional kernel learning	704
37	18.7	GPs and DNNs	705
38	18.7.1	Kernels derived from random DNNs (NN-GP)	706
39	18.7.2	Kernels derived from trained DNNs (neural tangent kernel)	709
40	18.7.3	Deep GPs	711
41	<b>19 Structured prediction</b>	<b>717</b>	
42	19.1	Introduction	717
43	19.2	Conditional random fields (CRFs)	717

1			
2	19.2.1	1d CRFs	717
3	19.2.2	2d CRFs	721
4	19.2.3	Parameter estimation	723
5	19.2.4	Other approaches	724
6	19.3	Time series forecasting	725
7	19.3.1	Structural time series models	725
8	19.3.2	Prophet	731
9	19.3.3	Gaussian processes for timeseries forecasting	732
10	19.3.4	Neural forecasting methods	733
11	19.3.5	Causal impact of a time series intervention	734
12	<b>20</b>	<b>Beyond the iid assumption</b>	<b>739</b>
13	20.1	Introduction	739
14	20.2	Distribution shift	739
15	20.2.1	Motivating examples	739
16	20.2.2	A causal view of distribution shift	741
17	20.2.3	Covariate shift	742
18	20.2.4	Domain shift	742
19	20.2.5	Label / prior shift	743
20	20.2.6	Concept shift	743
21	20.2.7	Manifestation shift	743
22	20.2.8	Selection bias	744
23	20.3	Training-time techniques for distribution shift	744
24	20.3.1	Importance weighting for covariate shift	745
25	20.3.2	Domain adaptation	746
26	20.3.3	Domain randomization	746
27	20.3.4	Data augmentation	747
28	20.3.5	Unsupervised label shift estimation	747
29	20.3.6	Distributionally robust optimization	748
30	20.4	Test-time techniques for distribution shift	748
31	20.4.1	Detecting shifts using two-sample testing	748
32	20.4.2	Detecting single out-of-distribution (OOD) inputs	748
33	20.4.3	Selective prediction	751
34	20.4.4	Open world recognition	752
35	20.4.5	Online adaptation	753
36	20.5	Learning from multiple distributions	754
37	20.5.1	Transfer learning	754
38	20.5.2	Few-shot learning	755
39	20.5.3	Prompt tuning	755
40	20.5.4	Zero-shot learning	756
41	20.5.5	Multi-task learning	756
42	20.5.6	Domain generalization	758
43	20.5.7	Invariant risk minimization	758
44	20.6	Meta-learning	759
45	20.6.1	Meta-learning as probabilistic inference for prediction	760
46			
47			

<u>1</u>	20.6.2	Gradient-based meta-learning	761
<u>2</u>	20.6.3	Metric-based few-shot learning	761
<u>3</u>	20.6.4	VERSA	762
<u>4</u>	20.6.5	Neural processes	762
<u>5</u>	20.7	Continual learning	762
<u>6</u>	20.7.1	Domain drift	763
<u>7</u>	20.7.2	Concept drift	763
<u>8</u>	20.7.3	Task incremental learning	764
<u>9</u>	20.7.4	Catastrophic forgetting	766
<u>10</u>	20.7.5	Online learning	767
<u>11</u>	20.8	Adversarial examples	769
<u>12</u>	20.8.1	Whitebox (gradient-based) attacks	770
<u>13</u>	20.8.2	Blackbox (gradient-free) attacks	771
<u>14</u>	20.8.3	Real world adversarial attacks	772
<u>15</u>	20.8.4	Defenses based on robust optimization	772
<u>16</u>	20.8.5	Why models have adversarial examples	773
<u>17</u>			
<u>18</u>			
<u>19</u>			

## 20 IV Generation 775

<u>21</u>	21 Generative models: an overview	777	
<u>22</u>	21.1	Introduction	777
<u>23</u>	21.2	Types of generative model	777
<u>24</u>	21.3	Goals of generative modeling	779
<u>25</u>	21.3.1	Generating data	779
<u>26</u>	21.3.2	Density estimation	780
<u>27</u>	21.3.3	Imputation	781
<u>28</u>	21.3.4	Structure discovery	781
<u>29</u>	21.3.5	Latent space interpolation	782
<u>30</u>	21.3.6	Representation learning	783
<u>31</u>	21.4	Evaluating generative models	783
<u>32</u>	21.4.1	Likelihood	784
<u>33</u>	21.4.2	Distances and divergences in feature space	786
<u>34</u>	21.4.3	Precision and recall metrics	787
<u>35</u>	21.4.4	Statistical tests	788
<u>36</u>	21.4.5	Challenges with using pretrained classifiers	788
<u>37</u>	21.4.6	Using model samples to train classifiers	788
<u>38</u>	21.4.7	Assessing overfitting	789
<u>39</u>	21.4.8	Human evaluation	789
<u>40</u>			
<u>41</u>			

## 42 22 Variational autoencoders 791

<u>43</u>	22.1	Introduction	791
<u>44</u>	22.2	VAE basics	791
<u>45</u>	22.2.1	Modeling assumptions	792
<u>46</u>	22.2.2	Evidence lower bound	793
<u>47</u>			

1	22.2.3	Optimization	794
2	22.2.4	The reparameterization trick	794
3	22.2.5	Computing the reparameterized ELBO	796
4	22.2.6	Comparison of VAEs and autoencoders	798
5	22.2.7	VAEs optimize in an augmented space	799
6	22.3	VAE generalizations	801
7	22.3.1	$\sigma$ -VAE	801
8	22.3.2	$\beta$ -VAE	802
9	22.3.3	InfoVAE	804
10	22.3.4	Multi-modal VAEs	806
11	22.3.5	VAEs with missing data	809
12	22.3.6	Semi-supervised VAEs	811
13	22.3.7	VAEs with sequential encoders/decoders	812
14	22.4	Avoiding posterior collapse	815
15	22.4.1	KL annealing	816
16	22.4.2	Lower bounding the rate	816
17	22.4.3	Free bits	816
18	22.4.4	Adding skip connections	816
19	22.4.5	Improved variational inference	816
20	22.4.6	Alternative objectives	817
21	22.4.7	Enforcing identifiability	817
22	22.5	VAEs with hierarchical structure	818
23	22.5.1	Bottom-up vs top-down inference	819
24	22.5.2	Example: Very deep VAE	820
25	22.5.3	Connection with autoregressive models	822
26	22.5.4	Variational pruning	823
27	22.5.5	Other optimization difficulties	823
28	22.6	Vector quantization VAE	824
29	22.6.1	Autoencoder with binary code	824
30	22.6.2	VQ-VAE model	825
31	22.6.3	Learning the prior	827
32	22.6.4	Hierarchical extension (VQ-VAE-2)	827
33	22.6.5	Discrete VAE	828
34	22.6.6	VQ-GAN	829
35	22.7	Wake-sleep algorithm	830
36	22.7.1	Wake phase	830
37	22.7.2	Sleep phase	831
38	22.7.3	Daydream phase	832
39	22.7.4	Summary of algorithm	833
40	23	Auto-regressive models	835
41	23.1	Introduction	835
42	23.2	Neural autoregressive density estimators (NADE)	836
43	23.3	Causal CNNs	836
44	23.3.1	1d causal CNN (Convolutional Markov models)	837

1	23.3.2	2d causal CNN (PixelCNN)	837
2	23.4	Transformer decoders	838
3	23.4.1	Text generation (GPT)	839
4	23.4.2	Music generation	839
5	23.4.3	Text-to-image generation (DALL-E)	840
6	<b>24 Normalizing Flows</b>	<b>843</b>	
7	24.1	Introduction	843
8	24.1.1	Preliminaries	843
9	24.1.2	Example	845
10	24.1.3	How to train a flow model	846
11	24.2	Constructing Flows	847
12	24.2.1	Affine flows	847
13	24.2.2	Elementwise flows	848
14	24.2.3	Coupling flows	850
15	24.2.4	Autoregressive flows	852
16	24.2.5	Residual flows	856
17	24.2.6	Continuous-time flows	859
18	24.3	Applications	860
19	24.3.1	Density estimation	860
20	24.3.2	Generative Modeling	861
21	24.3.3	Inference	861
22	<b>25 Energy-based models</b>	<b>863</b>	
23	25.1	Introduction	863
24	25.1.1	Example: Products of experts (PoE)	864
25	25.1.2	Computational difficulties	864
26	25.2	Maximum Likelihood Training	865
27	25.2.1	Gradient-based MCMC methods	866
28	25.2.2	Contrastive divergence	866
29	25.3	Score Matching (SM)	869
30	25.3.1	Basic score matching	870
31	25.3.2	Denoising Score Matching (DSM)	871
32	25.3.3	Sliced Score Matching (SSM)	872
33	25.3.4	Connection to Contrastive Divergence	873
34	25.3.5	Score-Based Generative Models	874
35	25.4	Noise Contrastive Estimation	877
36	25.4.1	Connection to Score Matching	878
37	25.5	Other Methods	879
38	25.5.1	Minimizing Differences/Derivatives of KL Divergences	879
39	25.5.2	Minimizing the Stein Discrepancy	880
40	25.5.3	Adversarial Training	880
41	<b>26 Denoising diffusion models</b>	<b>883</b>	
42	26.1	Model definition	883
43			
44			
45			
46			
47			

1	26.2 Examples	885
2	26.3 Model training	886
3	26.4 Connections with other generative models	888
4	26.4.1 Connection with score matching	888
5	26.4.2 Connection with VAEs	889
6	26.4.3 Connection with flow models	889
7		
8	<b>27 Generative adversarial networks</b>	<b>891</b>
9		
10	27.1 Introduction	891
11	27.2 Learning by Comparison	892
12	27.2.1 Guiding principles	893
13	27.2.2 Class probability estimation	894
14	27.2.3 Bounds on $f$ -divergences	897
15	27.2.4 Integral probability metrics	898
16	27.2.5 Moment matching	900
17	27.2.6 On density ratios and differences	901
18	27.3 Generative Adversarial Networks	902
19	27.3.1 From learning principles to loss functions	903
20	27.3.2 Gradient Descent	904
21	27.3.3 Challenges with GAN training	905
22	27.3.4 Improving GAN optimization	907
23	27.3.5 Convergence of GAN training	907
24	27.4 Conditional GANs	910
25	27.5 Inference with GANs	912
26	27.6 Neural architectures in GANs	912
27	27.6.1 The importance of discriminator architectures	913
28	27.6.2 Architectural inductive biases	913
29	27.6.3 Attention in GANs	913
30	27.6.4 Progressive generation	914
31	27.6.5 Regularization	915
32	27.6.6 Scaling up GAN models	916
33	27.7 Applications	916
34	27.7.1 GANs for image generation	917
35	27.7.2 Video generation	919
36	27.7.3 Audio generation	920
37	27.7.4 Text generation	920
38	27.7.5 Imitation Learning	921
39	27.7.6 Domain Adaptation	922
40	27.7.7 Design, Art and Creativity	922
41		
42		
43	<b>V Discovery</b>	<b>923</b>
44		
45	<b>28 Discovery methods: an overview</b>	<b>925</b>
46	28.1 Introduction	925
47		

1	28.2	Overview of Part V	926
2	<b>29 Latent variable models</b>	<b>927</b>	
3	29.1	Introduction	927
4	29.2	Mixture models	927
5	29.2.1	Gaussian mixture models (GMMs)	928
6	29.2.2	Bernoulli mixture models	930
7	29.2.3	Gaussian scale mixtures	930
8	29.2.4	Using GMMs as a prior for inverse imaging problems	932
9	29.3	Factor analysis	935
10	29.3.1	Vanilla factor analysis	935
11	29.3.2	Probabilistic PCA	939
12	29.3.3	Factor analysis models for paired data	942
13	29.3.4	Factor analysis with exponential family likelihoods	945
14	29.3.5	Factor analysis with DNN likelihoods	946
15	29.3.6	Factor analysis with GP likelihoods (GP-LVM)	947
16	29.4	Mixture of factor analysers	949
17	29.4.1	Model definition	949
18	29.4.2	Model fitting	950
19	29.4.3	MixFA for image generation	951
20	29.5	LVMs with non-Gaussian priors	955
21	29.5.1	Non-negative matrix factorization (NMF)	955
22	29.5.2	Multinomial PCA	956
23	29.5.3	Latent Dirichlet Allocation (LDA)	958
24	29.6	Independent components analysis (ICA)	959
25	29.6.1	Noiseless ICA model	960
26	29.6.2	The need for non-Gaussian priors	960
27	29.6.3	Maximum likelihood estimation	961
28	29.6.4	Alternatives to MLE	962
29	29.6.5	Sparse coding	964
30	29.6.6	Nonlinear ICA	965
31	<b>30 Hidden Markov models</b>	<b>967</b>	
32	30.1	Introduction	967
33	30.2	HMMs: parameterization	967
34	30.2.1	Transition model	967
35	30.2.2	Observation model	968
36	30.3	HMMs: Applications	971
37	30.3.1	Segmentation of time series data	971
38	30.3.2	Spelling correction	973
39	30.3.3	Protein sequence alignment	976
40	30.4	HMMs: parameter learning	977
41	30.4.1	The Baum-Welch (EM) algorithm	977
42	30.4.2	Parameter estimation using SGD	981
43	30.4.3	Parameter estimation using spectral methods	984
44			
45			
46			
47			

1	30.4.4	Bayesian parameter inference	984
2	30.5	HMMs: Generalizations	985
3	30.5.1	Hidden semi-Markov model (HSMM)	985
4	30.5.2	HSMMs for changepoint detection	987
5	30.5.3	Hierarchical HMMs	990
6	30.5.4	Factorial HMMs	992
7	30.5.5	Coupled HMMs	995
8	30.5.6	Dynamic Bayes nets (DBN)	995
9	<b>31 State-space models</b>	<b>997</b>	
10	31.1	Introduction	997
11	31.2	Linear dynamical systems	997
12	31.2.1	Example: Noiseless 1d spring-mass system	998
13	31.2.2	Example: Noisy 2d tracking problem	999
14	31.2.3	Example: Online linear regression	1002
15	31.2.4	Example: structural time series forecasting	1004
16	31.2.5	Parameter estimation	1004
17	31.3	Non-linear dynamical systems	1006
18	31.3.1	Example: nonlinear 2d tracking problem	1007
19	31.3.2	Example: Simultaneous localization and mapping (SLAM)	1007
20	31.3.3	Example: stochastic volatility models	1009
21	31.3.4	Example: Multi-target tracking	1010
22	31.4	Other kinds of SSM	1012
23	31.4.1	Exponential family SSM	1012
24	31.4.2	Bayesian SSM	1016
25	31.4.3	GP-SSM	1016
26	31.5	Deep state space models	1017
27	31.5.1	Deep Markov models	1017
28	31.5.2	Recurrent SSM	1018
29	31.5.3	Improving multi-step predictions	1019
30	31.5.4	Variational RNNs	1020
31	<b>32 Graph learning</b>	<b>1023</b>	
32	32.1	Introduction	1023
33	32.2	Latent variable models for graphs	1023
34	32.2.1	Stochastic block model	1023
35	32.2.2	Mixed membership stochastic block model	1025
36	32.2.3	Infinite relational model	1027
37	32.3	Graphical model structure learning	1029
38	32.3.1	Applications	1029
39	32.3.2	Relevance networks	1031
40	32.3.3	Learning sparse PGMs	1032
41	<b>33 Non-parametric Bayesian models</b>	<b>1033</b>	
42	33.1	Introduction	1033

1	33.2	Dirichlet process	1034
2	33.2.1	Definition	1034
3	33.2.2	Stick breaking construction of the DP	1036
4	33.2.3	The Chinese restaurant process (CRP)	1037
5	33.2.4	Dirichlet process mixture models	1039
6	33.3	Generalizations of the Dirichlet process	1044
7	33.3.1	Pitman-Yor process	1045
8	33.3.2	Dependent random probability measures	1046
9	33.4	The Indian buffet process and the Beta process	1048
10	33.5	Small-variance asymptotics	1051
11	33.6	Completely random measures	1054
12	33.7	Lévy processes	1055
13	33.8	Point processes with repulsion and reinforcement	1057
14	33.8.1	Poisson process	1057
15	33.8.2	Renewal process	1058
16	33.8.3	Hawkes process	1059
17	33.8.4	Gibbs point process	1061
18	33.8.5	Determinantal point process	1062
19	<b>34</b>	<b>Representation learning (Unfinished)</b>	<b>1065</b>
20	34.1	CLIP	1065
21	<b>35</b>	<b>Interpretability</b>	<b>1067</b>
22	35.1	Introduction	1067
23	35.1.1	The Role of Interpretability	1068
24	35.1.2	Terminology and Framework	1069
25	35.2	Methods for Interpretable Machine Learning	1073
26	35.2.1	Inherently Interpretable Models: The Model is its Explanation	1073
27	35.2.2	Semi-Inherently Interpretable Models: Example-Based Methods	1075
28	35.2.3	Post-hoc or Joint training: The Explanation gives a Partial View of the Model	1076
29	35.2.4	Transparency and Visualization	1080
30	35.3	Properties: The Abstraction Between Context and Method	1081
31	35.3.1	Properties of Explanations from Interpretable Machine Learning	1081
32	35.3.2	Properties of Explanations from Cognitive Science	1084
33	35.4	Evaluation of Interpretable Machine Learning Models	1085
34	35.4.1	Computational Evaluation: Does the Method have Desired Properties?	1086
35	35.4.2	User Study-based Evaluation: Does the Method Help a User Perform a Task?	1090
36	35.5	Discussion: How to Think about Interpretable Machine Learning	1094
37	<b>VI</b>	<b>Decision making</b>	<b>1101</b>
38	<b>36</b>	<b>Multi-step decision problems</b>	<b>1103</b>
39	36.1	Introduction	1103

1	36.2	Decision (influence) diagrams	1103
2	36.2.1	Example: oil wildcatter	1103
3	36.2.2	Information arcs	1104
4	36.2.3	Value of information	1105
5	36.2.4	Computing the optimal policy	1106
6	36.3	A/B testing	1106
7	36.3.1	A Bayesian approach	1107
8	36.3.2	Example	1110
9	36.4	Contextual bandits	1111
10	36.4.1	Types of bandit	1111
11	36.4.2	Applications	1113
12	36.4.3	Exploration-exploitation tradeoff	1113
13	36.4.4	The optimal solution	1113
14	36.4.5	Upper confidence bounds (UCB)	1115
15	36.4.6	Thompson sampling	1117
16	36.4.7	Regret	1118
17	36.5	Markov decision problems	1119
18	36.5.1	Basics	1120
19	36.5.2	Partially observed MDPs	1121
20	36.5.3	Episodes and returns	1122
21	36.5.4	Value functions	1122
22	36.5.5	Optimal value functions and policies	1123
23	36.6	Planning in an MDP	1124
24	36.6.1	Value iteration	1125
25	36.6.2	Policy iteration	1126
26	36.6.3	Linear programming	1127
27	<b>37 Reinforcement learning</b>		<b>1129</b>
28	37.1	Introduction	1129
29	37.1.1	Overview of methods	1129
30	37.1.2	Value based methods	1130
31	37.1.3	Policy search methods	1130
32	37.1.4	Model-based RL	1131
33	37.1.5	Exploration-exploitation tradeoff	1131
34	37.2	Value-based RL	1133
35	37.2.1	Monte Carlo RL	1133
36	37.2.2	Temporal difference (TD) learning	1134
37	37.2.3	TD learning with eligibility traces	1135
38	37.2.4	SARSA: on-policy TD control	1135
39	37.2.5	Q-learning: off-policy TD control	1136
40	37.2.6	Deep Q-network (DQN)	1139
41	37.3	Policy-based RL	1140
42	37.3.1	The policy gradient theorem	1140
43	37.3.2	REINFORCE	1141
44	37.3.3	Actor-critic methods	1141

1	37.3.4	Bound optimization methods	1143
2	37.3.5	Deterministic policy gradient methods	1145
3	37.3.6	Gradient-free methods	1146
4	37.4	Model-based RL	1146
5	37.4.1	Model predictive control (MPC)	1147
6	37.4.2	Combining model-based and model-free	1148
7	37.4.3	MBRL using Gaussian processes	1149
8	37.4.4	MBRL using DNNs	1150
9	37.4.5	MBRL using latent-variable models	1151
10	37.4.6	Robustness to model errors	1153
11	37.5	Off-policy learning	1153
12	37.5.1	Basic techniques	1154
13	37.5.2	The curse of horizon	1157
14	37.5.3	The deadly triad	1158
15	37.6	Control as inference	1160
16	37.6.1	Maximum entropy reinforcement learning	1160
17	37.6.2	Other approaches	1162
18	37.6.3	Imitation learning	1163
19	<b>38 Causality</b>	<b>1167</b>	
20	38.1	Introduction	1167
21	38.1.1	Why is causality different than other forms of ML?	1167
22	38.2	Causal Formalism	1169
23	38.2.1	Structural Causal Models	1169
24	38.2.2	Causal DAGs	1171
25	38.2.3	Identification	1173
26	38.2.4	Counterfactuals and the Causal Hierarchy	1174
27	38.3	Randomized Control Trials	1176
28	38.4	Confounder Adjustment	1177
29	38.4.1	Causal Estimand, Statistical Estimand, and Identification	1177
30	38.4.2	ATE Estimation with Observed Confounders	1180
31	38.4.3	Uncertainty Quantification	1185
32	38.4.4	Matching	1186
33	38.4.5	Practical Considerations and Procedures	1187
34	38.4.6	Summary and Practical Advice	1190
35	38.5	Instrumental Variable Strategies	1191
36	38.5.1	Additive Unobserved Confounding	1193
37	38.5.2	Instrument Monotonicity and Local Average Treatment Effect	1194
38	38.5.3	Two Stage Least Squares	1198
39	38.6	Difference in Differences	1198
40	38.6.1	Estimation	1202
41	38.7	Credibility Checks	1202
42	38.7.1	Placebo Checks	1203
43	38.7.2	Sensitivity Analysis to Unobserved Confounding	1203
44	38.8	The Do Calculus	1211
45			
46			
47			

<u>1</u>		
<u>2</u>	38.8.1	The three rules      1211
<u>3</u>	38.8.2	Revisiting Backdoor Adjustment      1212
<u>4</u>	38.8.3	Frontdoor Adjustment      1213
<u>5</u>	38.9	Further Reading      1215

<u>6</u>	<b>Bibliography</b>	<b>1229</b>
----------	---------------------	-------------

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



# 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 training and testing under 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 causal inference and decision making.

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 cover 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 \(KL divergence\)](#).
- Roy Frostig (Google), who wrote [Section 6.2 \(Automatic differentiation\)](#).
- Marco Cuturi (Apple, work done at Google), who wrote [Section 6.10 \(Optimal Transport\)](#).
- Jeff Bilmes (U. Washington), who wrote [Section 6.11 \(Submodular optimization\)](#).

- 1 • Justin Gilmer (Google), who wrote [Section 20.8 \(Adversarial examples\)](#).  
2 • Andrew Wilson (NYU), who helped write [Chapter 17 \(Bayesian neural networks\)](#) and [Chapter 18](#)  
3 ([Gaussian processes](#)).  
4 • George Papamakarios (Deepmind) and Balaji Lakshminarayanan (Google), who wrote [Chapter 24](#)  
5 ([Normalizing Flows](#)).  
6 • Yang Song (Stanford) and Durk Kingma (Google), who helped write [Chapter 25 \(Energy-based](#)  
7 [models\)](#).  
8 • Mihaela Rosca (Deepmind / UCL), Shakir Mohamed (Deepmind) and Balaji Lakshminarayanan  
9 (Google), who wrote [Chapter 27 \(Generative adversarial networks\)](#).  
10 • Vinayak Rao (Purdue), who wrote [Chapter 33 \(Non-parametric Bayesian models\)](#).  
11 • Ben Poole (Google) and Simon Kornblith (Google), who wrote [Chapter 34 \(Representation learning](#)  
12 ([Unfinished](#)) .  
13 • Been Kim (Google) and Finale Doshi-Velez (Harvard), who wrote [Chapter 35 \(Interpretability\)](#).  
14 • Lihong Li (Amazon, work done at Google), who helped write [Section 36.4 \(Contextual bandits\)](#)  
15 and [Chapter 37 \(Reinforcement learning\)](#).  
16 • Victor Veitch (Google / U. Chicago) and Alexander D'Amour (Google), who wrote [Chapter 38](#)  
17 ([Causality](#)).  
18

19

## 20 Acknowledgements

21 I would like to thank the following people who helped in various ways:  
22

- 23
- 24 • Mahmoud Soliman, for help with many issues related to latex, python, GCP, TPUs, etc.  
25 • Aleyna Kara, for help with many figures and software examples.  
26 • Gerardo Duran-Martin for help with many software examples.  
27 • Participants in the Google Summer of Code for 2021 and 2022.  
28 • Numerous people who proofread parts of the book, including: Kay Brodersen (Section 19.3.5),  
29 Krzysztof Choromanski (Chapter 6, Chapter 11), John Kearns (an earlier version of the the whole  
30 book), Lehman Pavasovic Krunoslav (Chapter 10, Chapter 12), Amir Globerson (Chapter 4),  
31 Ravin Kumar (Chapter 2), Junpeng Lao (Chapter 13, Chapter 30, Chapter 31), Scott Linderman  
32 (Section 31.4.1), Stephen Mandt (Chapter 22), Simon Prince (numerous chapters), Hal Varian  
33 (Section 19.3.5), Chris Williams (numerous chapters), Raymond Yeh (numerous chapters), et al.  
34 • Numerous people who contributed figures (acknowledged in the captions).  
35 • Numerous people who made their open source code available (acknowledged in the online code).

36

37

## 38 About the cover

39

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

42 Kevin Patrick Murphy

43 Palo Alto, California

44 March 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
1	Bernoulli	1	$x \in \{0, 1\}$	
2	Binomial	$N$	$x \in \{0, 1, \dots, N\}$	
3	Categorical	1	$x \in \{1, \dots, K\}$	
4	Multinomial	$N$	$\mathbf{x} \in \{0, 1, \dots, N\}^K$	$\sum_{k=1}^K x_k = N$

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

### 2.2.1.2 Categorical and multinomial distributions

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

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

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

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

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

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

See Table 2.1 for a summary of the notation.<sup>1</sup>

### 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  $\lambda$  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.

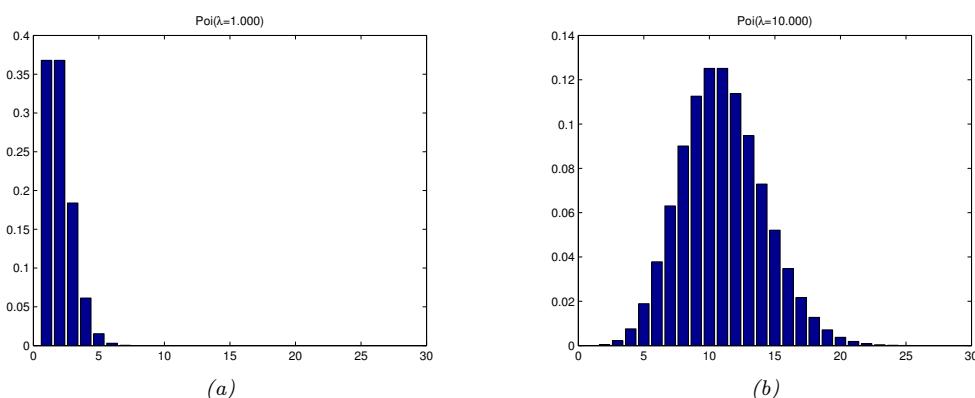


Figure 2.1: Illustration of some Poisson distributions for  $\lambda = 1$  (left) and  $\lambda = 10$  (right). We have truncated the  $x$ -axis to 30 for clarity, but the support of the distribution is over all the non-negative integers. Generated by [poisson\\_dist\\_plot.py](#).

#### 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  $\mu$  encodes the mean of the distribution, which is the same as the mode, since the distribution is unimodal. The parameters  $\sigma^2$  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  $\mu$ .

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  $\Phi(x)$ .

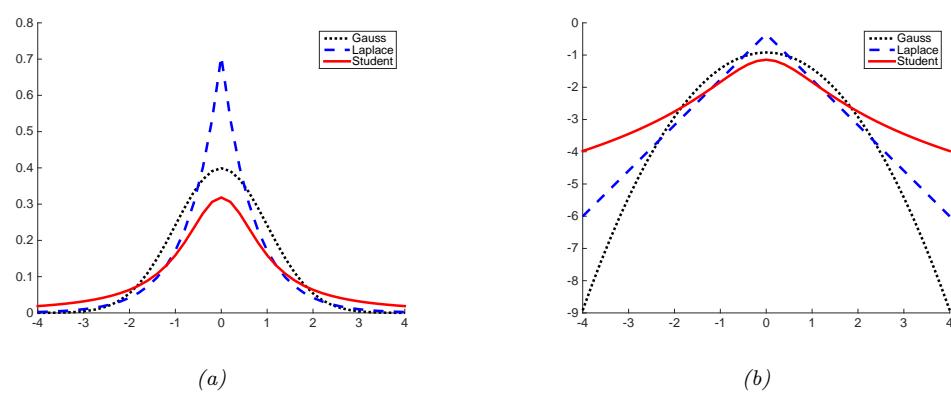


Figure 2.2: (a) The pdf's for a  $\mathcal{N}(0, 1)$ ,  $T_1(0, 1)$  and  $\text{Lap}(0, 1/\sqrt{2})$ . The mean is 0 and the variance is 1 for both the Gaussian and Laplace. The mean and variance of the Student distribution is undefined when  $\nu = 1$ . (b) Log of these pdf's. Note that the Student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

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

### 2.2.2.3 Student distribution

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

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

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

where  $\mu$  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**<sup>2</sup>, since large values of  $\nu$  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  $\mu$  is a location parameter and  $b > 0$  is a scale parameter. See Figure 2.2 for a plot. This distribution has the following properties:

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

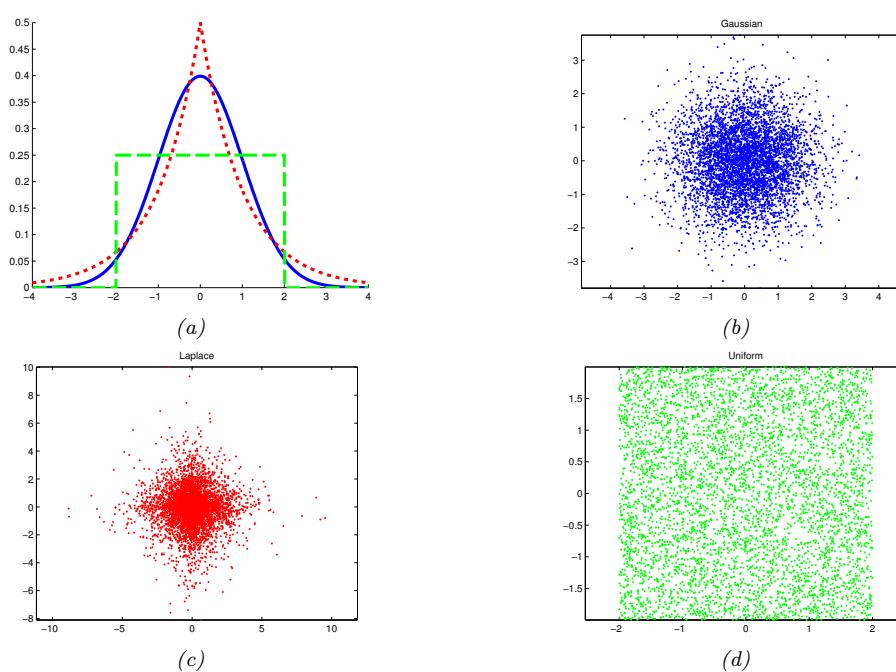


Figure 2.3: Illustration of Gaussian (blue), sub-Gaussian (uniform, green) and super-Gaussian (Laplace, red) distributions in 1d and 2d. Generated by [sub\\_super\\_gauss\\_plot.py](#).

### 2.2.2.6 Sub-Gaussian and super-Gaussian distributions

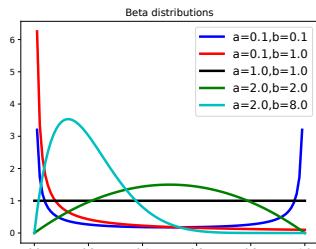
There are two main variants of the Gaussian distribution, known as **super-Gaussian** or **leptokurtic** (“Lepto” is Greek for “narrow”) and **sub-Gaussian** or **platykurtic** (“Platy” is Greek for “broad”). These distributions differ in terms of their **kurtosis**, which is a measure of how heavy or 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

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

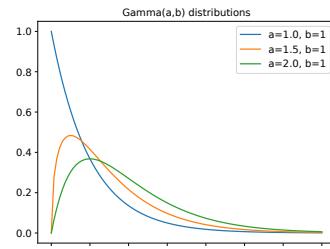
where  $\sigma$  is the standard deviation, and  $\mu_4$  is the 4'th **central moment**. (Thus  $\mu_1 = \mu$  is the mean, and  $\mu_2 = \sigma^2$  is the variance.) For a standard Gaussian, the kurtosis is 3, so some authors define the **excess kurtosis** as the kurtosis minus 3.

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.

<sup>2</sup>. This term is from [Kru13].



(a)



(b)

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

### 2.2.2.7 Beta distribution

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

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

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

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

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

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

See Figure 2.4a for plots of some beta distributions.

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

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

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

### 2.2.2.8 Gamma distribution

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

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

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

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

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

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

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

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

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

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

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

- **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  $\nu$  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  $a > 2$ .

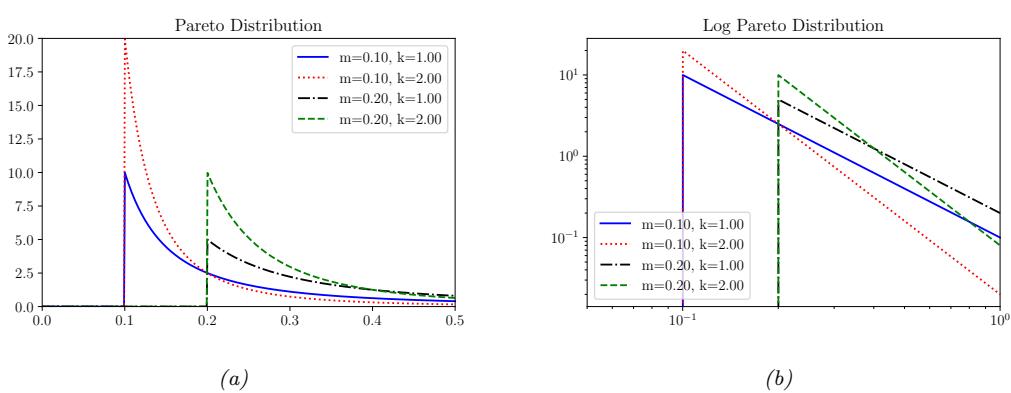


Figure 2.5: (a) The Pareto pdf  $\text{Pareto}(x|k, m)$ . (b) Same distribution on a log-log plot. Generated by `pareto_dist_plot.py`.

- 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.

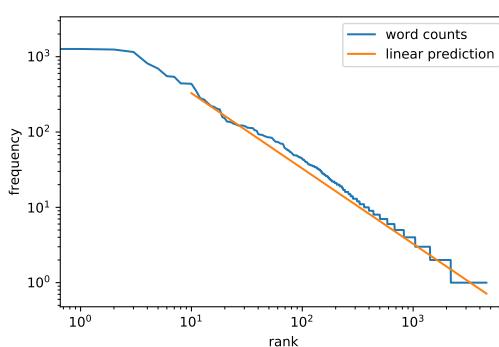


Figure 2.6: A log-log plot of the frequency vs the rank for the words in H. G. Wells’ The Time Machine. Generated by `zipfs_law_plot.py`. Adapted from a figure from [Zha+20a, Sec 8.3].

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  $\kappa$  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.<sup>3</sup>

### 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  $\kappa$  and  $a$  are constants. If we set  $a = 1$ , we recover Zipf’s law.<sup>4</sup> 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,  $p(x = 2) = \kappa 2^{-1} = 2\kappa 4^{-1} = 2p(x = 4)$ .

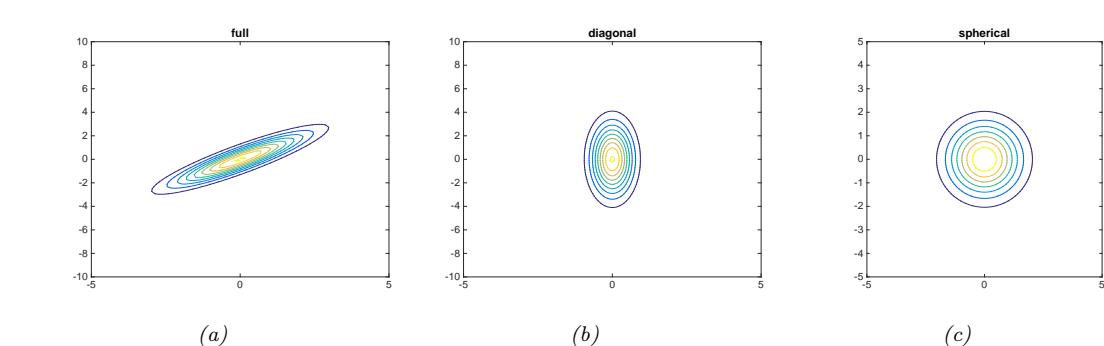


Figure 2.7: Visualization of a 2d Gaussian density in terms of level sets of constant probability density. (a) A full covariance matrix has elliptical contours. (b) A diagonal covariance matrix is an **axis aligned** ellipse. (c) A spherical covariance matrix has a circular shape. Generated by [gauss\\_plot\\_2d.py](#).

log rank, we will get a straight line with slope  $-1$ . This is in fact true, as illustrated in Figure 2.6.<sup>5</sup> 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

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

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

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

Figure 2.7 plot some MVN densities in 2d for three different kinds of covariance matrices. A **full covariance matrix** has  $D(D + 1)/2$  parameters, where we divide by 2 since  $\Sigma$  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  $\sigma^2$ .

### 2.3.2 Moment form and canonical form

It is common to parameterize the MVN in terms of the mean vector  $\mu$  and the covariance matrix  $\Sigma$ . 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  $\xi$  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  $\mu$  and  $\Sigma$ .

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

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

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

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

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

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

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

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

### 2.3.4 Bayes' rule for Gaussians

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

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

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

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

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

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

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

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

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

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

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

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

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

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

### 2.3.5 Example: sensor fusion with known measurement noise

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

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

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

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

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

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

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

### 2.3.6 Handling missing data

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

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

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

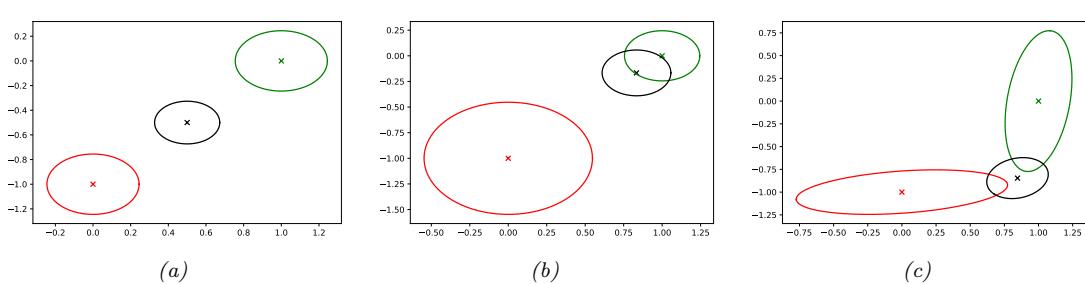


Figure 2.8: We observe  $\mathbf{x} = (0, -1)$  (red cross) and  $\mathbf{y} = (1, 0)$  (green cross) and estimate  $\mathbb{E}[z|\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}]$  (black cross). (a) Equally reliable sensors, so the posterior mean estimate is in between the two circles. (b) Sensor 2 is more 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  $p(z|y_1)$  by partitioning the joint into  $p(z, y_1, y_2)$ , marginalizing out  $y_2$ , and then conditioning on  $y_1$ . The result is as follows:

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

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

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

### 2.3.7 A calculus for linear Gaussian models

It is quite common to define large multivariate probability distributions in terms of local linear-Gaussian conditional distributions. For example, in Section 31.2, we discuss linear dynamical systems, which are linear-Gaussian models derived for time series analysis. To perform inference in such models, we can use the Kalman filter and Kalman smoother, which we discuss in Section 84.1. However, the derivation of these algorithms requires a lot of algebra. Here we present an abstract calculus for inference in linear-Gaussian systems.

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

### 2.3.7.1 Gaussian potentials

A Gaussian distribution has the following form:

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

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

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

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

where

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

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

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

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

#### 2.3.7.2 Converting a conditional linear-Gaussian distribution to a potential

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

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

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

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

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

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

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

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

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

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

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

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

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

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

### 2.3.7.3 Marginalization

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

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

where

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

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

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

### 2.3.7.4 Conditioning

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

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

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

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

### 2.3.7.5 Multiplication and division

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

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

Division is defined in an analogous way:

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

1

### 2.3.7.6 Product of Gaussians

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

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

8 where

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

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

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

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

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

20 In the scalar case, this becomes

21

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

25 **2.4 Some other multivariate continuous distributions**

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

29 **2.4.1 Multivariate Student distribution**

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

34

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

37

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

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

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

43

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

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

---

### 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  $\ell_2$ -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  $\ell_2$ -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  $\nu$  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  $\nu$  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.

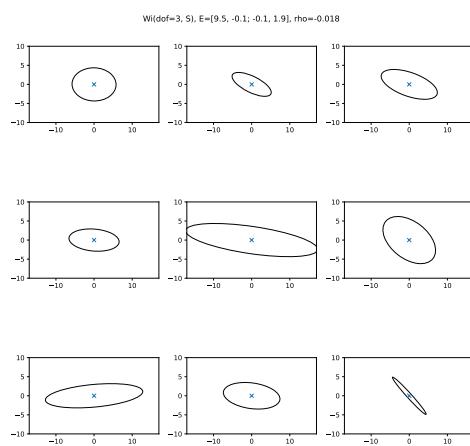


Figure 2.9: Some samples from the Wishart distribution,  $\Sigma \sim \text{Wi}(\mathbf{S}, \nu)$ , where  $\mathbf{S} = [3.1653, -0.0262; -0.0262, 0.6477]$ , and  $\nu = 3$ . Generated by [wishlist\\_plot.py](#).

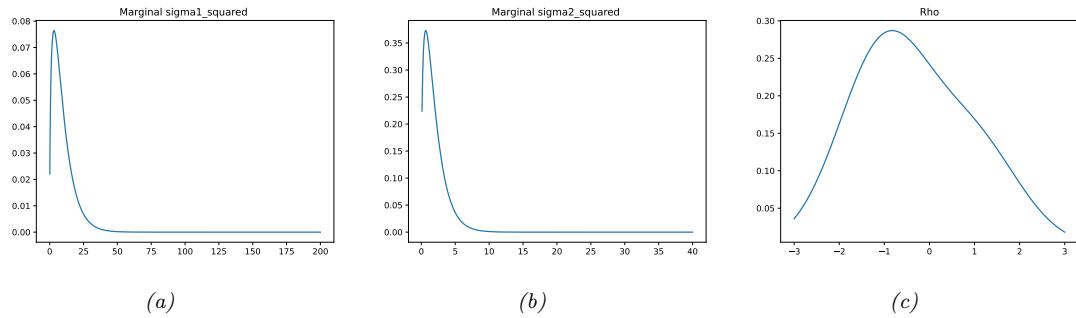


Figure 2.10: Marginals of the Wishart distribution in Figure 2.9. (a)  $p(\Sigma_{11})$ . (b)  $p(\Sigma_{22})$ . (c)  $p(\rho)$ , where  $\rho = \frac{\Sigma_{12}}{\sqrt{\Sigma_{11}\Sigma_{22}}}$  is the correlation coefficient. Generated by [wishlist\\_plot.py](#).

#### 2.4.4.2 Inverse Wishart distribution

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

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

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

One can show that the distribution has these properties

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

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

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

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

### 2.4.5 Dirichlet distribution

A multivariate generalization of the beta distribution is the **Dirichlet**<sup>6</sup> 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  $\alpha_k$  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.

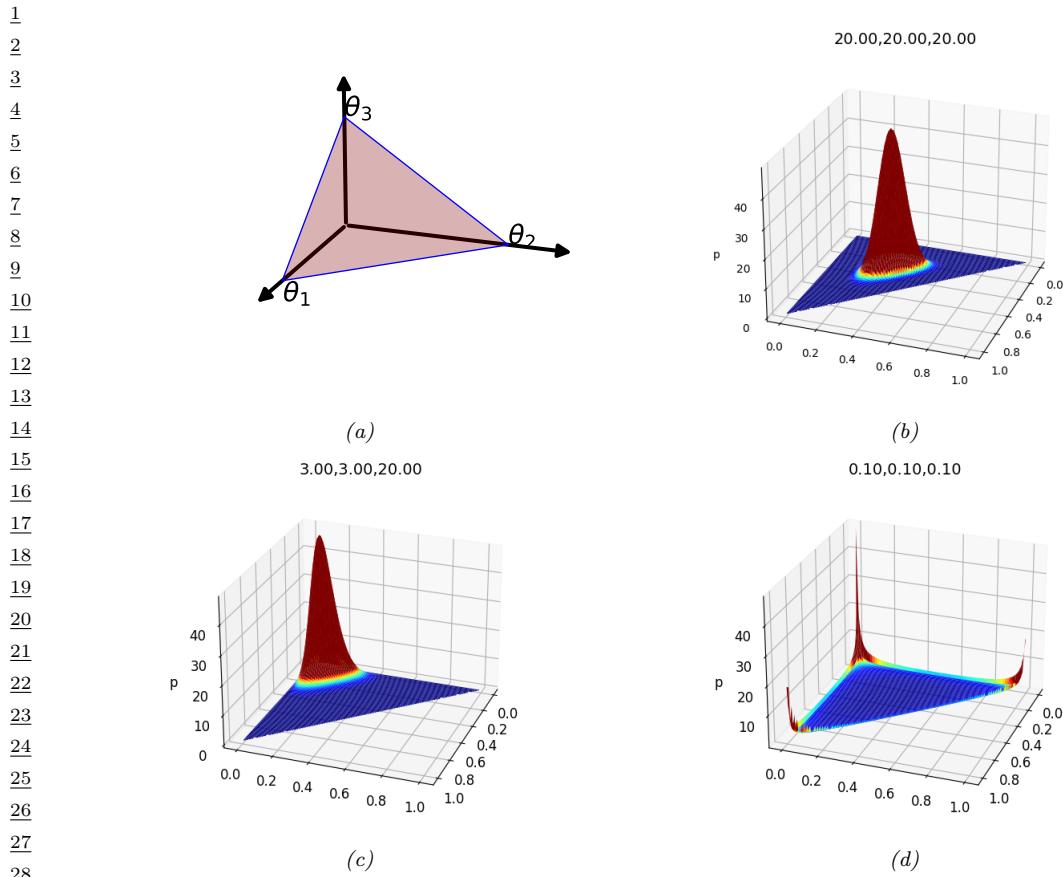
For future reference, here are some useful properties of the Dirichlet distribution:

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

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  $\alpha$  increases the precision (decreases the variance) of the distribution.

<sup>6</sup> Johann Dirichlet was a German mathematician, 1805–1859.



29 *Figure 2.11: (a) The Dirichlet distribution when  $K = 3$  defines a distribution over the simplex, which can be  
30 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  
31 by [dirichlet\\_3d\\_triangle\\_plot.py](#). (b) Plot of the Dirichlet density for  $\alpha = (20, 20, 20)$ . (c) Plot of the  
32 Dirichlet density for  $\alpha = (3, 3, 20)$ . (d) Plot of the Dirichlet density for  $\alpha = (0.1, 0.1, 0.1)$ . Generated by  
33 [dirichlet\\_3d\\_spiky\\_plot.py](#).*

34

35

## 36 2.5 The exponential family

37

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

41

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

46

47

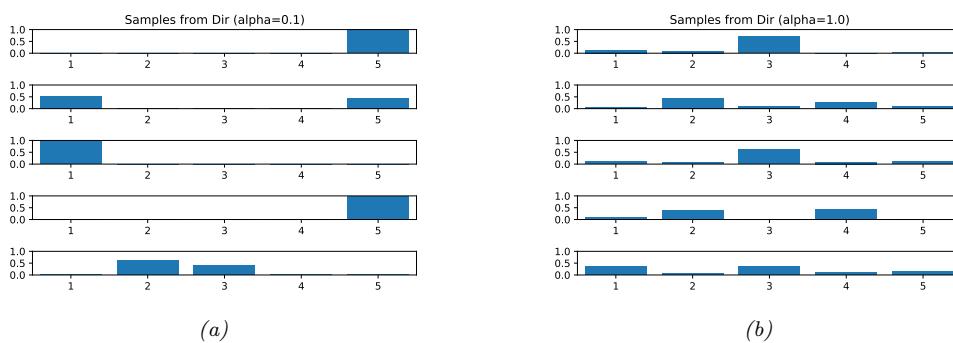


Figure 2.12: Samples from a 5-dimensional symmetric Dirichlet distribution for different parameter values. (a)  $\alpha = (0.1, \dots, 0.1)$ . This results in very sparse distributions, with many 0s. (b)  $\alpha = (1, \dots, 1)$ . This results in more uniform (and dense) distributions. Generated by `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**,  $\eta$  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 convex 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  $\phi$  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  $\phi$  to  $\eta$  is nonlinear, we call this a **curved exponential family**. If  $\eta = f(\phi) = \phi$ , the model is said to be in **canonical form**. If, in addition,  $\mathcal{T}(x) = x$ , we say this is a **natural exponential family** or **NEF**. In this case, it can be written as

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

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

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

We will see some examples below.

### 2.5.2 Examples

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

#### 2.5.2.1 Bernoulli distribution

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

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

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

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

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

We can see this as follows:

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

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

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

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

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

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

We can recover the mean parameter  $\mu$  from the canonical parameter  $\eta$  using

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

which we recognize as the logistic (sigmoid) function.

### 2.5.2.2 Categorical distribution

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

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

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

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

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

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

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

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

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

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

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

We can recover the mean parameters from the canonical parameters using

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

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

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

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

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

1  
2 and hence  
3

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

### 8 2.5.2.3 Univariate Gaussian

9 The univariate Gaussian is usually written as follows:  
10

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

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

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

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

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

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

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

29 The moment parameters are  
30

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

### 34 2.5.2.4 Univariate Gaussian with fixed variance

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

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

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

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

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

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

---

### 2.5.2.5 Multivariate Gaussian

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

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

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

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

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

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

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

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

We can convert this to exponential family notation as follows:

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

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

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

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

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

where

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

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

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

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

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

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

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

### 2.5.2.6 Non-examples

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

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

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

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

### 2.5.3 Log partition function is cumulant generating function

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

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

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

We prove this result below.

---

### 2.5.3.1 Derivation of the mean

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

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

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

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

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

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

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

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

### 2.5.3.2 Derivation of the variance

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

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

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

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

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

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

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

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

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

---

1    **2.5.3.3 Connection with the Fisher information matrix**

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

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

6    Hence for an exponential family model we have

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

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

10    **2.5.4 Canonical (natural) vs mean (moment) parameters**

11    Let  $\Omega$  be the set of normalizable natural parameters:

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

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

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

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

18    The set of valid moment parameters is given by

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

20    for some distribution  $p$ .

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

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

23    If the family is minimal, one can show that

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

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

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

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

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

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

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

33    (2.202)

---

### 2.5.5 MLE for the exponential family

The likelihood of an exponential family model has the form

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

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

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

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

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

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

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

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

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:

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

For a single data case, this becomes

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

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

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

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

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

---

### 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  $\sigma^2$  is called the **dispersion parameter**. For fixed  $\sigma^2$ , this is a natural exponential family.

### 2.5.7 Maximum entropy derivation of the exponential family

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

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

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

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

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

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

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

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

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

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

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

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

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

Hence the normalization constant is given by

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

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

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

## 2.6 Fisher information matrix (FIM)

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

### 2.6.1 Definition

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

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

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

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

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

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

We give an interpretation of this quantity below.

### 2.6.2 Equivalence between the FIM and the Hessian of the NLL

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

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

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

More precisely, we have the following theorem.

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

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

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

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

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

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

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

Combining this with the identity

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

we have

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

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

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

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

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

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

and hence

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

as claimed.  $\square$

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

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

From the above theorem, we have

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

We will use this result later in this chapter.

---

### 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

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

#### 2.6.3.2 FIM for the Gaussian

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

$$\ell(\boldsymbol{\theta}) = \log p(x|\boldsymbol{\theta}) = -\frac{1}{2v}(x-\mu)^2 - \frac{1}{2}\log(v) - \frac{1}{2}\log(2\pi) \quad (2.237)$$

The partial derivatives are given by

$$\frac{\partial \ell}{\partial \mu} = (x-\mu)v^{-1}, \quad \frac{\partial^2 \ell}{\partial \mu^2} = -v^{-1} \quad (2.238)$$

$$\frac{\partial \ell}{\partial v} = \frac{1}{2}v^{-2}(x-\mu)^2 - \frac{1}{2}v^{-1}, \quad \frac{\partial \ell}{\partial v^2} = -v^{-3}(x-\mu)^2 + \frac{1}{2}v^{-2} \quad (2.239)$$

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

and hence

$$\mathbf{F}(\boldsymbol{\theta}) = \begin{pmatrix} \mathbb{E}[v^{-1}] & \mathbb{E}[v^{-2}(x-\mu)] \\ \mathbb{E}[v^{-2}(x-\mu)] & \mathbb{E}[v^{-3}(x-\mu)^2 - \frac{1}{2}v^{-2}] \end{pmatrix} = \begin{pmatrix} \frac{1}{v} & 0 \\ 0 & \frac{1}{2v^2} \end{pmatrix} \quad (2.241)$$

#### 2.6.3.3 FIM for logistic regression

Consider  $\ell_2$ -regularized binary logistic regression. The negative log joint has the following form:

$$\mathcal{E}(\mathbf{w}) = -\log[p(\mathbf{y}|\mathbf{X}, \mathbf{w})p(\mathbf{w}|\lambda)] = -\mathbf{w}^\top \mathbf{X}^\top \mathbf{y} + \sum_{n=1}^N \log(1 + e^{\mathbf{w}^\top \mathbf{x}_n}) + \frac{\lambda}{2} \mathbf{w}^\top \mathbf{w} \quad (2.242)$$

The derivative has the form

$$\nabla_{\mathbf{w}} \mathcal{E}(\mathbf{w}) = -\mathbf{X}^T \mathbf{y} + \mathbf{X}^T \mathbf{s} + \lambda \mathbf{w} \quad (2.243)$$

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

$$\mathbf{F}(\mathbf{w}) = \mathbb{E}_{p(\mathbf{y}|\mathbf{X}, \mathbf{w}, \lambda)} [\nabla^2 \mathcal{E}(\mathbf{w})] = \mathbf{X}^T \mathbf{\Lambda} \mathbf{X} + \lambda \mathbf{I} \quad (2.244)$$

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

$$\Lambda_{nn} = \sigma(\mathbf{w}^T \mathbf{x}_n)(1 - \sigma(\mathbf{w}^T \mathbf{x}_n)) \quad (2.245)$$

#### 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:

$$D_{\text{KL}}(p_{\boldsymbol{\theta}} \| p_{\boldsymbol{\theta}'}) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x})} [\log p_{\boldsymbol{\theta}}(\mathbf{x}) - \log p_{\boldsymbol{\theta}'}(\mathbf{x})] \quad (2.246)$$

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

$$D_{\text{KL}}(p_{\boldsymbol{\theta}} \| p_{\boldsymbol{\theta}'}) \approx -\boldsymbol{\delta}^T \mathbb{E}[\nabla \log p_{\boldsymbol{\theta}}(\mathbf{x})] - \frac{1}{2} \boldsymbol{\delta}^T \mathbb{E}[\nabla^2 \log p_{\boldsymbol{\theta}}(\mathbf{x})] \boldsymbol{\delta} \quad (2.247)$$

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

$$D_{\text{KL}}(p_{\boldsymbol{\theta}} \| p_{\boldsymbol{\theta}'}) \approx \frac{1}{2} \boldsymbol{\delta}^T \mathbf{F}(\boldsymbol{\theta}) \boldsymbol{\delta} \quad (2.248)$$

where  $\mathbf{F}$  is the FIM

$$\mathbf{F} = -\mathbb{E}[\nabla^2 \log p_{\boldsymbol{\theta}}(\mathbf{x})] = \mathbb{E}[(\nabla \log p_{\boldsymbol{\theta}}(\mathbf{x}))(\nabla \log p_{\boldsymbol{\theta}}(\mathbf{x}))^T] \quad (2.249)$$

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

$$\nabla_{\boldsymbol{\eta}} A(\boldsymbol{\eta}) = \mathbb{E}[\mathcal{T}(\mathbf{x})] = \mathbf{m} \quad (2.250)$$

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

$$\nabla_{\boldsymbol{\eta}} \log p(\mathbf{x} | \boldsymbol{\eta}) = \mathcal{T}(\mathbf{x}) - \mathbb{E}[\mathcal{T}(\mathbf{x})] \quad (2.251)$$

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

$$(\mathbf{F}_\eta)_{ij} = \mathbb{E}_{p(\mathbf{x}|\boldsymbol{\eta})} \left[ \frac{\partial \log p(\mathbf{x}|\boldsymbol{\eta})}{\partial \eta_i} \frac{\partial \log p(\mathbf{x}|\boldsymbol{\eta})}{\partial \eta_j} \right] \quad (2.252)$$

$$= \mathbb{E}_{p(\mathbf{x}|\boldsymbol{\eta})} [(\mathcal{T}(\mathbf{x})_i - m_i)(\mathcal{T}(\mathbf{x})_j - m_j)] \quad (2.253)$$

$$= \text{Cov} [\mathcal{T}(\mathbf{x})_i, \mathcal{T}(\mathbf{x})_j] \quad (2.254)$$

or, in short,

$$\mathbf{F}_\eta = \text{Cov} [\mathcal{T}(\mathbf{x})] \quad (2.255)$$

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

$$(\mathbf{F}_m)_{ij} = \mathbb{E}_{p(\mathbf{x}|\mathbf{m})} \left[ \frac{\partial \log p(\mathbf{x}|\boldsymbol{\eta})}{\partial m_i} \frac{\partial \log p(\mathbf{x}|\boldsymbol{\eta})}{\partial m_j} \right] \quad (2.256)$$

From the chain rule we have

$$\frac{\partial \log p(x)}{\partial \alpha} = \frac{\partial \log p(x)}{\partial \beta} \frac{\partial \beta}{\partial \alpha} \quad (2.257)$$

and hence

$$\mathbf{F}_\alpha = \frac{\partial \boldsymbol{\beta}^\top}{\partial \alpha} \mathbf{F}_\beta \frac{\partial \boldsymbol{\beta}}{\partial \alpha} \quad (2.258)$$

Using the log trick

$$\nabla \mathbb{E}_{p(\mathbf{x})} [f(\mathbf{x})] = \mathbb{E}_{p(\mathbf{x})} [f(\mathbf{x}) \nabla \log p(\mathbf{x})] \quad (2.259)$$

and Equation (2.251) we have

$$\frac{\partial m_i}{\partial \eta_j} = \frac{\partial \mathbb{E} [\mathcal{T}(\mathbf{x})_i]}{\partial \eta_j} = \mathbb{E} \left[ \mathcal{T}(\mathbf{x})_i \frac{\partial \log p(\mathbf{x}|\boldsymbol{\eta})}{\partial \eta_j} \right] = \mathbb{E} [\mathcal{T}(\mathbf{x})_i (\mathcal{T}(\mathbf{x})_j - m_j)] \quad (2.260)$$

$$= \mathbb{E} [\mathcal{T}(\mathbf{x})_i \mathcal{T}(\mathbf{x})_j] - \mathbb{E} [\mathcal{T}(\mathbf{x})_i] m_j = \text{Cov} [\mathcal{T}(\mathbf{x})_i \mathcal{T}(\mathbf{x})_j] = (\mathbf{F}_\eta)_{ij} \quad (2.261)$$

and hence

$$\frac{\partial \boldsymbol{\eta}}{\partial \mathbf{m}} = \mathbf{F}_\eta^{-1} \quad (2.262)$$

so

$$\mathbf{F}_m = \frac{\partial \boldsymbol{\eta}^\top}{\partial \mathbf{m}} \mathbf{F}_\eta \frac{\partial \boldsymbol{\eta}}{\partial \mathbf{m}} = \mathbf{F}_\eta^{-1} \mathbf{F}_\eta \mathbf{F}_\eta^{-1} = \mathbf{F}_\eta^{-1} = \text{Cov} [\mathcal{T}(\mathbf{x})]^{-1} \quad (2.263)$$

## 2.7 Transformations of random variables

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

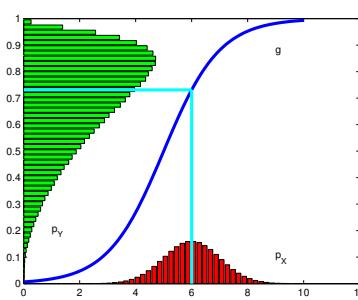


Figure 2.13: Example of the transformation of a density under a nonlinear transform. Note how the mode of the transformed distribution is not the transform of the original mode. Adapted from Exercise 1.4 of [Bis06]. Generated by [bayes\\_change\\_of\\_var.py](#).

### 2.7.1 Invertible transformations (bijections)

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

$$p_y(\mathbf{y}) = p_x(g(\mathbf{y})) \left| \det [\mathbf{J}(g)(\mathbf{y})] \right| \quad (2.264)$$

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 determinant of  $\mathbf{J}$ .

### 2.7.2 Monte Carlo approximation

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

$$p_{\mathcal{D}}(\mathbf{y}) = \frac{1}{S} \sum_{s=1}^S \delta(\mathbf{y} - \mathbf{y}^s) \quad (2.265)$$

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

### 2.7.3 Probability integral transform

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

$$P_Y(y) = \Pr(Y \leq y) = \Pr(P_X(X) \leq y) \quad (2.266)$$

$$= \Pr(X \leq P_X^{-1}(y)) = P_X(P_X^{-1}(y)) = y \quad (2.267)$$

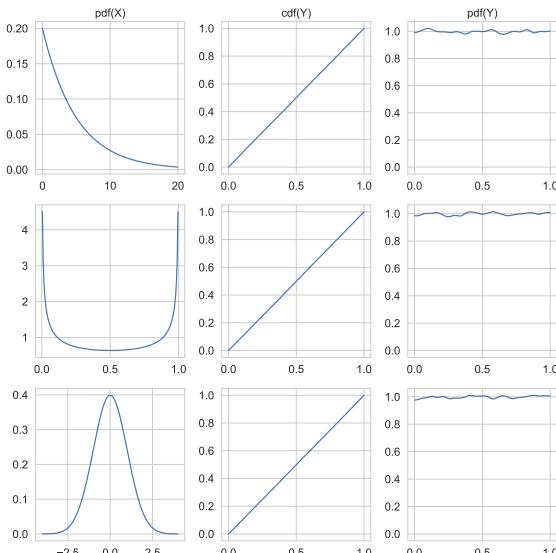


Figure 2.14: Illustration of the probability integral transform. Left column: 3 different pdf's for  $p(X)$  from which we sample  $x_n \sim p(x)$ . Middle column: empirical cdf of  $y_n = P_X(x_n)$ . Right column: empirical pdf of  $p(y_n)$  using a kernel density estimate. Adapted from Figure 11.17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

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

$$D_n = \max_x |P_n(x) - P(x)| \quad (2.268)$$

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  $x_n = P_X^{-1}(u_n)$ .

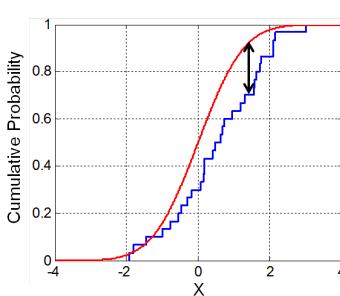


Figure 2.15: Illustration of the Kolmogorov–Smirnov statistic. The red line is a model CDF, the blue line is an empirical CDF, and the black arrow is the K–S statistic. From [https://en.wikipedia.org/wiki/Kolmogorov\\_Smirnov\\_test](https://en.wikipedia.org/wiki/Kolmogorov_Smirnov_test). Used with kind 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.,

$$p(\mathbf{x}_{t+\tau}|\mathbf{x}_t, \mathbf{x}_{1:t-1}) = p(\mathbf{x}_{t+\tau}|\mathbf{x}_t) \quad (2.269)$$

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:

$$p(\mathbf{x}_{1:T}) = p(\mathbf{x}_1)p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_3|\mathbf{x}_2)p(\mathbf{x}_4|\mathbf{x}_3)\dots = p(\mathbf{x}_1) \prod_{t=2}^T p(\mathbf{x}_t|\mathbf{x}_{t-1}) \quad (2.270)$$

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

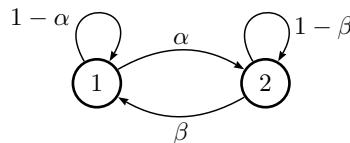
### 2.8.1 Parameterization

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

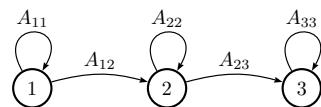
#### 2.8.1.1 Markov transition kernels

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

$$\mathbf{A} = \begin{pmatrix} 1 - \alpha & \alpha \\ \beta & 1 - \beta \end{pmatrix} \quad (2.271)$$

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

$$\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} & 0 \\ 0 & A_{22} & A_{23} \\ 0 & 0 & 1 \end{pmatrix} \quad (2.272)$$

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

$$A_{ij}(n) \triangleq p(X_{t+n} = j | X_t = i) \quad (2.273)$$

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

$$A_{ij}(m+n) = \sum_{k=1}^K A_{ik}(m) A_{kj}(n) \quad (2.274)$$

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

$$\mathbf{A}(m+n) = \mathbf{A}(m)\mathbf{A}(n) \quad (2.275)$$

Hence

$$\mathbf{A}(n) = \mathbf{A} \mathbf{A}(n-1) = \mathbf{A} \mathbf{A} \mathbf{A}(n-2) = \cdots = \mathbf{A}^n \quad (2.276)$$

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)  $n$ :

$$p(\mathbf{x}_{1:T}) = p(\mathbf{x}_{1:n}) \prod_{t=n+1}^T p(\mathbf{x}_t | \mathbf{x}_{t-n:t-1}) \quad (2.277)$$

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**.

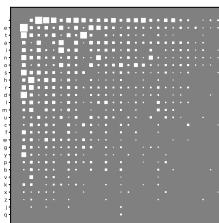
Note, however, we can always convert a higher order Markov model to a first order one by defining an augmented state space that contains the past  $n$  observations. For example, if  $n = 2$ , we define  $\tilde{\mathbf{x}}_t = (\mathbf{x}_{t-1}, \mathbf{x}_t)$  and use

$$p(\tilde{\mathbf{x}}_{1:T}) = p(\tilde{\mathbf{x}}_2) \prod_{t=3}^T p(\tilde{\mathbf{x}}_t | \tilde{\mathbf{x}}_{t-1}) = p(\mathbf{x}_1, \mathbf{x}_2) \prod_{t=3}^T p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{x}_{t-2}) \quad (2.278)$$

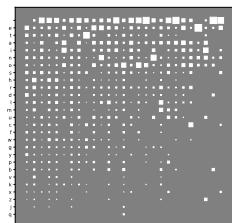
Therefore we will just focus on first-order models throughout the rest of this section.

### 2.8.2 Application: Language modeling

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



(a)



(b)

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

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

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

### 2.8.3 Parameter estimation

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

### 2.8.3.1 Maximum likelihood estimation

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

$$p(x_{1:T} | \boldsymbol{\theta}) = \pi(x_1) A(x_1, x_2) \dots A(x_{T-1}, x_T) \quad (2.279)$$

$$= \prod_{j=1}^K (\pi_j)^{\mathbb{I}(x_1=j)} \prod_{t=2}^T \prod_{i=1}^K \prod_{k=1}^K (A_{jk})^{\mathbb{I}(x_t=k, x_{t-1}=j)} \quad (2.280)$$

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

$$\log p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{i=1}^N \log p(\mathbf{x}_i|\boldsymbol{\theta}) = \sum_i N_j^1 \log \pi_j + \sum_i \sum_k N_{jk} \log A_{jk} \quad (2.281)$$

where we define the following counts:

$$N_j^1 \triangleq \sum_{i=1}^N \mathbb{I}(x_{i1} = j), \quad N_{jk} \triangleq \sum_{i=1}^N \sum_{t=1}^{T_i-1} \mathbb{I}(x_{i,t} = j, x_{i,t+1} = k), \quad N_j = \sum_k N_{jk} \quad (2.282)$$

Hence we can write the MLE as the normalized counts:

$$\hat{\pi}_j = \frac{N_j^1}{\sum_{j'} N_{j'}^1}, \quad \hat{A}_{jk} = \frac{N_{jk}}{N_j} \quad (2.283)$$

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

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

### 2.8.3.2 Sparse data problem

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

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.<sup>8</sup> Although such an approach can be surprisingly successful (as discussed in [HNP09]), it is rather unsatisfying, since humans are able to learn language from much less data (see e.g., [TX00]).

### 2.8.3.3 MAP estimation

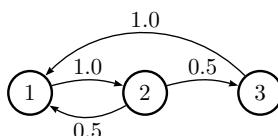
A simple solution to the sparse data problem is to use MAP estimation with a uniform Dirichlet prior,  $\mathbf{A}_{j:} \sim \text{Dir}(\alpha \mathbf{1})$ . In this case, the MAP estimate becomes

$$\hat{A}_{jk} = \frac{N_{jk} + \alpha}{N_j + K\alpha} \quad (2.284)$$

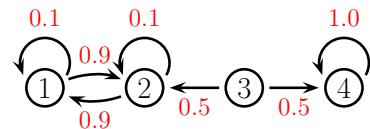
If  $\alpha = 1$ , this is called **add-one smoothing**.

<sup>7</sup> 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.

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



(a)



(b)

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

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

## 2.8.4 Stationary distribution of a Markov chain

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

### 2.8.4.1 What is a stationary distribution?

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

If we have an initial distribution over states of  $\pi_0$ , then at time 1 we have

$$\pi_1(j) = \sum_i \pi_0(i) A_{ij} \quad (2.285)$$

or, in matrix notation,  $\pi_1 = \pi_0 \mathbf{A}$ , where we have followed the standard convention of assuming  $\pi$  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

$$(\pi_1 \quad \pi_2 \quad \pi_3) = (\pi_1 \quad \pi_2 \quad \pi_3) \begin{pmatrix} 1 - A_{12} - A_{13} & A_{12} & A_{13} \\ A_{21} & 1 - A_{21} - A_{23} & A_{23} \\ A_{31} & A_{32} & 1 - A_{31} - A_{32} \end{pmatrix} \quad (2.286)$$

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

$$\pi_i \sum_{j \neq i} A_{ij} = \sum_{j \neq i} \pi_j A_{ji} \quad (2.287)$$

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

#### 2.8.4.2 Computing the stationary distribution

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

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

$$(\pi_1 \ \pi_2 \ \pi_3) \begin{pmatrix} 1 - A_{11} & -A_{12} & 1 \\ -A_{21} & 1 - A_{22} & 1 \\ -A_{31} & -A_{32} & 1 \end{pmatrix} = (0 \ 0 \ 1) \quad (2.288)$$

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

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

#### 2.8.4.3 When does a stationary distribution exist?

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

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

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

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

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

$$p(X_t = j) = \sum_i p(X_0 = i) A_{ij}(t) \rightarrow \pi_j \text{ as } t \rightarrow \infty \quad (2.289)$$

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  $\pi$ , its unique stationary distribution.*

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

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

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

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

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

**Theorem 2.8.2.** *Every irreducible, ergodic Markov chain has a limiting distribution, which is equal to  $\pi$ , 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  $\pi$  such that

$$\pi_i A_{ij} = \pi_j A_{ji} \quad (2.290)$$

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  $\pi$ , then  $\pi$  is a stationary distribution of the chain.*

*Proof.* To see this, note that

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

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].)

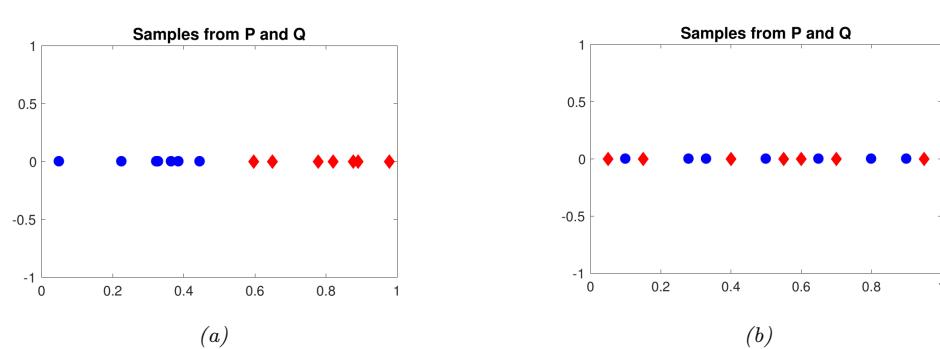


Figure 2.20: Samples from two distributions which are (a) different and (b) similar. From a figure from [GSJ19]. Used with kind permission of Arthur Getton.

### 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:

$$D_f(p||q) = \int q(\mathbf{x})f\left(\frac{p(\mathbf{x})}{q(\mathbf{x})}\right)d\mathbf{x} \quad (2.292)$$

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  $\phi$ -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:

$$D_{\text{KL}}(p||q) = \int p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x} \quad (2.293)$$

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:

$$D_\alpha^A(p||q) \triangleq \frac{4}{1-\alpha^2} \left( 1 - \int p(\mathbf{x})^{(1+\alpha)/2} q(\mathbf{x})^{(1-\alpha)/2} d\mathbf{x} \right) \quad (2.294)$$

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

$$D_\alpha^M(p||q) = \frac{1}{\alpha(1-\alpha)} \left( 1 - \int p(\mathbf{x})^\alpha q(\mathbf{x})^{1-\alpha} d\mathbf{x} \right) \quad (2.295)$$

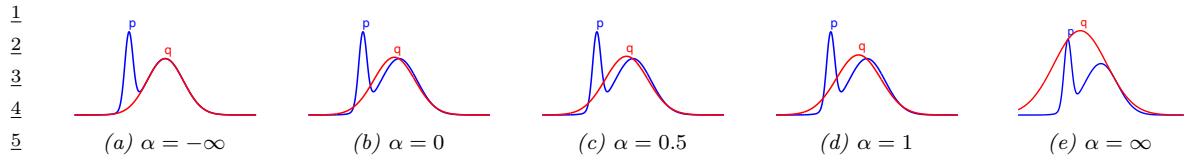


Figure 2.21: The Gaussian  $q$  which minimizes  $\alpha$ -divergence to  $p$  (a mixture of two Gaussians), for varying  $\alpha$ . From Figure 1 of [Min05]. Used with kind permission of Tom Minka.

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:

$$D_H^2(p||q) \triangleq \frac{1}{2} \int \left( p(\mathbf{x})^{\frac{1}{2}} - q(\mathbf{x})^{\frac{1}{2}} \right)^2 d\mathbf{x} = 1 - \int \sqrt{p(\mathbf{x})q(\mathbf{x})} d\mathbf{x} \quad (2.296)$$

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

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

### 2.9.1.4 Chi-squared distance

The **chi-squared distance**  $\chi^2$  is defined by

$$\chi^2(p, q) \triangleq \frac{1}{2} \int \frac{(q(\mathbf{x}) - p(\mathbf{x}))^2}{q(\mathbf{x})} d\mathbf{x} \quad (2.298)$$

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

$$\int d\mathbf{x} q(\mathbf{x}) \left( \frac{p(\mathbf{x})}{q(\mathbf{x})} - 1 \right)^2 = \int d\mathbf{x} q(\mathbf{x}) \left( \frac{p(\mathbf{x}) - q(\mathbf{x})}{q(\mathbf{x})} \right)^2 = \int d\mathbf{x} \frac{1}{q(\mathbf{x})} (p(\mathbf{x}) - q(\mathbf{x}))^2 \quad (2.299)$$

## 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:

$$D_{\mathcal{F}}(P, Q) \triangleq \sup_{f \in \mathcal{F}} |\mathbb{E}_{p(\mathbf{x})} [f(\mathbf{x})] - \mathbb{E}_{q(\mathbf{x}')} [f(\mathbf{x}')]| \quad (2.300)$$

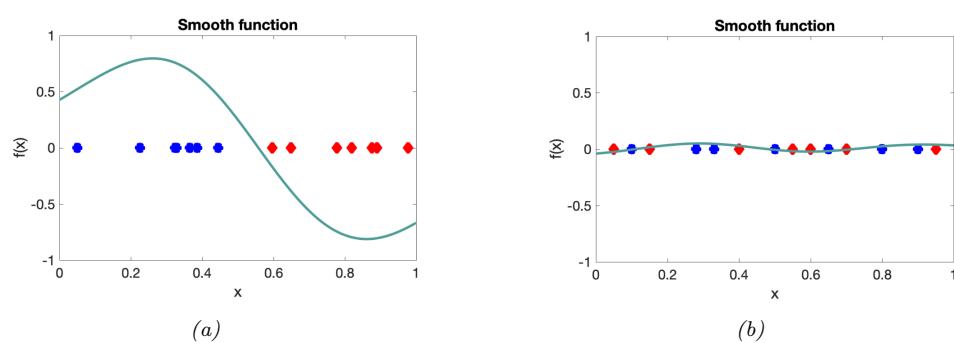


Figure 2.22: A smooth witness function for comparing two distributions which are (a) different and (b) similar. From a figure from [GSJ19]. Used with kind permission of Arthur Getton.

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

$$\|f\|_L = \sup_{\mathbf{x} \neq \mathbf{x}'} \frac{|f(\mathbf{x}) - f(\mathbf{x}')|}{\|\mathbf{x} - \mathbf{x}'\|} \quad (2.301)$$

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

$$W_1(P, Q) \triangleq \sup_{\|f\|_L \leq 1} |\mathbb{E}_{p(\mathbf{x})} [f(\mathbf{x})] - \mathbb{E}_{q(\mathbf{x}')} [f(\mathbf{x}')]| \quad (2.302)$$

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 27.4).

#### 2.9.3.1 MMD as an IPM

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

$$\text{MMD}(P, Q; \mathcal{F}) = \sup_{f \in \mathcal{F}: \|f\|_L \leq 1} [\mathbb{E}_{p(\mathbf{x})} [f(\mathbf{x})] - \mathbb{E}_{q(\mathbf{x}')} [f(\mathbf{x}')]] \quad (2.303)$$

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

$$f(\mathbf{x}) = \langle f, \phi(\mathbf{x}) \rangle_{\mathcal{F}} = \sum_{l=1}^{\infty} f_l \phi_l(\mathbf{x}) \quad (2.304)$$

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

$$\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})] = \langle f, \mathbb{E}_{p(\mathbf{x})}[\phi(\mathbf{x})] \rangle_{\mathcal{F}} = \langle f, \boldsymbol{\mu}_P \rangle_{\mathcal{F}} \quad (2.305)$$

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

$$\text{MMD}(P, Q; \mathcal{F}) = \sup_{\|f\| \leq 1} \langle f, \boldsymbol{\mu}_P - \boldsymbol{\mu}_Q \rangle_{\mathcal{F}} = \frac{\|\boldsymbol{\mu}_P - \boldsymbol{\mu}_Q\|}{\|\boldsymbol{\mu}_P - \boldsymbol{\mu}_Q\|} \quad (2.306)$$

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

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

### 2.9.3.2 Computing the MMD using the kernel trick

In this section, we describe how to compute Equation (2.306) 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

$$\text{MMD}^2(\mathcal{X}, \mathcal{X}') \triangleq \left\| \frac{1}{N} \sum_{n=1}^N \phi(\mathbf{x}_n) - \frac{1}{M} \sum_{m=1}^M \phi(\mathbf{x}'_m) \right\|^2 \quad (2.307)$$

$$\begin{aligned} &= \frac{1}{N^2} \sum_{n=1}^N \sum_{n'=1}^N \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_{n'}) - \frac{2}{NM} \sum_{n=1}^N \sum_{m=1}^M \phi(\mathbf{x}_n)^T \phi(\mathbf{x}'_m) \\ &\quad + \frac{1}{M^2} \sum_{m=1}^M \sum_{m'=1}^M \phi(\mathbf{x}'_{m'})^T \phi(\mathbf{x}'_m) \end{aligned} \quad (2.308)$$

Since Equation (2.308) only involves inner products of the feature vectors, we can use the kernel trick (Section 18.2.2) to rewrite the above as follows:

$$\text{MMD}^2(\mathcal{X}, \mathcal{X}') = \frac{1}{N^2} \sum_{n=1}^N \sum_{n'=1}^N \mathcal{K}(\mathbf{x}_n, \mathbf{x}_{n'}) - \frac{2}{NM} \sum_{n=1}^N \sum_{m=1}^M \mathcal{K}(\mathbf{x}_n, \mathbf{x}'_m) + \frac{1}{M^2} \sum_{m=1}^M \sum_{m'=1}^M \mathcal{K}(\mathbf{x}'_m, \mathbf{x}'_{m'}) \quad (2.309)$$

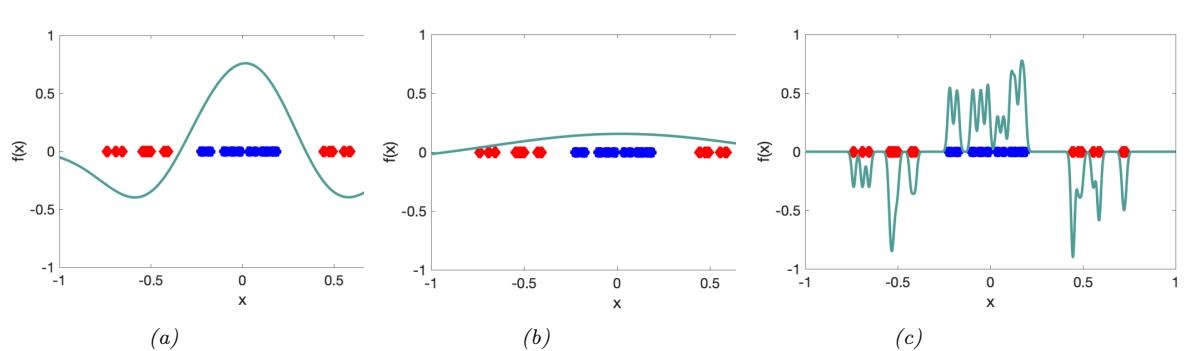


Figure 2.23: Effect of bandwidth parameter  $\sigma$  on the witness function defined by a Gaussian kernel. From a figure from [GSJ19]. Used with kind permission of Dougal Sutherland.

### 2.9.3.3 Linear time computation

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

The key idea is to notice that evaluating

$$\text{witness}^2(\mathbf{v}) = (\mu_O(\mathbf{v}) - \mu_P(\mathbf{v}))^2 \quad (2.310)$$

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:

$$\text{UME}^2(P, Q) = \frac{1}{J} \sum_{j=1}^J [\boldsymbol{\mu}_P(\mathbf{v}_j) - \boldsymbol{\mu}_Q(\mathbf{v}_j)]^2 \quad (2.311)$$

where  $\mu_B(v) = \mathbb{E}_{x \sim P} [\mathcal{K}(x, v)]$  can be estimated empirically in  $O(N)$  time, and similarly for  $\mu_O(v)$ .

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

#### 2.9.3.4 Choosing the right kernel

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

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

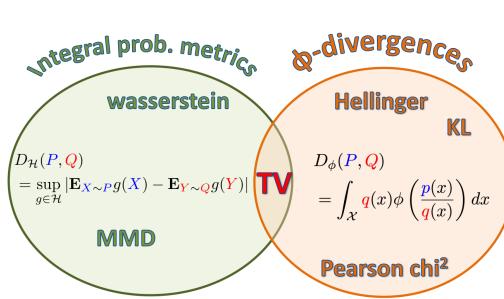


Figure 2.24: Summary of the two main kinds of divergence measures between two probability distributions  $P$  and  $Q$ . From a figure from [GSJ19]. Used with kind permission of Arthur Getton.

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

#### 2.9.4 Total variation distance

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

$$D_{\text{TV}}(p, q) \triangleq \frac{1}{2} \|p - q\|_1 = \frac{1}{2} \int |p(\mathbf{x}) - q(\mathbf{x})| d\mathbf{x} \quad (2.312)$$

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

$$\frac{1}{2} \int q(\mathbf{x}) \left| \frac{p(\mathbf{x})}{q(\mathbf{x})} - 1 \right| d\mathbf{x} = \frac{1}{2} \int q(\mathbf{x}) \left| \frac{p(\mathbf{x}) - q(\mathbf{x})}{q(\mathbf{x})} \right| d\mathbf{x} = \frac{1}{2} \int |p(\mathbf{x}) - q(\mathbf{x})| d\mathbf{x} \quad (2.313)$$

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

$$\frac{P(\mathbf{x})}{Q(\mathbf{x})} = \frac{p(\mathbf{x}|y = 1)}{p(\mathbf{x}|y = 0)} = \frac{p(y = 1|\mathbf{x})p(\mathbf{x})}{p(y = 1)} / \frac{p(y = 0|\mathbf{x})p(\mathbf{x})}{p(y = 0)} \quad (2.314)$$

$$= \frac{p(y = 1|\mathbf{x})}{p(y = 0|\mathbf{x})} \frac{1 - \pi}{\pi} \quad (2.315)$$

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

$$R(h) = \mathbb{E}_{p(\mathbf{x}|y)p(y)} [-y \log h(\mathbf{x}) - (1 - y) \log(1 - h(\mathbf{x}))] \quad (2.316)$$

$$= \pi \mathbb{E}_{P(\mathbf{x})} [-\log h(\mathbf{x})] + (1 - \pi) \mathbb{E}_{Q(\mathbf{x})} [-\log(1 - h(\mathbf{x}))] \quad (2.317)$$

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  $\ell$ , where we minimize over some function class  $\mathcal{F}$ .<sup>9</sup> In [NWJ09], they show that for every f-divergence, there is a loss function  $\ell$  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  $\chi^2$  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

$$R_{h^*} = \inf_h \int \ell(\tilde{y}, h(\mathbf{x})) p(\mathbf{x}|\tilde{y}) p(\tilde{y}) d\mathbf{x} d\tilde{y} \quad (2.318)$$

$$= \inf_h 0.5 \int \ell(1, h(\mathbf{x})) p(\mathbf{x}|\tilde{y} = 1) d\mathbf{x} + 0.5 \int \ell(-1, h(\mathbf{x})) p(\mathbf{x}|\tilde{y} = -1) d\mathbf{x} \quad (2.319)$$

$$= \inf_h \int h(\mathbf{x}) Q(\mathbf{x}) d\mathbf{x} - \int h(\mathbf{x}) P(\mathbf{x}) d\mathbf{x} \quad (2.320)$$

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

which matches Equation (2.300). Thus the classifier plays the same role as the witness function.

<sup>9</sup> 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  $\theta$  to data  $x$ . (In the conditional setting, the forwards mapping is from  $(\theta, x)$  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  $\theta$  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  $\theta$  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  $\theta$  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. (For a historical review of Bayesian statistics, see e.g., [Lad96].)

### 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]<sup>1</sup>. There are also some more technical reasons for using the Bayesian approach, which we discuss below.

#### 3.1.3.1 De Finetti's theorem

The Bayesian approach is based on the idea of putting a prior on our parameters, and then using this extended model to represent our data. In this section, we justify why this is a reasonable thing to do.

First, a definition. We say that a sequence of random variables  $(\mathbf{x}_1, \mathbf{x}_2, \dots)$  is **infinitely exchangeable** if, for any  $n$ , the joint probability  $p(\mathbf{x}_1, \dots, \mathbf{x}_n)$  is invariant to permutation of the indices. That is, for any permutation  $\pi$ , we have

$$p(\mathbf{x}_1, \dots, \mathbf{x}_n) = p(\mathbf{x}_{\pi_1}, \dots, \mathbf{x}_{\pi_n}) \quad (3.3)$$

Note that iid implies exchangeability but not vice versa. For example, suppose  $(\mathbf{x}_1, \dots, \mathbf{x}_n)$  is a set of MNIST images, and let  $\mathbf{x}_0$  be a background image. The sequence  $(\mathbf{x}_0 + \mathbf{x}_1, \dots, \mathbf{x}_0 + \mathbf{x}_n)$  is infinitely exchangeable but not iid, since all the variables share a hidden common factor, namely the background  $\mathbf{x}_0$ . Thus the more examples we see, the better we will be able to estimate the shared  $\mathbf{x}_0$ , and thus the better we can predict future elements.

Thus the key advantage of working with the exchangeability assumption is that it allows us to learn from similarities between data points. In fact, one can show the following result:

<sup>1</sup> 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].

**Theorem 3.1.1 (de Finetti's theorem).** *A sequence of random variables  $(\mathbf{x}_1, \mathbf{x}_2, \dots)$  is infinitely exchangeable iff, for all  $n$ , we have*

$$p(\mathbf{x}_1, \dots, \mathbf{x}_n) = \int \prod_{i=1}^n p(\mathbf{x}_i | \boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta} \quad (3.4)$$

where  $\boldsymbol{\theta}$  is some hidden common random variable (possibly infinite dimensional). That is,  $\mathbf{x}_i$  are iid conditional on  $\boldsymbol{\theta}$ .

We often interpret  $\boldsymbol{\theta}$  as a parameter. The theorem tells us that, if our data is exchangeable, then there must exist a parameter  $\boldsymbol{\theta}$ , and a likelihood  $p(\mathbf{x}_i | \boldsymbol{\theta})$ , and a prior  $p(\boldsymbol{\theta})$ . Thus the Bayesian approach follows automatically from exchangeability [O'N09].

### 3.1.3.2 The Dutch book theorem

Bayesian inference forms the foundation of Bayesian decision theory, which we discuss in ???. It can be shown that any other method for representing uncertainty is guaranteed to cause a decision maker to lose money if deployed in the context of a gambling game. This is called the **Dutch book** theorem. For details, see e.g., [Háj08]. Of course, this argument extends beyond gambling and applies to any form of decision making under uncertainty.

### 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.

### 3.1.5 Why not just use MAP estimation?

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

$$\hat{\boldsymbol{\theta}} = \operatorname{argmax}_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta} | \mathcal{D}) = \operatorname{argmax}_{\boldsymbol{\theta}} [\log p(\mathcal{D} | \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})] \quad (3.5)$$

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

#### 3.1.5.1 The MAP estimate gives no measure of uncertainty

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

tails, we know that the maximum likelihood estimate is

$$\hat{\theta}_{\text{mle}} = \frac{N_1}{N_1 + N_0} = \frac{N_1}{N} \quad (3.6)$$

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

Now suppose we put a prior on  $\theta$ . 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

$$\hat{\theta}_{\text{map}} = \frac{\hat{\alpha} - 1}{\hat{\alpha} - 1 + \hat{\beta} - 1} \quad (3.7)$$

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**:

$$p(\mathbf{y} | \mathbf{x}, \mathcal{D}) = \int p(\mathbf{y} | \mathbf{x}, \theta) p(\theta | \mathcal{D}) d\theta \quad (3.8)$$

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**,  $p(\theta | \mathcal{D}) \approx \delta(\theta - \hat{\theta})$ ; the value  $\hat{\theta}$  could be the MLE or a MAP estimate. If we substitute this into Equation (3.8), and use the sifting property of delta functions, we get the following approximation to the posterior predictive:

$$p(\mathbf{y} | \mathbf{x}, \mathcal{D}) \approx \int p(\mathbf{y} | \mathbf{x}, \theta) \delta(\theta - \hat{\theta}) d\theta = p(\mathbf{y} | \mathbf{x}, \hat{\theta}) \quad (3.9)$$

This is called a **plugin approximation**, and is very widely used, due to its simplicity.

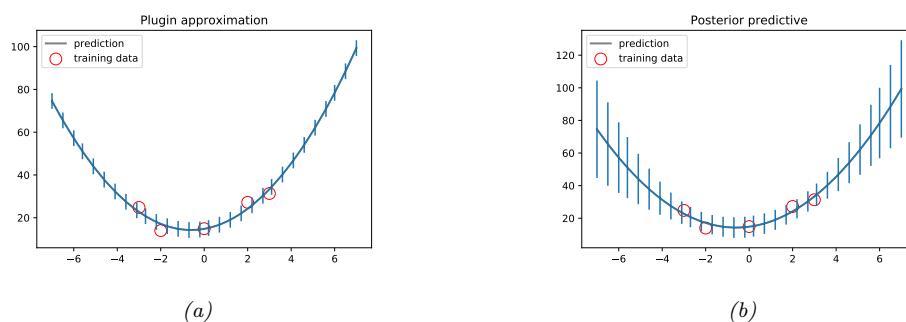


Figure 3.1: Predictions made by a polynomial regression model fit to a small dataset. (a) Plugin approximation to predictive density using the MLE. The curves shows the posterior mean,  $\mathbb{E}[y|\mathbf{x}]$ , and the error bars show the posterior standard deviation,  $\text{std}[y|\mathbf{x}]$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

However, the plugin approximation ignores uncertainty in the parameter estimates, which can result in an underestimate of the uncertainty. For example, in Figure 3.1a plots the plugin approximation  $p(y|\mathbf{x}, \hat{\boldsymbol{\theta}})$  for a linear regression model  $p(y|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(y|\mathbf{w}^\top \mathbf{x}, \sigma^2)$ , where we plug in the MLEs for  $\mathbf{w}$  and  $\sigma^2$  (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  $\sigma$  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 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 predicting 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.

### 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

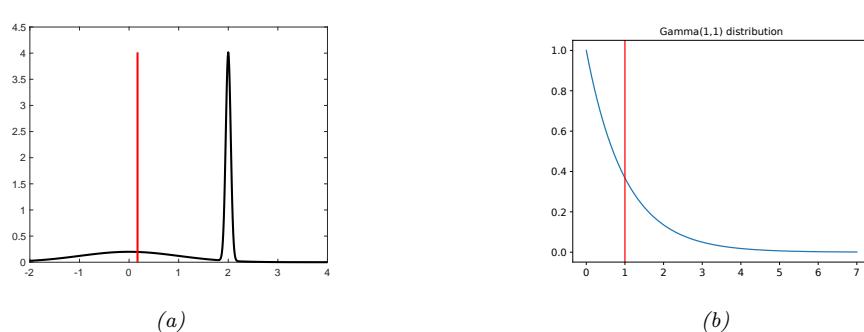


Figure 3.2: Two distributions in which the mode (highest point) is untypical of the distribution; the mean (vertical red line) is a better summary. (a) A bimodal distribution. Generated by `bimodal_dist_plot.py`. (b) A skewed  $\text{Ga}(1, 1)$  distribution. Generated by `gamma_dist_plot.py`.

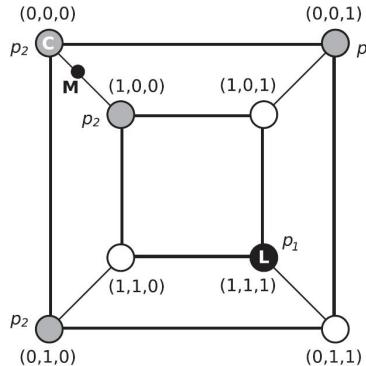


Figure 3.3: A distribution on a discrete space in which the mode (black point  $L$ , with probability  $p_1$ ) is untypical of most of the probability mass (gray circles, with probability  $p_2 < p_1$ ). The small black circle labeled  $M$  (near the top left) is the posterior mean, which is not well defined in a discrete state space.  $C$  (the top left vertex) is the centroid estimator, made up of the maximizer of the posterior marginals. See text for details. From Figure 1 of [CL07]. Used with kind permission of Luis Carvahlo.

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.

---

1    **3.1.5.4 The MAP estimate is only optimal for 0-1 loss**

2    The MAP estimate is the optimal estimate when the loss function is 0-1 loss,  $\ell(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}) = \mathbb{I}(\boldsymbol{\theta} \neq \hat{\boldsymbol{\theta}})$ , as  
3    we show in Section 3.8.3.1. However, this does not give any “partial credit” for estimating some of the  
4    components of  $\boldsymbol{\theta}$  correctly. An alternative is to use the **Hamming loss**:  $\ell(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}) = \sum_{d=1}^D \mathbb{I}(\theta_d \neq \hat{\theta}_d)$ .  
5  
6    In this case, one can show that the optimal estimator is the vector of **max marginals**

$$\hat{\boldsymbol{\theta}} = \left[ \operatorname{argmax}_{\theta_d} \int_{\boldsymbol{\theta}_{-d}} p(\boldsymbol{\theta} | \mathcal{D}) d\boldsymbol{\theta}_{-d} \right]_{d=1}^D \quad (3.10)$$

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].

10    **3.1.5.5 The MAP estimate is not invariant to reparameterization**

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

15    For example, let  $\hat{x} = \operatorname{argmax}_x p_x(x)$  be the MAP estimate for  $x$ . Now let  $y = f(x)$  be a  
16    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  
17    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  
18    variables (Section 2.7.1) to conclude  $p_y(y) = p_x(f^{-1}(y)) \left| \frac{df^{-1}(y)}{dy} \right|$ . Alternatively we can use a Monte  
19    Carlo approximation. The result is shown in Figure 2.13. We see that the original Gaussian for  
20     $p(x)$  has become “squashed” by the sigmoid nonlinearity. In particular, we see that the mode of the  
21    transformed distribution is not equal to the transform of the original mode.

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

26    **3.1.5.6 MAP estimation cannot handle the cold-start problem**

27    As the amount of data increases, the posterior  $p(\boldsymbol{\theta} | \mathcal{D})$  sometimes shrinks to a point, since the  
28    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.  
29    (We will illustrate this phenomenon in more detail later in this chapter.) Thus one may think that in  
30    the era of **big data**, we do not need to model posterior uncertainty. However, this assumes that the  
31    data is informative about all of the unknown variables. In many problems there is a **long tail** of  
32    data, in which a small number of items occur frequently, but most items occur rarely. Consequently  
33    there may be a lot of uncertainty about these rare items (see e.g., [Jor11]).

34    For example, in a recommender system, we will always be faced with new users, about whom we  
35    know very little (the so-called **cold start problem**). Bayesian methods can help create personalized  
36    recommendations in such cases, by borrowing statistical strength from other similar users for whom

we have more data (see Section 3.5). Similarly, when performing online learning and sequential decision making, an agent will often encounter new data that may not have been seen before (indeed, it may actively seek such novelty), so it may often be in a small data regime. For example, see the discussion of Bayesian optimization in Section 6.9 and bandits in Section 36.4.

## 3.2 Closed-form analysis using conjugate priors

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

### 3.2.1 The binomial model

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

#### 3.2.1.1 Bernoulli likelihood

We assume the data are **iid** or **independent and identically distributed**. Thus the likelihood has the form

$$p(\mathcal{D}|\theta) = \prod_{n=1}^N \theta^{y_n} (1-\theta)^{1-y_n} = \theta^{N_1} (1-\theta)^{N_0} \quad (3.11)$$

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  $\theta$ . 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:

$$p(\mathcal{D}|\theta) = \text{Bin}(y|N, \theta) = \binom{N}{y} \theta^y (1-\theta)^{N-y} \quad (3.12)$$

The scaling factor  $\binom{N}{y}$  is independent of  $\theta$ , so we can ignore it. Thus this likelihood is proportional to the Bernoulli likelihood in Equation (3.11), so our inferences about  $\theta$  will be the same for both models.

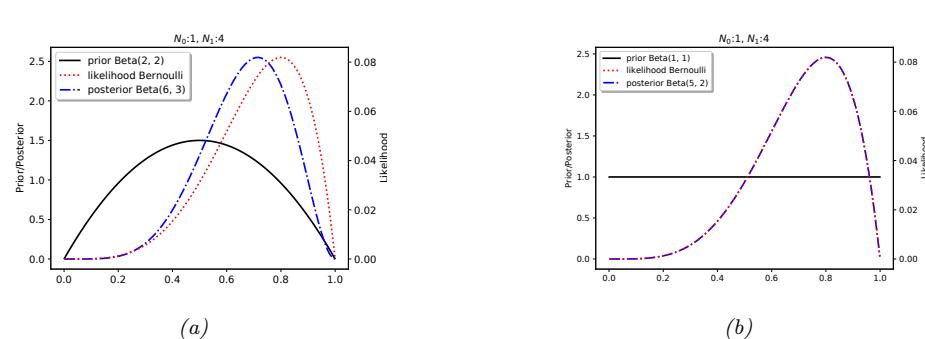


Figure 3.4: Updating a Beta prior with a Bernoulli likelihood with sufficient statistics  $N_1 = 4, N_0 = 1$ . (a) Beta(2,2) prior. (b) Uniform Beta(1,1) prior. Generated by `beta_binom_post_plot.py`.

### 3.2.1.3 Prior

To simplify the computations, we will assume that the prior  $p(\boldsymbol{\theta}) \in \mathcal{F}$  is a conjugate prior for the likelihood function  $p(\mathbf{y}|\boldsymbol{\theta})$ . This means that the posterior is in the same parameterized family as the prior, i.e.,  $p(\boldsymbol{\theta}|\mathcal{D}) \in \mathcal{F}$ .

To ensure this property when using the Bernoulli (or Binomial) likelihood, we should use a prior of the following form:

$$p(\theta) \propto \theta^{\check{\alpha}-1} (1-\theta)^{\check{\beta}-1} = \text{Beta}(\theta | \check{\alpha}, \check{\beta}) \quad (3.13)$$

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 (3.11) with the beta prior in Equation (2.22) we get a beta posterior:

$$p(\theta | \mathcal{D}) \propto \theta^{N_1} (1-\theta)^{N_0} \theta^{\bar{\alpha}-1} (1-\theta)^{\bar{\beta}-1} \quad (3.14)$$

$$\propto \text{Beta}(\theta | \check{\alpha} + N_1, \check{\beta} + N_0) \quad (3.15)$$

$$\equiv \text{Beta}(\theta | \hat{\alpha}, \hat{\beta}) \quad (3.16)$$

where  $\hat{\alpha} \triangleq \check{\alpha} + N_1$  and  $\hat{\beta} \triangleq \check{\beta} + N_0$  are the parameters of the posterior. Since the posterior has the same functional form as the prior, we say that the beta distribution is a conjugate prior for the Bernoulli likelihood.

The parameters of the prior are called **hyper-parameters**. It is clear that (in this example) the hyper-parameters play a role analogous to the sufficient statistics; they are therefore often called **pseudo counts**. We see that we can compute the posterior by simply adding the observed counts (from the likelihood) to the pseudo counts (from the prior).

The strength of the prior is controlled by  $\bar{N} = \bar{\alpha} + \beta$ ; 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  $\bar{\alpha} = \bar{\beta} = 2$ . 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  $\bar{\alpha} = \bar{\beta} = 1$ , the corresponding prior becomes the uniform distribution:

$$p(\theta) = \text{Beta}(\theta|1, 1) \propto \theta^0(1-\theta)^0 = \text{Unif}(\theta|0, 1) \quad (3.17)$$

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

$$\hat{\theta}_{\text{map}} = \arg \max_{\theta} p(\theta|\mathcal{D}) \quad (3.18)$$

$$= \arg \max_{\theta} \log p(\theta|\mathcal{D}) \quad (3.19)$$

$$= \arg \max_{\theta} \log p(\theta) + \log p(\mathcal{D}|\theta) \quad (3.20)$$

One can show that this is given by

$$\hat{\theta}_{\text{map}} = \frac{\bar{\alpha} + N_1 - 1}{\bar{\alpha} + N_1 - 1 + \bar{\beta} + N_0 - 1} \quad (3.21)$$

If we use a  $\text{Beta}(\theta|2, 2)$  prior, this amounts to **add-one smoothing**:

$$\hat{\theta}_{\text{map}} = \frac{N_1 + 1}{N_1 + 1 + N_0 + 1} = \frac{N_1 + 1}{N + 2} \quad (3.22)$$

If we use a uniform prior,  $p(\theta) \propto 1$ , the MAP estimate becomes the MLE, since  $\log p(\theta) = 0$ :

$$\hat{\theta}_{\text{mle}} = \arg \max_{\theta} \log p(\mathcal{D}|\theta) \quad (3.23)$$

When we use a Beta prior, the uniform distribution is  $\bar{\alpha} = \bar{\beta} = 1$ . In this case, the MAP estimate reduces to the MLE:

$$\hat{\theta}_{\text{mle}} = \frac{N_1}{N_1 + N_0} = \frac{N_1}{N} \quad (3.24)$$

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  $\theta$  instead of estimating it, as explained in Section 3.2.1.9.

1

### 3.2.1.7 Posterior mean

2  
3 The posterior mode can be a poor summary of the posterior, since it corresponds to a single point.  
4 The posterior mean is a more robust estimate, since it integrates over the whole space.

5 If  $p(\theta|\mathcal{D}) = \text{Beta}(\theta|\hat{\alpha}, \hat{\beta})$ , then the posterior mean is given by  
6

$$\bar{\theta} \triangleq \mathbb{E}[\theta|\mathcal{D}] = \frac{\hat{\alpha}}{\hat{\beta} + \hat{\alpha}} = \frac{\hat{\alpha}}{\hat{N}} \quad (3.25)$$

7  
8 where  $\hat{N} = \hat{\beta} + \hat{\alpha}$  is the strength (equivalent sample size) of the posterior.  
9

10 We will now show that the posterior mean is a convex combination of the prior mean,  $m = \check{\alpha} / \check{N}$   
11 (where  $\check{N} \triangleq \check{\alpha} + \check{\beta}$  is the prior strength), and the MLE:  $\hat{\theta}_{\text{mle}} = \frac{N_1}{N}$ :  
12  
13

$$\mathbb{E}[\theta|\mathcal{D}] = \frac{\check{\alpha} + N_1}{\check{\alpha} + N_1 + \check{\beta} + N_0} = \frac{\check{N} m + N_1}{N + \check{N}} = \frac{\check{N}}{N + \check{N}} m + \frac{N}{N + \check{N}} \frac{N_1}{N} = \lambda m + (1 - \lambda) \hat{\theta}_{\text{mle}} \quad (3.26)$$

14  
15 where  $\lambda = \frac{\check{N}}{N}$  is the ratio of the prior to posterior equivalent sample size. So the weaker the prior,  
16 the smaller is  $\lambda$ , and hence the closer the posterior mean is to the MLE.  
17  
18

19

### 3.2.1.8 Posterior variance

20  
21 To capture some notion of uncertainty in our estimate, a common approach is to compute the  
22 **standard error** of our estimate, which is just the posterior standard deviation:  
23  
24

$$\text{se}(\theta) = \sqrt{\mathbb{V}[\theta|\mathcal{D}]} \quad (3.27)$$

25  
26 In the case of the Bernoulli model, we showed that the posterior is a beta distribution. The variance  
27 of the beta posterior is given by  
28  
29

$$\mathbb{V}[\theta|\mathcal{D}] = \frac{\hat{\alpha}\hat{\beta}}{(\hat{\alpha} + \hat{\beta})^2(\hat{\alpha} + \hat{\beta} + 1)} = \mathbb{E}[\theta|\mathcal{D}]^2 \frac{\hat{\beta}}{\hat{\alpha}(1 + \hat{\alpha} + \hat{\beta})} \quad (3.28)$$

30  
31 where  $\hat{\alpha} = \check{\alpha} + N_1$  and  $\hat{\beta} = \check{\beta} + N_0$ . If  $N \gg \check{\alpha} + \check{\beta}$ , this simplifies to  
32

$$\mathbb{V}[\theta|\mathcal{D}] \approx \frac{N_1 N_0}{N^3} = \frac{\hat{\theta}(1 - \hat{\theta})}{N} \quad (3.29)$$

33  
34 where  $\hat{\theta}$  is the MLE. Hence the standard error is given by  
35  
36

$$\sigma = \sqrt{\mathbb{V}[\theta|\mathcal{D}]} \approx \sqrt{\frac{\hat{\theta}(1 - \hat{\theta})}{N}} \quad (3.30)$$

37  
38 We see that the uncertainty goes down at a rate of  $1/\sqrt{N}$ . We also see that the uncertainty (variance)  
39 is maximized when  $\hat{\theta} = 0.5$ , and is minimized when  $\hat{\theta}$  is close to 0 or 1. This makes sense, since it is  
40 easier to be sure that a coin is biased than to be sure that it is fair.  
41  
42

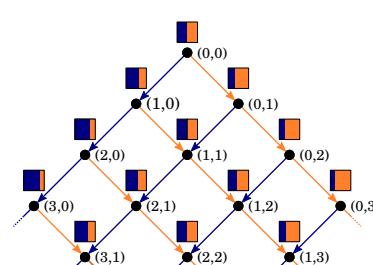


Figure 3.5: Illustration of sequential Bayesian updating for the beta-Bernoulli model. Each colored box represents the predicted distribution  $p(x_t | \mathbf{h}_t)$ , where  $\mathbf{h}_t = (N_{1,t}, N_{0,t})$  is the sufficient statistic derived from history of observations up until time  $t$ , namely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

### 3.2.1.9 Posterior predictive

Suppose we want to predict future observations. A very common approach is to first compute an estimate of the parameters based on training data,  $\hat{\theta}(\mathcal{D})$ , and then to plug that parameter back into the model and use  $p(y|\hat{\theta})$  to predict the future; this is called a **plug-in approximation**. However, this can result in overfitting. As an extreme example, suppose we have seen  $N = 3$  heads in a row. The MLE is  $\hat{\theta} = 3/3 = 1$ . However, if we use this estimate, we would predict that tails are impossible. One solution to this is to compute a MAP estimate. Here we discuss a fully Bayesian solution, in which we marginalize out  $\theta$ .

### Bernoulli model

For the Bernoulli model, the resulting **posterior predictive distribution** has the form

$$p(y = 1 | \mathcal{D}) = \int_0^1 p(y = 1 | \theta) p(\theta | \mathcal{D}) d\theta \quad (3.31)$$

$$= \int_0^1 \theta \text{Beta}(\theta | \hat{\alpha}, \hat{\beta}) d\theta = \mathbb{E}[\theta | \mathcal{D}] = \frac{\hat{\alpha}}{\hat{\alpha} + \hat{\beta}} \quad (3.32)$$

If we use a uniform prior,  $p(\theta) = \text{Beta}(\theta | 1, 1)$ , the predictive distribution becomes

$$p(y = 1 | \mathcal{D}) = \frac{N_1 + 1}{N_1 + N_0 + 2} \quad (3.33)$$

This is known as **Laplace's rule of succession**. See Figure 3.5 for an illustration of this in the sequential setting.

### 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  $\theta$  is

the same as before, but the posterior predictive distribution is different:

$$p(y|\mathcal{D}, M) = \int_0^1 \text{Bin}(y|M, \theta) \text{Beta}(\theta | \hat{\alpha}, \hat{\beta}) d\theta \quad (3.34)$$

$$= \binom{M}{y} \frac{1}{B(\hat{\alpha}, \hat{\beta})} \int_0^1 \theta^y (1-\theta)^{M-y} \theta^{\hat{\alpha}-1} (1-\theta)^{\hat{\beta}-1} d\theta \quad (3.35)$$

We recognize the integral as the normalization constant for a  $\text{Beta}(\hat{\alpha} + y, M - y + \hat{\beta})$  distribution.  
Hence

$$\int_0^1 \theta^{y+\hat{\alpha}-1} (1-\theta)^{M-y+\hat{\beta}-1} d\theta = B(y+\hat{\alpha}, M-y+\hat{\beta}) \quad (3.36)$$

Thus we find that the posterior predictive is given by the following, known as the (compound) **beta-binomial** distribution:

$$\text{BetaBinom}(x|M, \hat{\alpha}, \hat{\beta}) \triangleq \binom{M}{x} \frac{B(x+\hat{\alpha}, M-x+\hat{\beta})}{B(\hat{\alpha}, \hat{\beta})} \quad (3.37)$$

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  $\text{Beta}(1,1)$  prior. In Figure 3.6(b), we plot the plug-in approximation, given by

$$p(\theta|\mathcal{D}) \approx \delta(\theta - \hat{\theta}) \quad (3.38)$$

$$p(y|\mathcal{D}, M) = \int_0^1 \text{Bin}(y|M, \theta) p(\theta|\mathcal{D}) d\theta = \text{Bin}(y|M, \hat{\theta}) \quad (3.39)$$

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

$$p(\mathcal{D}|\mathcal{M}) = \int p(\boldsymbol{\theta}|\mathcal{M}) p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{M}) d\boldsymbol{\theta} \quad (3.40)$$

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

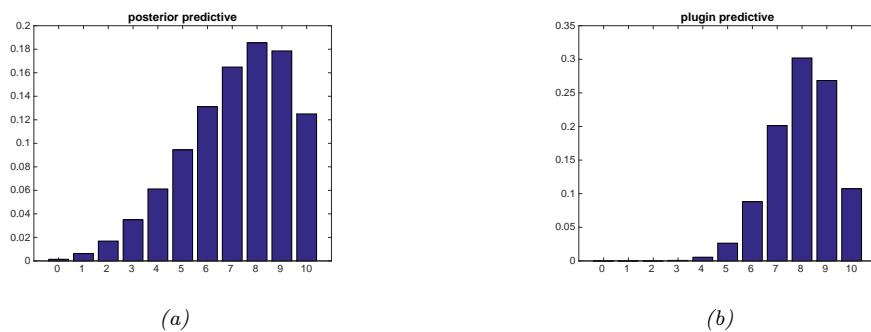


Figure 3.6: (a) Posterior predictive distributions for 10 future trials after seeing  $N_1 = 4$  heads and  $N_0 = 1$  tails. (b) Plug-in approximation based on the same data. In both cases, we use a uniform prior. Generated by `beta_binom_post_pred_plot.py`.

$p(\theta|\mathcal{D}) = \text{Beta}(\theta|a', b')$ , where  $a' = a + N_1$  and  $b' = b + N_0$ . We know the normalization constant of the posterior is  $B(a', b')$ . Hence

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \quad (3.41)$$

$$= \frac{1}{p(\mathcal{D})} \left[ \frac{1}{B(a, b)} \theta^{a-1} (1-\theta)^{b-1} \right] \left[ \binom{N}{N_1} \theta^{N_1} (1-\theta)^{N_0} \right] \quad (3.42)$$

$$= \binom{N}{N_1} \frac{1}{p(\mathcal{D})} \frac{1}{B(a, b)} [\theta^{a+N_1-1} (1-\theta)^{b+N_0-1}] \quad (3.43)$$

S<sub>0</sub>

$$\frac{1}{B(a+N_1, b+N_2)} = \binom{N}{N_1} \frac{1}{v(\mathcal{D})} \frac{1}{B(a, b)} \quad (3.44)$$

$$p(\mathcal{D}) = \binom{N}{N_1} \frac{B(a + N_1, b + N_0)}{B(a, b)} \quad (3.45)$$

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

### 3.2.1.11 Mixtures of conjugate priors

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

$$p(\theta) = 0.5 \text{ Beta}(\theta|20, 20) + 0.5 \text{ Beta}(\theta|30, 10) \quad (3.46)$$

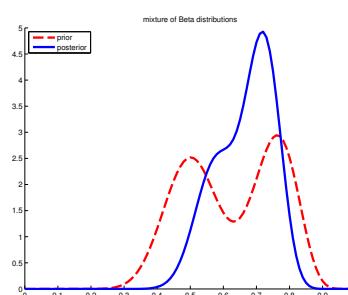


Figure 3.7: A mixture of two Beta distributions. Generated by [mixbetademo.py](#).

If  $\theta$  comes from the first distribution, the coin is fair, but if it comes from the second, it is biased towards heads.

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

$$p(\theta) = \sum_k p(h = k)p(\theta|h = k) \quad (3.47)$$

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

$$p(\theta|\mathcal{D}) = \sum_k p(h = k|\mathcal{D})p(\theta|\mathcal{D}, h = k) \quad (3.48)$$

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

$$p(h = k|\mathcal{D}) = \frac{p(h = k)p(\mathcal{D}|h = k)}{\sum_{k'} p(h = k')p(\mathcal{D}|h = k')} \quad (3.49)$$

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

Returning to our example above, if we have the prior in Equation (3.46), and we observe  $N_1 = 20$  heads and  $N_0 = 10$  tails, then, using Equation (3.45), the posterior becomes

$$p(\theta|\mathcal{D}) = 0.346 \text{ Beta}(\theta|40, 30) + 0.654 \text{ Beta}(\theta|30, 20) \quad (3.50)$$

See Figure 3.7 for an illustration.

We can compute the posterior probability that the coin is biased towards heads as follows:

$$\Pr(\theta > 0.5|\mathcal{D}) = \sum_k \Pr(\theta > 0.5|\mathcal{D}, h = k)p(h = k|\mathcal{D}) = 0.9604 \quad (3.51)$$

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

$$p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{n=1}^N \text{Cat}(y_n|\boldsymbol{\theta}) = \prod_{n=1}^N \prod_{c=1}^C \theta_c^{\mathbb{I}(y_n=c)} = \prod_{c=1}^C \theta_c^{N_c} \quad (3.52)$$

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:

$$\text{Dir}(\boldsymbol{\theta}|\boldsymbol{\alpha}) \triangleq \frac{1}{B(\boldsymbol{\alpha})} \prod_{k=1}^K \theta_k^{\alpha_k-1} \mathbb{I}(\boldsymbol{\theta} \in S_K) \quad (3.53)$$

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

$$B(\boldsymbol{\alpha}) \triangleq \frac{\prod_{k=1}^K \Gamma(\alpha_k)}{\Gamma(\sum_{k=1}^K \alpha_k)} \quad (3.54)$$

#### 3.2.2.3 Posterior

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

$$p(\boldsymbol{\theta}|\mathcal{D}) \propto p(\mathcal{D}|\boldsymbol{\theta})\text{Dir}(\boldsymbol{\theta}|\boldsymbol{\alpha}) \quad (3.55)$$

$$= \left[ \prod_k \theta_k^{N_k} \right] \left[ \prod_k \theta_k^{\alpha_k-1} \right] \quad (3.56)$$

$$= \text{Dir}(\boldsymbol{\theta}|\alpha_1+N_1, \dots, \alpha_K+N_K) \quad (3.57)$$

$$= \text{Dir}(\boldsymbol{\theta}|\boldsymbol{\hat{\alpha}}) \quad (3.58)$$

where  $\hat{\alpha}_k = \alpha_k + N_k$  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

$$\bar{\theta}_k = \frac{\hat{\alpha}_k}{\sum_{k'=1}^K \hat{\alpha}_{k'}} \quad (3.59)$$

The posterior mode, which corresponds to the MAP estimate, is given by

$$\hat{\theta}_k = \frac{\hat{\alpha}_k - 1}{\sum_{k'=1}^K (\hat{\alpha}_{k'} - 1)} \quad (3.60)$$

If we use  $\tilde{\alpha}_k = 1$ , corresponding to a uniform prior, the MAP becomes the MLE:

$$\hat{\theta}_k = N_k / N \quad (3.61)$$

### 3.2.2.4 Posterior predictive

The posterior predictive distribution is given by

$$p(y = k | \mathcal{D}) = \int p(y = k | \boldsymbol{\theta}) p(\boldsymbol{\theta} | \mathcal{D}) d\boldsymbol{\theta} \quad (3.62)$$

$$= \int \theta_k p(\theta_k | \mathcal{D}) d\theta_k = \mathbb{E}[\theta_k | \mathcal{D}] = \frac{\hat{\alpha}_k}{\sum_{k'} \hat{\alpha}_{k'}} \quad (3.63)$$

In other words, the posterior predictive distribution is given by

$$p(y | \mathcal{D}) = \text{Cat}(y | \bar{\boldsymbol{\theta}}) \quad (3.64)$$

where  $\bar{\boldsymbol{\theta}} \triangleq \mathbb{E}[\boldsymbol{\theta} | \mathcal{D}]$  are the posterior mean parameters. If instead we plug-in the MAP estimate, we will suffer from the zero-count problem. The only way to get the same effect as add-one smoothing is to use a MAP estimate with  $\tilde{\alpha}_c = 2$ .

Equation (3.63) gives the probability of a single future event, conditioned on past observations  $\mathbf{y} = (y_1, \dots, y_N)$ . In some cases, we want to know the probability of observing a batch of future data, say  $\tilde{\mathbf{y}} = (\tilde{y}_1, \dots, \tilde{y}_M)$ . We can compute this as follows:

$$p(\tilde{\mathbf{y}} | \mathbf{y}) = \frac{p(\tilde{\mathbf{y}}, \mathbf{y})}{p(\mathbf{y})} \quad (3.65)$$

The denominator is the marginal likelihood of the training data, and the numerator is the marginal likelihood of the training and future test data. We discuss how to compute such marginal likelihoods in Section 3.2.2.5.

### 3.2.2.5 Marginal likelihood

By the same reasoning as in Section 3.2.1.10, one can show that the marginal likelihood for the Dirichlet-categorical model is given by

$$p(\mathcal{D}) = \frac{B(\mathbf{N} + \boldsymbol{\alpha})}{B(\boldsymbol{\alpha})} \quad (3.66)$$

where

$$B(\boldsymbol{\alpha}) = \frac{\prod_{k=1}^K \Gamma(\alpha_k)}{\Gamma(\sum_k \alpha_k)} \quad (3.67)$$

Hence we can rewrite the above result in the following form, which is what is usually presented in the literature:

$$p(\mathcal{D}) = \frac{\Gamma(\sum_k \alpha_k)}{\Gamma(N + \sum_k \alpha_k)} \prod_k \frac{\Gamma(N_k + \alpha_k)}{\Gamma(\alpha_k)} \quad (3.68)$$

### 3.2.3 The univariate Gaussian model

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

#### 3.2.3.1 Posterior of $\mu$ given $\sigma^2$

If  $\sigma^2$  is a known constant, the likelihood for  $\mu$  has the form

$$p(\mathcal{D}|\mu) \propto \exp\left(-\frac{1}{2\sigma^2} \sum_{n=1}^N (y_n - \mu)^2\right) \quad (3.69)$$

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

$$p(\mu | \mathcal{D}, \sigma^2) = \mathcal{N}(\mu | \hat{m}, \hat{\tau}^2) \quad (3.70)$$

$$\hat{\tau}^2 = \frac{1}{\frac{N}{\sigma^2} + \frac{1}{\tilde{\tau}^2}} = \frac{\sigma^2 \tilde{\tau}^2}{N \tilde{\tau}^2 + \sigma^2} \quad (3.71)$$

$$\hat{m} = \tilde{\tau}^2 \left( \frac{\tilde{m}}{\tilde{\tau}^2} + \frac{N \bar{y}}{\sigma^2} \right) = \frac{\sigma^2}{N \tilde{\tau}^2 + \sigma^2} \tilde{m} + \frac{N \tilde{\tau}^2}{N \tilde{\tau}^2 + \sigma^2} \bar{y} \quad (3.72)$$

where  $\bar{y} \triangleq \frac{1}{N} \sum_{n=1}^N y_n$  is the empirical mean.

This result is easier to understand if we work in terms of the precision parameters, which are just inverse variances. Specifically, let  $\lambda = 1/\sigma^2$  be the observation precision, and  $\check{\lambda} = 1/\tilde{\tau}^2$  be the precision of the prior. We can then rewrite the posterior as follows:

$$p(\mu | \mathcal{D}, \lambda) = \mathcal{N}(\mu | \hat{m}, \hat{\lambda}^{-1}) \quad (3.73)$$

$$\hat{\lambda} = \check{\lambda} + N\lambda \quad (3.74)$$

$$\hat{m} = \frac{N\lambda \bar{y} + \check{\lambda} \tilde{m}}{\hat{\lambda}} = \frac{N\lambda}{N\lambda + \check{\lambda}} \bar{y} + \frac{\check{\lambda}}{N\lambda + \check{\lambda}} \tilde{m} \quad (3.75)$$

These equations are quite intuitive: the posterior precision  $\hat{\lambda}$  is the prior precision  $\check{\lambda}$  plus  $N$  units of measurement precision  $\lambda$ . Also, the posterior mean  $\hat{m}$  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 ( $\check{\lambda}$  is small relative to  $\lambda$ ), we put more weight on the empirical mean. If the prior is strong relative to the signal strength ( $\check{\lambda}$  is large relative to  $\lambda$ ), we put more weight on the prior. This is illustrated in Figure 3.8. Note

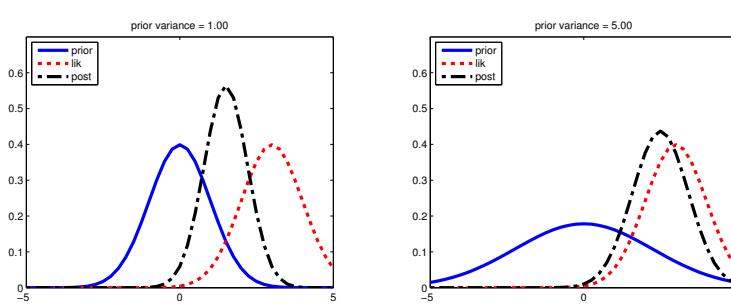


Figure 3.8: Inferring the mean of a univariate Gaussian with known  $\sigma^2$ . (a) Using strong prior,  $p(\mu) = \mathcal{N}(\mu|0, 1)$ . (b) Using weak prior,  $p(\mu) = \mathcal{N}(\mu|0, 5)$ . Generated by [gaussInferParamsMean1d.py](#).

also that the posterior mean is written in terms of  $N\lambda\bar{x}$ , so having  $N$  measurements each of precision  $\lambda$  is like having one measurement with value  $\bar{x}$  and precision  $N\lambda$ .

To gain further insight into these equations, consider the posterior after seeing a single data point  $y$  (so  $N = 1$ ). Then the posterior mean can be written in the following equivalent ways:

$$\hat{m} = \frac{\check{\lambda}}{\check{\lambda}} \check{m} + \frac{\lambda}{\check{\lambda}} y \quad (3.76)$$

$$= \check{m} + \frac{\lambda}{\check{\lambda}} (y - \check{m}) \quad (3.77)$$

$$= y - \frac{\check{\lambda}}{\check{\lambda}} (y - \check{m}) \quad (3.78)$$

The first equation is a convex combination of the prior mean and the data. The second equation is the prior mean adjusted towards the data  $y$ . The third equation is the data adjusted towards the prior mean; this is called a **shrinkage** estimate. This is easier to see if we define the weight  $w = \check{\lambda}/\check{\lambda}$ . Then we have

$$\hat{m} = y - w(y - \check{m}) = (1 - w)y + w \check{m} \quad (3.79)$$

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.

### 3.2.3.2 Posterior of $\sigma^2$ given $\mu$

If  $\mu$  is a known constant, the likelihood for  $\sigma^2$  has the form

$$p(\mathcal{D}|\sigma^2) \propto (\sigma^2)^{-N/2} \exp\left(-\frac{1}{2\sigma^2} \sum_{n=1}^N (y_n - \mu)^2\right) \quad (3.80)$$

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

$$\text{IG}(\sigma^2 | \check{a}, \check{b}) = \frac{\check{b}^{\check{a}}}{\Gamma(\check{a})} (\sigma^2)^{-(\check{a}+1)} \exp(-\frac{\check{b}}{\sigma^2}) \quad (3.81)$$

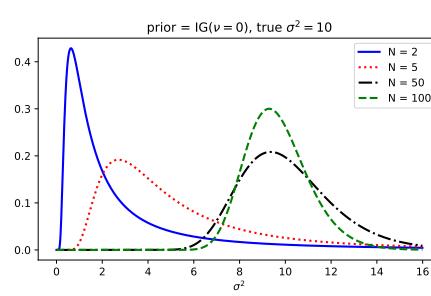


Figure 3.9: Sequential updating of the posterior for  $\sigma^2$  starting from an uninformative prior. The data was generated from a Gaussian with known mean  $\mu = 5$  and unknown variance  $\sigma^2 = 10$ . Generated by `gauss_seq_update_sigma_1d.py`

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

$$p(\sigma^2 | \mu, \mathcal{D}) = \text{IG}(\sigma^2 | \bar{a}, \bar{b}) \quad (3.82)$$

$$\bar{a} = \check{a} + N/2 \quad (3.83)$$

$$\bar{b} = \check{b} + \frac{1}{2} \sum_{n=1}^N (y_n - \mu)^2 \quad (3.84)$$

See Figure 3.9 for an illustration.

One small annoyance with using the  $\text{IG}(\check{a}, \check{b})$  distribution is that the strength of the prior is encoded in both  $\check{a}$  and  $\check{b}$ . Therefore, in the Bayesian statistics literature it is common to use an alternative parameterization of the IG distribution, known as the (scaled) **inverse chi-squared distribution**:

$$\chi^{-2}(\sigma^2 | \check{\nu}, \check{\tau}^2) = \text{IG}(\sigma^2 | \frac{\check{\nu}}{2}, \frac{\check{\nu} \check{\tau}^2}{2}) \propto (\sigma^2)^{-\check{\nu}/2-1} \exp(-\frac{\check{\nu} \check{\tau}^2}{2\sigma^2}) \quad (3.85)$$

Here  $\check{\nu}$  (called the degrees of freedom or dof parameter) controls the strength of the prior, and  $\check{\tau}^2$  encodes the prior mean. With this prior, the posterior becomes

$$p(\sigma^2 | \mathcal{D}, \mu) = \chi^{-2}(\sigma^2 | \hat{\nu}, \hat{\tau}^2) \quad (3.86)$$

$$\hat{\nu} = \check{\nu} + N \quad (3.87)$$

$$\hat{\tau}^2 = \frac{\check{\nu} \check{\tau}^2 + \sum_{n=1}^N (y_n - \mu)^2}{\hat{\nu}} \quad (3.88)$$

We see that the posterior dof  $\hat{\nu}$  is the prior dof  $\check{\nu}$  plus  $N$ , and the posterior sum of squares  $\hat{\nu} \hat{\tau}^2$  is the prior sum of squares  $\check{\nu} \check{\tau}^2$  plus the data sum of squares.

### 3.2.3.3 Posterior of $\mu$ and $\sigma^2$ : conjugate prior

Now suppose we want to infer both the mean and variance. The corresponding conjugate prior is the **normal inverse Gamma**:

$$\text{NIG}(\mu, \sigma^2 | \check{m}, \check{\kappa}, \check{a}, \check{b}) \triangleq \mathcal{N}(\mu | \check{m}, \sigma^2 / \check{\kappa}) \text{IG}(\sigma^2 | \check{a}, \check{b}) \quad (3.89)$$

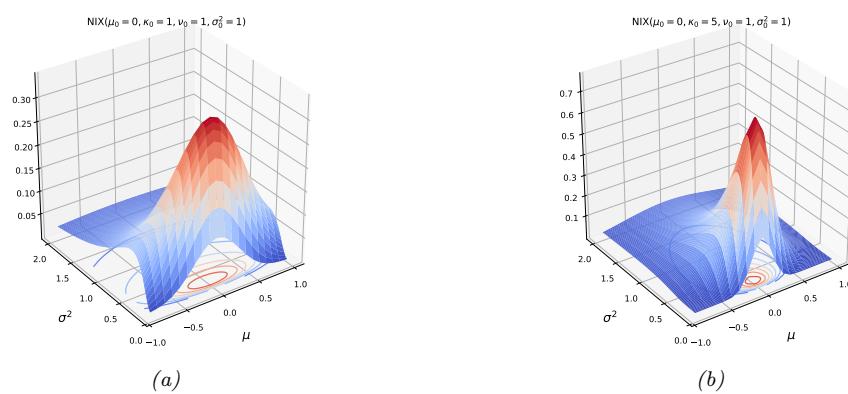


Figure 3.10: The  $N(\mu, \sigma^2 | m, \kappa, \nu, \sigma^2)$  distribution.  $m$  is the prior mean and  $k$  is how strongly we believe this;  $\sigma^2$  is the prior variance and  $\nu$  is how strongly we believe this. (a)  $m = 0$ ,  $\kappa = 1$ ,  $\nu = 1$ ,  $\sigma^2 = 1$ . Notice that the contour plot (underneath the surface) is shaped like a “squashed egg”. (b) We increase the strength of our belief in the mean by setting  $\kappa = 5$ , so the distribution for  $\mu$  around  $m = 0$  becomes narrower. Generated by [nix\\_plots.py](#).

However, it is common to use a reparameterization of this known as the **normal inverse chi-squared** or **NIX** distribution [Gel+14a, p67], which is defined by

$$NI\chi^2(\mu, \sigma^2 | \check{m}, \check{\kappa}, \check{\nu}, \check{\tau}^2) \triangleq \mathcal{N}(\mu | \check{m}, \sigma^2 / \check{\kappa}) \chi^{-2}(\sigma^2 | \check{\nu}, \check{\tau}^2) \quad (3.90)$$

$$\propto \left(\frac{1}{\sigma^2}\right)^{(\tilde{\nu}+3)/2} \exp\left(-\frac{\tilde{\nu}\tilde{\tau}^2 + \tilde{\kappa}(\mu - \tilde{m})^2}{2\sigma^2}\right) \quad (3.91)$$

See Figure 3.10 for some plots. Along the  $\mu$  axis, the distribution is shaped like a Gaussian, and along the  $\sigma^2$  axis, the distribution is shaped like a  $\chi^{-2}$ ; the contours of the joint density have a “squashed egg” appearance. Interestingly, we see that the contours for  $\mu$  are more peaked for small values of  $\sigma^2$ , which makes sense, since if the data is low variance, we will be able to estimate its mean more reliably.

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

$$p(\mu, \sigma^2 | \mathcal{D}) = NI\chi^2(\mu, \sigma^2 | \bar{m}, \bar{\kappa}, \bar{\nu}, \bar{\tau}^2) \quad (3.92)$$

$$\hat{m} = \frac{\check{\kappa}\check{m} + N\bar{x}}{\hat{c}} \quad (3.93)$$

$$\hat{\mathcal{E}} = \mathcal{E} + N \quad (3.94)$$

$$\hat{\mu} - \check{\nu} + N \quad (3.95)$$

$$\hat{\nu}\hat{\tau}^2 = \check{\nu}\check{\tau}^2 + \sum_{n=1}^N (y_n - \bar{y})^2 + \frac{N \check{\kappa}}{\check{\kappa} + N} (\check{m} - \bar{y})^2 \quad (3.96)$$

The interpretation of this is as follows. For  $\mu$ , the posterior mean  $\hat{m}$  is a convex combination of the prior mean  $\check{m}$  and the MLE  $\bar{x}$ ; the strength of this posterior,  $\hat{\kappa}$ , is the prior strength  $\check{\kappa}$  plus the

number of data points  $N$ . For  $\sigma^2$ , we work instead with the sum of squares: the posterior sum of squares,  $\widehat{\nu}\widehat{\tau}^2$ , is the prior sum of squares  $\check{\nu}\check{\tau}^2$  plus the data sum of squares,  $\sum_{n=1}^N(y_n - \bar{y})^2$ , plus a term due to the discrepancy between the prior mean  $\check{m}$  and the MLE  $\bar{y}$ . The strength of this posterior,  $\widehat{\nu}$ , is the prior strength  $\check{\nu}$  plus the number of data points  $N$ ;

The posterior marginal for  $\sigma^2$  is just

$$p(\sigma^2|\mathcal{D}) = \int p(\mu, \sigma^2|\mathcal{D})d\mu = \chi^{-2}(\sigma^2|\widehat{\nu}, \widehat{\tau}^2) \quad (3.97)$$

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

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

$$p(\mu|\mathcal{D}) = \int p(\mu, \sigma^2|\mathcal{D})d\sigma^2 = \mathcal{T}(\mu|\widehat{m}, \widehat{\tau}^2 / \widehat{\kappa}, \widehat{\nu}) \quad (3.98)$$

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

### 3.2.3.4 Posterior of $\mu$ and $\sigma^2$ : uninformative prior

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

$$p(\mu, \sigma^2) \propto p(\mu)p(\sigma^2) \propto \sigma^{-2} \quad (3.99)$$

We can simulate this with a conjugate prior by using

$$p(\mu, \sigma^2) = NI\chi^2(\mu, \sigma^2 | \check{m}=0, \check{\kappa}=0, \check{\nu}=-1, \check{\tau}^2=0) \quad (3.100)$$

With this prior, the posterior has the form

$$p(\mu, \sigma^2|\mathcal{D}) = NI\chi^2(\mu, \sigma^2 | \widehat{m}=\bar{y}, \widehat{\kappa}=N, \widehat{\nu}=N-1, \widehat{\tau}^2=s^2) \quad (3.101)$$

where

$$s^2 \triangleq \frac{1}{N-1} \sum_{n=1}^N (y_n - \bar{y})^2 = \frac{N}{N-1} \hat{\sigma}_{\text{mle}}^2 \quad (3.102)$$

$s$  is known as the **sample standard deviation**. Hence the marginal posterior for the mean is given by

$$p(\mu|\mathcal{D}) = \mathcal{T}(\mu|\bar{y}, \frac{s^2}{N}, N-1) = \mathcal{T}(\mu|\bar{y}, \frac{\sum_{n=1}^N (y_n - \bar{y})^2}{N(N-1)}, N-1) \quad (3.103)$$

Thus the posterior variance of  $\mu$  is

$$\mathbb{V}[\mu|\mathcal{D}] = \frac{\widehat{\nu}}{\widehat{\nu}-2} \widehat{\tau}^2 = \frac{N-1}{N-3} \frac{s^2}{N} \rightarrow \frac{s^2}{N} \quad (3.104)$$

The square root of this is called the **standard error of the mean**:

$$\text{se}(\mu) \triangleq \sqrt{\mathbb{V}[\mu|\mathcal{D}]} \approx \frac{s}{\sqrt{N}} \quad (3.105)$$

Thus we can approximate the 95% **credible interval** for  $\mu$  using

$$I_{.95}(\mu|\mathcal{D}) = \bar{y} \pm 2 \frac{s}{\sqrt{N}} \quad (3.106)$$

### 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

$$p(\mathcal{D}|\boldsymbol{\mu}) = \mathcal{N}(\bar{\mathbf{y}}|\boldsymbol{\mu}, \frac{1}{N}\boldsymbol{\Sigma}) \quad (3.107)$$

For simplicity, we will use a conjugate prior, which in this case is a Gaussian. In particular, if  $p(\boldsymbol{\mu}) = \mathcal{N}(\boldsymbol{\mu}|\tilde{\mathbf{m}}, \tilde{\mathbf{V}})$  then we can derive a Gaussian posterior for  $\boldsymbol{\mu}$  based on the results in Section 2.3.4. We get

$$p(\boldsymbol{\mu}|\mathcal{D}, \boldsymbol{\Sigma}) = \mathcal{N}(\boldsymbol{\mu}|\hat{\mathbf{m}}, \hat{\mathbf{V}}) \quad (3.108)$$

$$\hat{\mathbf{V}}^{-1} = \tilde{\mathbf{V}}^{-1} + N\boldsymbol{\Sigma}^{-1} \quad (3.109)$$

$$\hat{\mathbf{m}} = \hat{\mathbf{V}} (\boldsymbol{\Sigma}^{-1}(N\bar{\mathbf{y}}) + \tilde{\mathbf{V}}^{-1}\tilde{\mathbf{m}}) \quad (3.110)$$

Figure 3.11 gives a 2d example of these results.

#### 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:

$$p(\mathcal{D}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \propto |\boldsymbol{\Sigma}|^{-\frac{N}{2}} \exp\left(-\frac{1}{2}\text{tr}(\mathbf{S}_\mu \boldsymbol{\Sigma}^{-1})\right) \quad (3.111)$$

where

$$\mathbf{S}_\mu \triangleq \sum_{n=1}^N (\mathbf{y}_n - \boldsymbol{\mu})(\mathbf{y}_n - \boldsymbol{\mu})^\top \quad (3.112)$$

is the scatter matrix around  $\boldsymbol{\mu}$ .

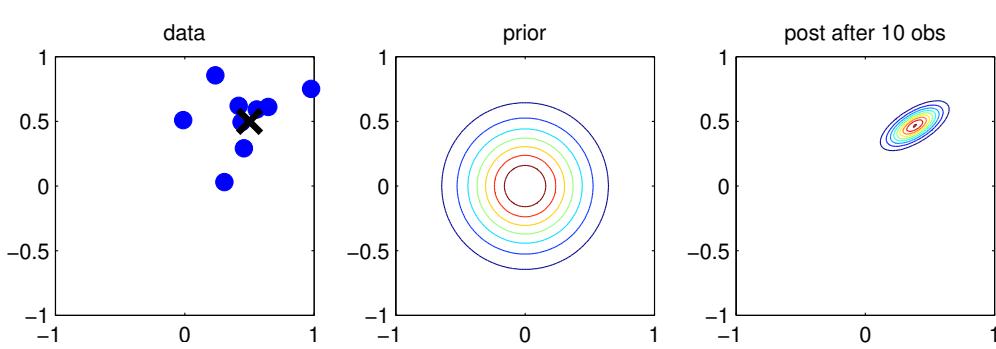


Figure 3.11: Illustration of Bayesian inference for the mean of a 2d Gaussian. (a) The data is generated from  $\mathbf{y}_n \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ , where  $\boldsymbol{\mu} = [0.5, 0.5]^\top$  and  $\boldsymbol{\Sigma} = 0.1[2, 1; 1, 1]$ . (b) The prior is  $p(\boldsymbol{\mu})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

## 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:

$$\text{IW}(\boldsymbol{\Sigma} | \check{\mathbf{S}}, \check{\nu}) \propto |\boldsymbol{\Sigma}|^{-(\check{\nu}+D+1)/2} \exp\left(-\frac{1}{2}\text{tr}(\check{\mathbf{S}} \boldsymbol{\Sigma}^{-1})\right) \quad (3.113)$$

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  $N$ .

## Posterior

Multiplying the likelihood and prior we find that the posterior is also inverse Wishart:

$$\begin{aligned} p(\boldsymbol{\Sigma} | \mathcal{D}, \boldsymbol{\mu}) &\propto |\boldsymbol{\Sigma}|^{-\frac{N}{2}} \exp\left(-\frac{1}{2}\text{tr}(\boldsymbol{\Sigma}^{-1} \mathbf{S}_\mu)\right) |\boldsymbol{\Sigma}|^{-(\check{\nu}+D+1)/2} \\ &\quad \exp\left(-\frac{1}{2}\text{tr}(\boldsymbol{\Sigma}^{-1} \check{\mathbf{S}})\right) \end{aligned} \quad (3.114)$$

$$= |\boldsymbol{\Sigma}|^{-\frac{N+(\check{\nu}+D+1)}{2}} \exp\left(-\frac{1}{2}\text{tr}[\boldsymbol{\Sigma}^{-1}(\mathbf{S}_\mu + \check{\mathbf{S}})]\right) \quad (3.115)$$

$$= \text{IW}(\boldsymbol{\Sigma} | \hat{\mathbf{S}}, \hat{\nu}) \quad (3.116)$$

$$\hat{\nu} = \check{\nu} + N \quad (3.117)$$

$$\hat{\mathbf{S}} = \check{\mathbf{S}} + \mathbf{S}_\mu \quad (3.118)$$

In words, this says that the posterior strength  $\hat{\nu}$  is the prior strength  $\check{\nu}$  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  $\mathbf{S}_\mu$ .

1    **3.2.4.3 Posterior of  $\Sigma$  and  $\mu$**

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

5    **Likelihood**

7    The likelihood is given by

$$\underline{9} \quad p(\mathcal{D} | \boldsymbol{\mu}, \boldsymbol{\Sigma}) \propto |\boldsymbol{\Sigma}|^{-\frac{N}{2}} \exp \left( -\frac{1}{2} \sum_{n=1}^N (\mathbf{y}_n - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{y}_n - \boldsymbol{\mu}) \right) \quad (3.119)$$

12    One can show that

$$\underline{14} \quad \sum_{n=1}^N (\mathbf{y}_n - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{y}_n - \boldsymbol{\mu}) = \text{tr}(\boldsymbol{\Sigma}^{-1} \mathbf{S}_{\bar{\mathbf{y}}}) + N(\bar{\mathbf{y}} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\bar{\mathbf{y}} - \boldsymbol{\mu}) \quad (3.120)$$

17    where

$$\underline{19} \quad \mathbf{S}_{\bar{\mathbf{y}}} \triangleq \sum_{n=1}^N (\mathbf{y}_n - \bar{\mathbf{y}})(\mathbf{y}_n - \bar{\mathbf{y}})^\top = \mathbf{Y}^\top \mathbf{C}_N \mathbf{Y} \quad (3.121)$$

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

$$\underline{24} \quad \mathbf{C}_N \triangleq \mathbf{I}_N - \frac{1}{N} \mathbf{1}_N \mathbf{1}_N^\top \quad (3.122)$$

26    Hence we can rewrite the likelihood as follows:

$$\underline{28} \quad p(\mathcal{D} | \boldsymbol{\mu}, \boldsymbol{\Sigma}) \propto |\boldsymbol{\Sigma}|^{-\frac{N}{2}} \exp \left( -\frac{N}{2} (\boldsymbol{\mu} - \bar{\mathbf{y}})^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{\mu} - \bar{\mathbf{y}}) \right) \exp \left( -\frac{1}{2} \text{tr}(\boldsymbol{\Sigma}^{-1} \mathbf{S}_{\bar{\mathbf{y}}}) \right) \quad (3.123)$$

31    We will use this form below.

33    **Prior**

35    The obvious prior to use is the following

$$\underline{37} \quad p(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathcal{N}(\boldsymbol{\mu} | \tilde{\boldsymbol{\mu}}, \tilde{\mathbf{C}}) \text{IW}(\boldsymbol{\Sigma} | \tilde{\mathbf{S}}, \tilde{\nu}) \quad (3.124)$$

38    where IW is the inverse Wishart distribution. Unfortunately,  $\boldsymbol{\mu}$  and  $\boldsymbol{\Sigma}$  appear together in a non-  
39    factorized way in the likelihood in Equation (3.123) (see the first exponent term), so the factored  
40    prior in Equation (3.124) is not conjugate to the likelihood.<sup>2</sup>

41    The above prior is sometimes called **conditionally conjugate**, since both conditionals,  $p(\boldsymbol{\mu} | \boldsymbol{\Sigma})$  and  
42     $p(\boldsymbol{\Sigma} | \boldsymbol{\mu})$ , are individually conjugate. To create a fully conjugate prior, we need to use a prior where  $\boldsymbol{\mu}$   
43    and  $\boldsymbol{\Sigma}$  are dependent on each other. We will use a joint distribution of the form  $p(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = p(\boldsymbol{\mu} | \boldsymbol{\Sigma})p(\boldsymbol{\Sigma})$ .

45    2. Using the language of directed graphical models, we see that  $\boldsymbol{\mu}$  and  $\boldsymbol{\Sigma}$  become dependent when conditioned on  $\mathcal{D}$   
46    due to explaining away. See Figure 3.12(a).

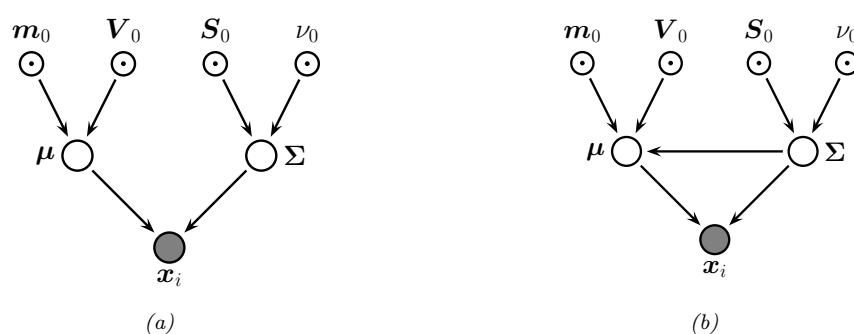


Figure 3.12: Graphical models representing different kinds of assumptions about the parameter priors. (a) A semi-conjugate prior for a Gaussian. (b) A conjugate prior for a Gaussian.

Looking at the form of the likelihood equation, Equation (3.123), we see that a natural conjugate prior has the form of a **Normal-inverse-Wishart** or **NIW** distribution, defined as follows:

$$\text{NIW}(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \check{\boldsymbol{m}}, \check{\kappa}, \check{\nu}, \check{\mathbf{S}}) \triangleq \mathcal{N}(\boldsymbol{\mu} | \check{\boldsymbol{m}}, \frac{1}{\check{\kappa}} \boldsymbol{\Sigma}) \times \text{IW}(\boldsymbol{\Sigma} | \check{\mathbf{S}}, \check{\nu}) \quad (3.125)$$

$$\begin{aligned} &= \frac{1}{Z_{\text{NIW}}} |\boldsymbol{\Sigma}|^{-\frac{1}{2}} \exp \left( -\frac{\check{\kappa}}{2} (\boldsymbol{\mu} - \check{\boldsymbol{m}})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{\mu} - \check{\boldsymbol{m}}) \right) \\ &\times |\boldsymbol{\Sigma}|^{-\frac{\check{\nu}+D+1}{2}} \exp \left( -\frac{1}{2} \text{tr}(\boldsymbol{\Sigma}^{-1} \check{\mathbf{S}}) \right) \end{aligned} \quad (3.126)$$

where the normalization constant is given by

$$Z_{\text{NIW}} \triangleq 2^{\check{\nu}D/2} \Gamma_D(\check{\nu}/2) (2\pi/\check{\kappa})^{D/2} |\check{\mathbf{S}}|^{-\check{\nu}/2} \quad (3.127)$$

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

### Posterior

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

$$\mathbf{S}_0 \triangleq \sum_{n=1}^N \mathbf{y}_n \mathbf{y}_n^T = \mathbf{Y}^T \mathbf{Y} \quad (3.128)$$

so we can rewrite the scatter matrix as follows:

$$\mathbf{S}_{\bar{\mathbf{y}}} = \mathbf{S}_0 - \frac{1}{N} \left( \sum_{n=1}^N \mathbf{y}_n \right) \left( \sum_{n=1}^N \mathbf{y}_n \right)^T = \mathbf{S}_0 - N \bar{\mathbf{y}} \bar{\mathbf{y}}^T \quad (3.129)$$

<sup>3</sup> Note that our uncertainty in the mean is proportional to the covariance. In particular, if we believe that the variance is large, then our uncertainty in  $\boldsymbol{\mu}$  must be large too. This makes sense intuitively, since if the data has large spread, it will be hard to pin down its mean.

Now we can multiply the likelihood and the prior to give

$$p(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \mathcal{D}) \propto |\boldsymbol{\Sigma}|^{-\frac{N}{2}} \exp\left(-\frac{N}{2}(\boldsymbol{\mu} - \bar{\mathbf{y}})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu} - \bar{\mathbf{y}})\right) \exp\left(-\frac{1}{2}\text{tr}(\boldsymbol{\Sigma}^{-1} \mathbf{S}_{\bar{\mathbf{y}}})\right) \quad (3.130)$$

$$\times |\boldsymbol{\Sigma}|^{-\frac{\nu+D+2}{2}} \exp\left(-\frac{\kappa}{2}(\boldsymbol{\mu} - \tilde{\mathbf{m}})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu} - \tilde{\mathbf{m}})\right) \exp\left(-\frac{1}{2}\text{tr}(\boldsymbol{\Sigma}^{-1} \tilde{\mathbf{S}})\right) \quad (3.131)$$

$$= |\boldsymbol{\Sigma}|^{-(N+\nu+D+2)/2} \exp\left(-\frac{1}{2}\text{tr}(\boldsymbol{\Sigma}^{-1} \mathbf{M})\right) \quad (3.132)$$

where

$$\mathbf{M} \triangleq N(\boldsymbol{\mu} - \bar{\mathbf{y}})(\boldsymbol{\mu} - \bar{\mathbf{y}})^\top + \kappa(\boldsymbol{\mu} - \tilde{\mathbf{m}})(\boldsymbol{\mu} - \tilde{\mathbf{m}})^\top + \mathbf{S}_{\bar{\mathbf{y}}} + \tilde{\mathbf{S}} \quad (3.133)$$

$$= (\kappa + N)\boldsymbol{\mu}\boldsymbol{\mu}^\top - \boldsymbol{\mu}(\kappa\tilde{\mathbf{m}} + N\bar{\mathbf{y}})^\top - (\kappa\tilde{\mathbf{m}} + N\bar{\mathbf{x}})\boldsymbol{\mu}^\top + \kappa\tilde{\mathbf{m}}\tilde{\mathbf{m}}^\top + \mathbf{S}_0 + \tilde{\mathbf{S}} \quad (3.134)$$

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

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

$$= (\kappa + N) \left( \boldsymbol{\mu} - \frac{\kappa\tilde{\mathbf{m}} + N\bar{\mathbf{y}}}{\kappa + N} \right) \left( \boldsymbol{\mu} - \frac{\kappa\tilde{\mathbf{m}} + N\bar{\mathbf{x}}}{\kappa + N} \right)^\top \quad (3.136)$$

$$- \frac{(\kappa\tilde{\mathbf{m}} + N\bar{\mathbf{x}})(\kappa\tilde{\mathbf{m}} + N\bar{\mathbf{y}})^\top}{\kappa + N} \quad (3.137)$$

$$= \hat{\kappa}(\boldsymbol{\mu} - \hat{\mathbf{m}})(\boldsymbol{\mu} - \hat{\mathbf{m}})^\top - \hat{\kappa}\hat{\mathbf{m}}\hat{\mathbf{m}}^\top \quad (3.138)$$

Hence we can rewrite the posterior as follows:

$$p(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \mathcal{D}) \propto |\boldsymbol{\Sigma}|^{(\hat{\nu}+D+1)/2} \exp\left(-\frac{1}{2}\text{tr}[\boldsymbol{\Sigma}^{-1}(\hat{\kappa}(\boldsymbol{\mu} - \hat{\mathbf{m}})(\boldsymbol{\mu} - \hat{\mathbf{m}})^\top + \hat{\mathbf{S}})]\right) \quad (3.139)$$

$$= \text{NIW}(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \hat{\mathbf{m}}, \hat{\kappa}, \hat{\nu}, \hat{\mathbf{S}}) \quad (3.140)$$

where

$$\hat{\mathbf{m}} = \frac{\kappa\tilde{\mathbf{m}} + N\bar{\mathbf{y}}}{\hat{\kappa}} = \frac{\kappa}{\kappa + N}\tilde{\mathbf{m}} + \frac{N}{\kappa + N}\bar{\mathbf{y}} \quad (3.141)$$

$$\hat{\kappa} = \kappa + N \quad (3.142)$$

$$\hat{\nu} = \nu + N \quad (3.143)$$

$$\hat{\mathbf{S}} = \tilde{\mathbf{S}} + \mathbf{S}_{\bar{\mathbf{y}}} + \frac{\kappa N}{\kappa + N}(\bar{\mathbf{y}} - \tilde{\mathbf{m}})(\bar{\mathbf{y}} - \tilde{\mathbf{m}})^\top \quad (3.144)$$

$$= \tilde{\mathbf{S}} + \mathbf{S}_0 + \kappa\tilde{\mathbf{m}}\tilde{\mathbf{m}}^\top - \hat{\kappa}\hat{\mathbf{m}}\hat{\mathbf{m}}^\top \quad (3.145)$$

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

---

1    **Posterior marginals**

2    We have computed the joint posterior

3

$$p(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \mathcal{D}) = \mathcal{N}(\boldsymbol{\mu} | \boldsymbol{\Sigma}, \mathcal{D}) p(\boldsymbol{\Sigma} | \mathcal{D}) = \mathcal{N}(\boldsymbol{\mu} | \hat{\boldsymbol{m}}, \frac{1}{\hat{\kappa}} \boldsymbol{\Sigma}) \text{IW}(\boldsymbol{\Sigma} | \hat{\mathbf{S}}, \hat{\nu}) \quad (3.146)$$

4

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

6

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

8

$$p(\boldsymbol{\Sigma} | \mathcal{D}) = \int p(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \mathcal{D}) d\boldsymbol{\mu} = \text{IW}(\boldsymbol{\Sigma} | \hat{\mathbf{S}}, \hat{\nu}) \quad (3.147)$$

9

10   For the mean, one can show that

11

$$p(\boldsymbol{\mu} | \mathcal{D}) = \int p(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \mathcal{D}) d\boldsymbol{\Sigma} = \mathcal{T}(\boldsymbol{\mu} | \hat{\boldsymbol{m}}, \frac{\hat{\mathbf{S}}}{\hat{\kappa}\hat{\nu}'}, \hat{\nu}') \quad (3.148)$$

12

13   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.

14    **Posterior predictive**

15

16   Following Section 3.2.2.4, we now discuss how to predict future data by integrating out the parameters. If  $\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ , 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:

17

$$p(\mathbf{y} | \mathcal{D}) = \int \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}, \boldsymbol{\Sigma}) \text{NIW}(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \hat{\boldsymbol{m}}, \hat{\kappa}, \hat{\nu}, \hat{\mathbf{S}}) d\boldsymbol{\mu} d\boldsymbol{\Sigma} \quad (3.149)$$

18

$$= \mathcal{T}(\mathbf{y} | \hat{\boldsymbol{m}}, \frac{\hat{\mathbf{S}} (\hat{\kappa} + 1)}{\hat{\kappa}\hat{\nu}'}, \hat{\nu}') \quad (3.150)$$

19

20   where  $\hat{\nu}' = \hat{\nu} - D + 1$ .

21    **3.2.5 Conjugate-exponential models**

22

23   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  $p(\mathcal{D} | \boldsymbol{\theta}) = p(s(\mathcal{D}) | \boldsymbol{\theta})$ . This suggests that the only family of distributions for which conjugate priors exist is the exponential family, a result proved in [DY79].<sup>4</sup> In the sections below, we show how to perform conjugate analysis for a generic exponential family model.

24

25   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  $p(\theta | \mathbf{x}) = \text{Pareto}(\max(\theta_0, m), \kappa + N)$ .

1    **3.2.5.1 Likelihood**

2    Recall that the likelihood of the exponential family is given by

3     $p(\mathcal{D}|\boldsymbol{\eta}) = h(\mathcal{D}) \exp(\boldsymbol{\eta}^\top \mathbf{s}(\mathcal{D}) - NA(\boldsymbol{\eta}))$     (3.151)

4    where  $\mathbf{s}(\mathcal{D}) = \sum_{n=1}^N \mathbf{s}(\mathbf{x}_n)$  and  $h(\mathcal{D}) \triangleq \prod_{n=1}^N h(\mathbf{x}_n)$ .

5    **3.2.5.2 Prior**

6    We will write the prior in a form that mirrors the likelihood:

7     $p(\boldsymbol{\eta}|\tilde{\boldsymbol{\tau}}, \tilde{\nu}) = \frac{1}{Z(\tilde{\boldsymbol{\tau}}, \tilde{\nu})} \exp(\tilde{\boldsymbol{\tau}}^\top \boldsymbol{\eta} - \tilde{\nu} A(\boldsymbol{\eta}))$     (3.152)

8    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.  
9    The parameters  $\tilde{\boldsymbol{\tau}}$  can be derived from **virtual samples** representing our prior beliefs.

10    **3.2.5.3 Posterior**

11    The posterior is given by

12     $p(\boldsymbol{\eta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\eta})p(\boldsymbol{\eta})}{p(\mathcal{D})}$     (3.153)

13     $= \frac{h(\mathcal{D})}{Z(\tilde{\boldsymbol{\tau}}, \tilde{\nu})p(\mathcal{D})} \exp((\tilde{\boldsymbol{\tau}} + \mathbf{s}(\mathcal{D}))^\top \boldsymbol{\eta} - (\tilde{\nu} + N)A(\boldsymbol{\eta}))$     (3.154)

14     $= \frac{1}{Z(\hat{\boldsymbol{\tau}}, \hat{\nu})} \exp(\hat{\boldsymbol{\tau}}^\top \boldsymbol{\eta} - \hat{\nu} A(\boldsymbol{\eta}))$     (3.155)

15    where

16     $\hat{\boldsymbol{\tau}} = \tilde{\boldsymbol{\tau}} + \mathbf{s}(\mathcal{D})$     (3.156)

17     $\hat{\nu} = \tilde{\nu} + N$     (3.157)

18     $Z(\hat{\boldsymbol{\tau}}, \hat{\nu}) = \frac{Z(\tilde{\boldsymbol{\tau}}, \tilde{\nu})}{h(\mathcal{D})} p(\mathcal{D})$     (3.158)

19    We see that this has the same form as the prior, but where we update the sufficient statistics and the  
20    sample size. We also see that the marginal likelihood is given by

21     $p(\mathcal{D}) = \frac{Z(\hat{\boldsymbol{\tau}}, \hat{\nu})h(\mathcal{D})}{Z(\tilde{\boldsymbol{\tau}}, \tilde{\nu})}$     (3.159)

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

24     $\mathbb{E}[\boldsymbol{\eta}|\mathcal{D}] = \frac{\hat{\boldsymbol{\tau}}}{\hat{\nu}} = \frac{\tilde{\boldsymbol{\tau}} + \mathbf{s}(\mathcal{D})}{\tilde{\nu} + N} = \frac{\tilde{\nu}}{\tilde{\nu} + N} \frac{\tilde{\boldsymbol{\tau}}}{\tilde{\nu}} + \frac{N}{\tilde{\nu} + N} \frac{\mathbf{s}(\mathcal{D})}{N}$     (3.160)

25     $= \lambda \mathbb{E}[\boldsymbol{\eta}] + (1 - \lambda) \hat{\boldsymbol{\eta}}_{\text{mle}}$     (3.161)

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

---

1    **3.2.5.4 Posterior predictive density**

2    We will now derive the predictive density for future observables  $\mathcal{D}' = (\tilde{x}_1, \dots, \tilde{x}_{N'})$  given past data  
3     $\mathcal{D} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$ :

4

$$p(\mathcal{D}'|\mathcal{D}) = \int p(\mathcal{D}'|\boldsymbol{\eta})p(\boldsymbol{\eta}|\mathcal{D})d\boldsymbol{\eta} \quad (3.162)$$

5

$$= \int h(\mathcal{D}') \exp(\boldsymbol{\eta}^\top \mathbf{s}(\mathcal{D}') - N' A(\boldsymbol{\eta})) \frac{1}{Z(\tilde{\boldsymbol{\tau}}, \tilde{\nu})} \exp(\boldsymbol{\eta}^\top \tilde{\boldsymbol{\tau}} - \tilde{\nu} A(\boldsymbol{\eta})) d\boldsymbol{\eta} \quad (3.163)$$

6

$$= h(\mathcal{D}') \frac{Z(\tilde{\boldsymbol{\tau}} + \mathbf{s}(\mathcal{D}) + \mathbf{s}(\mathcal{D}'), \tilde{\nu} + N + N')}{Z(\tilde{\boldsymbol{\tau}} + \mathbf{s}(\mathcal{D}), \tilde{\nu} + N)} \quad (3.164)$$

7    **3.2.5.5 Example: Bernoulli distribution**

8    As a simple example, let us revisit the Beta-Bernoulli model in our new notation.

9    The likelihood is given by

10

$$p(\mathcal{D}|\theta) = (1-\theta)^N \exp\left(\log\left(\frac{\theta}{1-\theta}\right) \sum_i x_n\right) \quad (3.165)$$

11    Hence the conjugate prior is given by

12

$$p(\theta|\nu_0, \tau_0) \propto (1-\theta)^{\nu_0} \exp\left(\log\left(\frac{\theta}{1-\theta}\right)\tau_0\right) \quad (3.166)$$

13

$$= \theta^{\tau_0} (1-\theta)^{\nu_0 - \tau_0} \quad (3.167)$$

14    If we define  $\alpha = \tau_0 + 1$  and  $\beta = \nu_0 - \tau_0 + 1$ , we see that this is a beta distribution.

15    We can derive the posterior as follows, where  $s = \sum_i \mathbb{I}(x_i = 1)$  is the sufficient statistic:

16

$$p(\theta|\mathcal{D}) \propto \theta^{\tau_0+s} (1-\theta)^{\nu_0 - \tau_0 + n - s} \quad (3.168)$$

17

$$= \theta^{\tau_n} (1-\theta)^{\nu_n - \tau_n} \quad (3.169)$$

18    We can derive the posterior predictive distribution as follows. Assume  $p(\theta) = \text{Beta}(\theta|\alpha, \beta)$ , and  
19    let  $s = s(\mathcal{D})$  be the number of heads in the past data. We can predict the probability of a given  
20    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:

21

$$p(\mathcal{D}'|\mathcal{D}) = \int_0^1 p(\mathcal{D}'|\theta|\text{Beta}(\theta|\alpha_n, \beta_n)) d\theta \quad (3.170)$$

22

$$= \frac{\Gamma(\alpha_n + \beta_n)}{\Gamma(\alpha_n)\Gamma(\beta_n)} \int_0^1 \theta^{\alpha_n + t' - 1} (1-\theta)^{\beta_n + m - t' - 1} d\theta \quad (3.171)$$

23

$$= \frac{\Gamma(\alpha_n + \beta_n)}{\Gamma(\alpha_n)\Gamma(\beta_n)} \frac{\Gamma(\alpha_{n+m})\Gamma(\beta_{n+m})}{\Gamma(\alpha_{n+m} + \beta_{n+m})} \quad (3.172)$$

24    where

25

$$\alpha_{n+m} = \alpha_n + s' = \alpha + s + s' \quad (3.173)$$

26

$$\beta_{n+m} = \beta_n + (m - s') = \beta + (n - s) + (m - s') \quad (3.174)$$

### 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  $\theta$ . 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  $\theta$  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  $\tau^2$  (relative to  $\sigma^2$ ) plays a role similar to the strength of an  $\ell_2$ -regularization term in ridge regression. In the Bayesian setting, we need to ensure we use sensible priors for the variance parameters,  $\tau^2$  and  $\sigma^2$ . 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  $\sigma^2$  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  $\epsilon$ . 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.<sup>5</sup> A more common approach is to represent the  $D \times D$  covariance matrix  $\Sigma$  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:

$$\Sigma = \text{diag}(\boldsymbol{\sigma}) \mathbf{R} \text{diag}(\boldsymbol{\sigma}) \quad (3.175)$$

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

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix} \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix} = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix} \quad (3.176)$$

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

For example,

$$p(\boldsymbol{\sigma}) = \prod_{d=1}^D \text{Expon}(\sigma_d|1) \quad (3.177)$$

For the correlation matrix, it is common to use as a prior the **LKJ distribution**, named after the authors of [LKJ09]. This has the form

$$\text{LKJ}(\mathbf{R}|\eta) \propto |\mathbf{R}|^{\eta-1} \quad (3.178)$$

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.

---

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

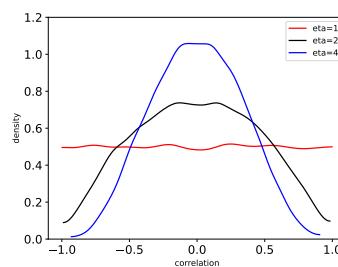


Figure 3.13: Distribution on the correlation coefficient  $\rho$  induced by a 2d LKJ distribution with varying parameter. Adapted from Figure 14.3 of [McE20]. Generated by `lkj_numpyro.py`.

## 3.4 Noninformative priors

When we have little or no domain specific knowledge, it is desirable to use an **uninformative**, **noninformative** or **objective** priors, to “let the data speak for itself”. Unfortunately, there is no unique way to define such priors, and they all encode some kind of knowledge. It is therefore better to use the term **diffuse prior**, **minimally informative prior** or **default prior**.

In the sections below, we briefly mention some common approaches for creating default priors. For further details, see e.g., [KW96] and the Stan website.<sup>6</sup>

### 3.4.1 Maximum entropy priors

A natural way to define an uninformative prior is to use one that has **maximum entropy**, since it makes the least commitments to any particular value in the state space (see Section 5.2 for a 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  $\boldsymbol{\theta}$ , and we would like our prior to match these constraints, but otherwise be maximally entropic. More precisely, suppose we want to find a distribution  $p(\boldsymbol{\theta})$  with maximum entropy, subject to the constraints that the expected values of certain features or functions  $f_k(\boldsymbol{\theta})$  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

$$\mathbb{E}[f_1(\theta)] = \mathbb{E}[\theta] = \sum_c c p_c = 1.5 \quad (3.179)$$

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

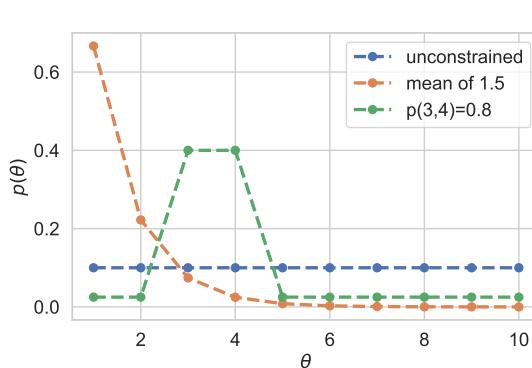


Figure 3.14: Illustration of 3 different maximum entropy priors. Adapted from Figure 1.10 of [MKL11]. Generated by `maxent_priors.py`.

In addition, we have the constraint  $\sum_c p_c = 1$ . Thus we need to solve the following optimization problem:

$$\min_{\mathbf{p}} \mathbb{H}(\mathbf{p}) \quad \text{s.t.} \quad \sum_c c p_c = 1.5, \quad \sum_c p_c = 1.0 \quad (3.180)$$

This gives the decaying exponential curve in Figure 3.14. Now suppose we know that  $\theta$  is either 3 or 4 with probability 0.8. We can encode this using

$$\mathbb{E}[f_1(\theta)] = \mathbb{E}[\mathbb{I}(\theta \in \{3, 4\})] = \Pr(\theta \in \{3, 4\}) = 0.8 \quad (3.181)$$

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  $\theta$  be a random variable with prior  $p_\theta(\theta)$ , and let  $\phi = f(\theta)$  be some invertible transformation of  $\theta$ . 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  $\theta$ . 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  $\theta$  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.<sup>7</sup> In 1d, the Jeffreys prior is given by  $p(\theta) \propto \sqrt{F(\theta)}$ , where  $F$  is the Fisher information (Section 2.6).

<sup>7</sup> 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].

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).

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  $\phi$  as follows:

$$p_\phi(\phi) = p_\theta(\theta) \left| \frac{d\theta}{d\phi} \right| \quad (3.182)$$

$$\propto \sqrt{F(\theta) \left( \frac{d\theta}{d\phi} \right)^2} = \sqrt{\mathbb{E} \left[ \left( \frac{d \log p(x|\theta)}{d\theta} \right)^2 \right] \left( \frac{d\theta}{d\phi} \right)^2} \quad (3.183)$$

$$= \sqrt{\mathbb{E} \left[ \left( \frac{d \log p(x|\theta)}{d\theta} \frac{d\theta}{d\phi} \right)^2 \right]} = \sqrt{\mathbb{E} \left[ \left( \frac{d \log p(x|\phi)}{d\phi} \right)^2 \right]} \quad (3.184)$$

$$= \sqrt{F(\phi)} \quad (3.185)$$

Thus the prior distribution is the same whether we use the  $\theta$  parameterization or the  $\phi$  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  $\theta$ . From Equation (2.236), we have

$$p(\theta) \propto \theta^{-\frac{1}{2}} (1-\theta)^{-\frac{1}{2}} = \frac{1}{\sqrt{\theta(1-\theta)}} \propto \text{Beta}(\theta | \frac{1}{2}, \frac{1}{2}) \quad (3.186)$$

Now consider the odds parameterization,  $\phi = \theta/(1-\theta)$ , so  $\theta = \frac{\phi}{\phi+1}$ . The likelihood becomes

$$p(x|\phi) \propto \left( \frac{\phi}{\phi+1} \right)^x \left( 1 - \frac{\phi}{\phi+1} \right)^{n-x} = \phi^x (\phi+1)^{-x} (\phi+1)^{-n+x} = \phi^x (\phi+1)^{-x} \quad (3.187)$$

Thus the log likelihood is

$$\ell = x \log \phi - n \log \phi + 1 \quad (3.188)$$

The first and second derivatives are

$$\frac{d\ell}{d\phi} = \frac{x}{\phi} - \frac{n}{\phi+1} \quad (3.189)$$

$$\frac{d^2\ell}{d\phi^2} = -\frac{x}{\phi^2} - \frac{n}{(\phi+1)^2} \quad (3.190)$$

Since  $\mathbb{E}[x] = n\theta = n\frac{\phi}{\phi+1}$ , the Fisher information matrix is given by

$$F(\phi) = \frac{n}{\phi(\phi+1)} - \frac{n}{(\phi+1)^2} \quad (3.191)$$

$$= \frac{n(\phi+1) - n\phi}{\phi(\phi+1)^2} = \frac{n}{\phi(\phi+1)^2} \quad (3.192)$$

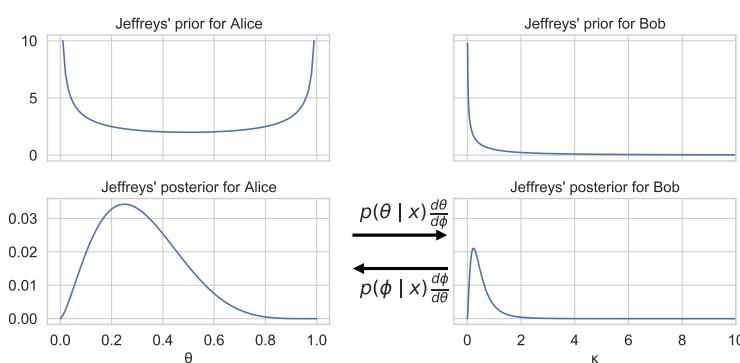


Figure 3.15: Illustration of Jeffrey's prior for Alice (who uses the rate  $\theta$ ) and Bob (who uses the odds  $\phi = \theta/(1-\theta)$ ). Adapted from Figure 1.9 of [MKL11]. Generated by [jeffreys\\_prior\\_binomial.py](#).

Hence

$$p_\phi(\phi) \propto \phi^{-0.5} (1 + \phi)^{-1} \quad (3.193)$$

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

$$p(\boldsymbol{\theta}) \propto \text{Dir}(\boldsymbol{\theta} | \frac{1}{2}, \dots, \frac{1}{2}) \quad (3.194)$$

Note that this is different from the more obvious choices of  $\text{Dir}(\frac{1}{K}, \dots, \frac{1}{K})$  or  $\text{Dir}(1, \dots, 1)$ .

### 3.4.2.3 Jeffreys prior for the mean and variance of a univariate Gaussian

Consider a 1d Gaussian  $x \sim \mathcal{N}(\mu, \sigma^2)$  with both parameters unknown, so  $\boldsymbol{\theta} = (\mu, \sigma^2)$ . From Equation (2.241), the Fisher information matrix is

$$\mathbf{F}(\boldsymbol{\theta}) = \begin{pmatrix} 1/\sigma^2 & 0 \\ 0 & 2/\sigma^2 \end{pmatrix} \quad (3.195)$$

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

$$p(\mu, \sigma^2) \propto 1/\sigma^2 \quad (3.196)$$

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

$$p(\mu, \sigma^2) = NI\chi^2(\mu, \sigma^2 | \mu_0 = 0, \check{\kappa} = 0, \check{\nu} = -1, \check{\sigma}^2 = 0) \quad (3.197)$$

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

1 **3.4.3 Invariant priors**

2 If we have “objective” prior knowledge about a problem in the form of invariances, we may be able to  
3 encode this into a prior, as we show below.  
4

5

6 **3.4.3.1 Translation-invariant priors**

7 A **location-scale family** is a family of probability distributions parameterized by a location  $\mu$  and  
8 scale  $\sigma$ . If  $x$  is an rv in this family, then  $y = a + bx$  is also an rv in the same family.

9 When inferring the location parameter  $\mu$ , it is intuitively reasonable to want to use a **translation-**  
10 **invariant prior**, which satisfies the property that the probability mass assigned to any interval,  
11  $[A, B]$  is the same as that assigned to any other shifted interval of the same width, such as  $[A - c, B - c]$ .  
12 That is,

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

13 This can be achieved using

$$p(\mu) \propto 1 \quad (3.199)$$

14 since

$$\int_{A-c}^{B-c} 1 d\mu = (A - c) - (B - c) = (A - B) = \int_A^B 1 d\mu \quad (3.200)$$

15 This is the same as the Jeffreys prior for a Gaussian with unknown mean  $\mu$  and fixed variance.  
16 This follows since  $F(\mu) = 1/\sigma^2 \propto 1$ , from Equation (2.241), and hence  $p(\mu) \propto 1$ .

17

18 **3.4.3.2 Scale-invariant prior**

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

$$\int_{A/c}^{B/c} p(\sigma) d\sigma = \int_A^B p(\sigma) d\sigma \quad (3.201)$$

22 This can be achieved by using

$$p(\sigma) \propto 1/\sigma \quad (3.202)$$

23 since then

$$\int_{A/c}^{B/c} \frac{1}{\sigma} d\sigma = [\log \sigma]_{A/c}^{B/c} = \log(B/c) - \log(A/c) = \log(B) - \log(A) = \int_A^B \frac{1}{\sigma} d\sigma \quad (3.203)$$

24 This is the same as the Jeffreys prior for a Gaussian with fixed mean  $\mu$  and unknown scale  $\sigma$ . This  
25 follows since  $F(\sigma) = 2/\sigma^2$ , from Equation (2.241), and hence  $p(\sigma) \propto 1/\sigma$ .

---

1    **3.4.3.3 Learning invariant priors**

3 Whenever we have knowledge of some kind of invariance we want our model to satisfy, we can use  
4 this to encode a corresponding prior. Sometimes this is done analytically (see e.g., [Rob07, Ch.9]).  
5 When this is intractable, it may be possible to learn invariant priors by solving a variational  
6 optimization problem (see e.g., [NS18]).  
7

8    **3.4.4 Reference priors**

10 One way to define a noninformative prior is as a distribution which is maximally far from all possible  
11 posteriors, when averaged over datasets. This is the basic idea behind a **reference prior** [Ber05;  
12 BBS09]. More precisely, we say that  $p(\boldsymbol{\theta})$  is a reference prior if it maximizes the expected KL  
13 divergence between posterior and prior:  
14

$$\frac{15}{16} \quad p^*(\boldsymbol{\theta}) = \underset{p(\boldsymbol{\theta})}{\operatorname{argmax}} \int_{\mathcal{D}} p(\mathcal{D}) D_{\text{KL}}(p(\boldsymbol{\theta}|\mathcal{D}) \| p(\boldsymbol{\theta})) d\mathcal{D} \quad (3.204)$$

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

19 We can eliminate the integral over datasets by noting that

$$\frac{21}{22} \quad \int p(\mathcal{D}) \int p(\boldsymbol{\theta}|\mathcal{D}) \log \frac{p(\boldsymbol{\theta}|\mathcal{D})}{p(\boldsymbol{\theta})} = \int p(\boldsymbol{\theta}) \int p(\mathcal{D}|\boldsymbol{\theta}) \log \frac{p(\mathcal{D}|\boldsymbol{\theta})}{p(\mathcal{D})} = \mathbb{E}_{\boldsymbol{\theta}} [D_{\text{KL}}(p(\mathcal{D}|\boldsymbol{\theta}) \| p(\mathcal{D}))] \quad (3.205)$$

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

25 One can show that, in 1d, the corresponding prior is equivalent to the Jeffreys prior. In higher  
26 dimensions, we can compute the reference prior for one parameter at a time, using the chain rule.  
27 However, this can become computationally intractable. See [NS17] for a tractable approximation  
28 based on variational inference (Section 10.1).  
29

30    **3.5 Hierarchical priors**

32 Bayesian models require specifying a prior  $p(\boldsymbol{\theta})$  for the parameters. The parameters of the prior are  
33 called **hyperparameters**, and will be denoted by  $\phi$ . If these are unknown, we can put a prior on  
34 them; this defines a **hierarchical Bayesian model**, or **multi-level model**, which can visualize  
35 like this:  $\phi \rightarrow \boldsymbol{\theta} \rightarrow \mathcal{D}$ . We assume the prior on the hyper-parameters is fixed (e.g., we may use some  
36 kind of minimally informative prior), so the joint distribution has the form  
37

$$\frac{38}{39} \quad p(\phi, \boldsymbol{\theta}, \mathcal{D}) = p(\phi)p(\boldsymbol{\theta}|\phi)p(\mathcal{D}|\boldsymbol{\theta}) \quad (3.206)$$

40 The hope is that we can learn the hyperparameters by treating the parameters themselves as  
41 datapoints.

42 A common setting in which such an approach makes sense is when we have  $J > 1$  related datasets,  
43  $\mathcal{D}_j$ , each with their own parameters  $\boldsymbol{\theta}_j$ . Inferring  $p(\boldsymbol{\theta}_j|\mathcal{D}_j)$  independently for each group  $j$  can give  
44 poor results if  $\mathcal{D}_j$  is a small dataset (e.g., if condition  $j$  corresponds to a rare combination of features,  
45 or a sparsely populated region). We could of course pool all the data to compute a single model,  
46  $p(\boldsymbol{\theta}|\mathcal{D})$ , but that would not let us model the subpopulations. A hierarchical Bayesian model lets us  
47

**borrow statistical strength** from groups with lots of data (and hence well-informed posteriors  $p(\boldsymbol{\theta}_j|\mathcal{D})$ ) in order to help groups with little data (and hence highly uncertain posteriors  $p(\boldsymbol{\theta}_j|\mathcal{D})$ ). The idea is that well-informed groups  $j$  will have a good estimate of  $\boldsymbol{\theta}_j$ , from which we can infer  $\boldsymbol{\phi}$ , which can be used to help estimate  $\boldsymbol{\theta}_k$  for groups  $k$  with less data. (Information is shared via the hidden common parent node  $\boldsymbol{\phi}$  in the graphical model, as shown in Figure 3.16.) We give some examples of this below.

After fitting such models, we can compute two kinds of posterior predictive distributions. If we want to predict observations for an existing group  $j$ , we need to use

$$p(y_j|\mathcal{D}) = \int p(y_j|\boldsymbol{\theta}_j)p(\boldsymbol{\theta}_j|\mathcal{D})d\boldsymbol{\theta}_j \quad (3.207)$$

However, if we want to predict observations for a new group  $*$  that has not yet been measured, but which is comparable to (or **exchangeable with**) the existing groups  $1 : J$ , we need to use

$$p(y_*|\mathcal{D}) = \int p(y_*|\boldsymbol{\theta}_*)p(\boldsymbol{\theta}_*|\boldsymbol{\phi})p(\boldsymbol{\phi}|\mathcal{D})d\boldsymbol{\theta}_*d\boldsymbol{\phi} \quad (3.208)$$

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

### 3.5.1 A hierarchical binomial model

Suppose we want to estimate the prevalence of some disease amongst different group of individuals, either people or animals. Let  $N_j$  be the size of the  $j$ 'th group, and let  $y_j$  be the number of positive cases for group  $j = 1 : J$ . We assume  $y_j \sim \text{Bin}(N_j, \theta_j)$ , and we want to estimate the rates  $\theta_j$ . Since some groups may have small population sizes, we may get unreliable results if we estimate each  $\theta_j$  separately; for example we may observe  $y_j = 0$  resulting in  $\hat{\theta}_j = 0$ , even though the true infection rate is higher.

One solution is to assume all the  $\theta_j$  are the same; this is called **parameter tying**. The resulting 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 rather strong one. A compromise approach is to assume that the  $\theta_j$  are similar, but that there may be group-specific variations. This can be modeled by assuming the  $\theta_j$  are drawn from some common distribution, say  $\theta_j \sim \text{Beta}(a, b)$ . The full joint distribution can be written as

$$p(\mathcal{D}, \boldsymbol{\theta}, \boldsymbol{\phi}) = p(\boldsymbol{\phi})p(\boldsymbol{\theta}|\boldsymbol{\phi})p(\mathcal{D}|\boldsymbol{\theta}) = p(\boldsymbol{\phi}) \left[ \prod_{j=1}^J \text{Beta}(\theta_j|\boldsymbol{\phi}) \right] \left[ \prod_{j=1}^J \text{Bin}(y_j|N_j, \theta_j) \right] \quad (3.209)$$

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  $\mu$  and standard deviation  $\sigma$ . We will use an improper prior for these moments,  $p(\mu, \sigma) \propto 1$ . This induces the following prior over the original hyper-parameters:

$$p(a, b) \propto (a + b)^{-5/2} \quad (3.210)$$

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

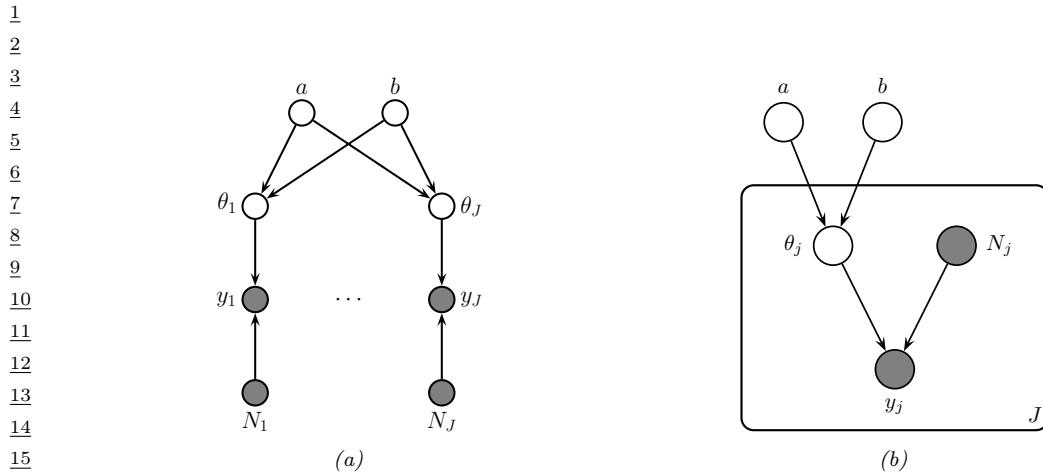


Figure 3.16: PGM for a hierarchical binomial model. (a) “Unrolled” model. (b) Same model, using plate notation.

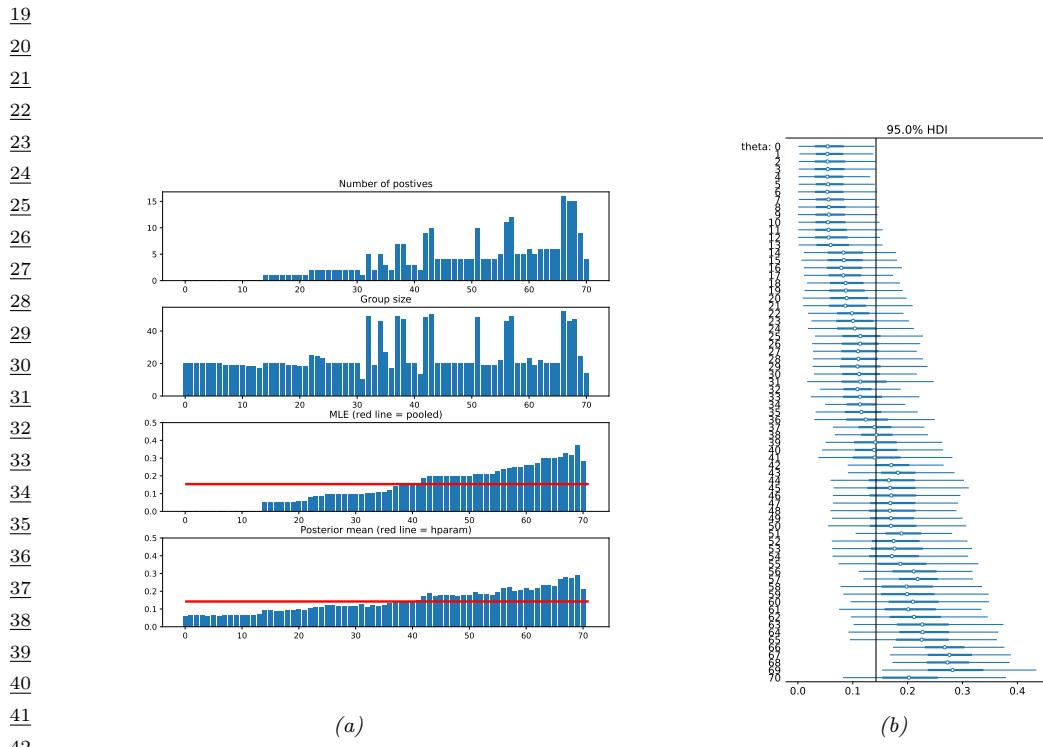


Figure 3.17: Data and inferences for the hierarchical binomial model fit using HMC. Generated by [hbayes\\_binom\\_rats\\_pymc3.ipynb](#).

45  
46  
47

Carlo sampling, and which is implemented in many software libraries. The result is a set of samples from the posterior,  $(\phi^s, \theta^s) \sim p(\phi, \theta | \mathcal{D})$  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; this phenomenon is called **shrinkage**. The amount of shrinkage is controlled by the prior on  $(a, b)$ , which is inferred from the data.

In Figure 3.17a(b), we show the 95% credible intervals for each parameter, as well as the overall population mean. (This is known as a **forest plot**.) We can use this to decide if any group is significantly different than any specified target value (e.g., the overall average).

### 3.5.2 A hierarchical Gaussian model

In this section, we consider a variation of the model in Section 3.5.1, where this time we have real-valued data instead of binary count data. More specifically we assume  $y_{ij} \sim \mathcal{N}(\theta_j, \sigma^2)$ , where  $\theta_j$  is the unknown mean for group  $j$ , and  $\sigma^2$  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  $\sigma^2$  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  $\sigma_j$ '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

$$p(\mu, \tau^2, \theta_{1:J} | \mathcal{D}) \propto p(\mu)p(\tau^2) \prod_{j=1}^J \mathcal{N}(\theta_j | \mu, \tau^2) \mathcal{N}(y_j | \theta_j, \sigma_j^2) \quad (3.211)$$

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  $\sigma_j$ . See Figure 3.18a for an illustration of the data. We also show the pooled

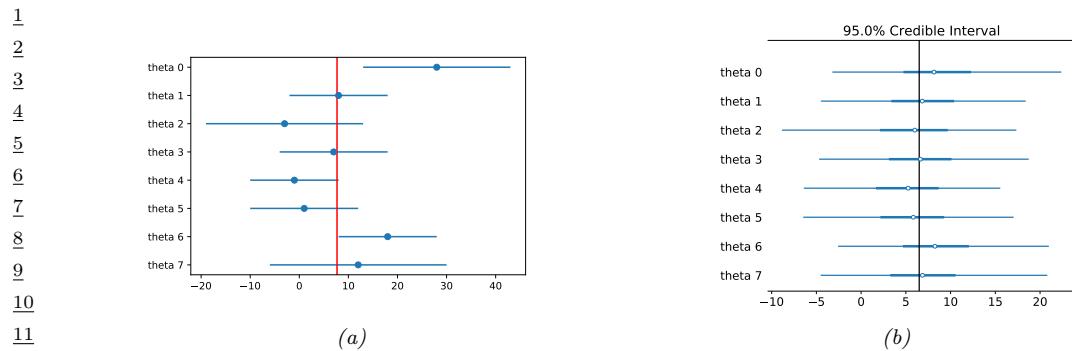


Figure 3.18: 8-schools dataset. (a) Raw data. Each row plots  $y_j \pm \sigma_j$ . Vertical line is the pooled estimate. (b) Posterior 95% credible intervals for  $\theta_j$ . Vertical line is posterior mean  $\mathbb{E}[\mu|\mathcal{D}]$ . Generated by `schools8_pymc3.py`.

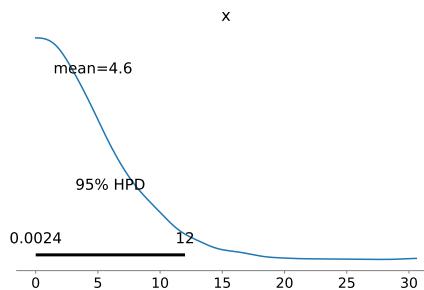


Figure 3.19: Marginal posterior density  $p(\tau|\mathcal{D})$  for the 8-schools dataset. Generated by `schools8_pymc3.py`.

MLE for  $\theta$ , which is a precision weighted average of the data:

$$\bar{y}_{..} = \frac{\sum_{j=1}^J \frac{1}{\sigma_j^2} \bar{y}_j}{\sum_{j=1}^J \frac{1}{\sigma_j^2}} \quad (3.212)$$

We see that school 0 has an unusually large improvement (28 points) compared to the overall mean, suggesting that the estimating  $\theta_0$  just based on  $\mathcal{D}_0$  might be unreliable. However, we can easily apply our hierarchical model. We will use HMC to do approximate inference. (See Section 3.6.2 for a faster approximate method.)

After computing the (approximate) posterior, we can compute the marginal posteriors  $p(\theta_j|\mathcal{D})$  for each school. These distributions are shown in Figure 3.18b. Once again, we see shrinkage 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 hyper-parameters to their posterior mean values, and use the approximation

$$p(\mu, \tau^2 | \mathcal{D}) = \delta(\mu - \bar{\mu}) \delta(\tau^2 - \bar{\tau}^2) \quad (3.213)$$

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

$$p(\theta_j | \mathcal{D}) \approx p(\theta_j | \mathcal{D}_j, \bar{\mu}, \bar{\tau}^2) \quad (3.214)$$

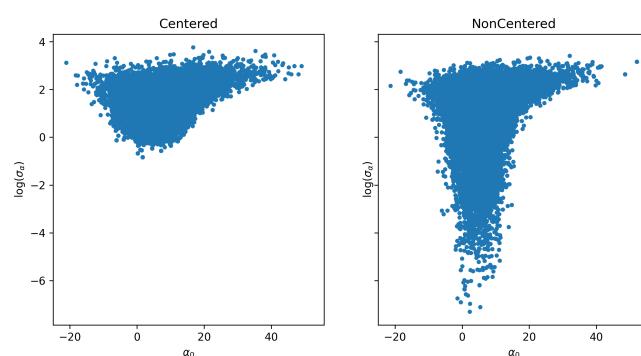


Figure 3.20: Posterior  $p(\mu_0, \log(\tau) | \mathcal{D})$  for the 8 schools model using (a) centered parameterization and (b) non-centered parameterization. Generated by [schools8\\_pymc3.ipynb](#).

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

$$\mathbb{E}[\theta_j | \mathcal{D}, \bar{\mu}, \tau^2] = w_j \bar{\mu} + (1 - w_j) \hat{\theta}_j \quad (3.215)$$

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

$$w_j = \frac{\sigma_j^2}{\sigma_j^2 + \tau^2} \quad (3.216)$$

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

### 3.5.2.2 Non-centered parameterization

It turns out that posterior inference in this model is difficult for many algorithms because of the tight dependence between the variance hyper parameter  $\tau^2$  and the group means  $\theta_j$ , as illustrated by the **funnel shape** in Figure 3.20. In particular, consider making local move through parameter space. The algorithm can only “visit” the place where  $\tau^2$  is small (corresponding to strong shrinkage to the prior) if all the  $\theta_j$  are close to the prior mean  $\mu$ . It may be hard to move into the area where  $\tau^2$  is small unless all groups *simultaneously* move their  $\theta_j$  estimates closer to  $\mu$ .

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

$$\theta_j = \mu + \tau \eta_j \quad (3.217)$$

$$\eta_j \sim \mathcal{N}(0, 1) \quad (3.218)$$

See Figure 3.21b for the corresponding graphical model. By writing  $\theta_j$  as a deterministic function of its parents plus a local noise term, we have reduced the dependence between  $\theta_j$  and  $\tau$  and hence the

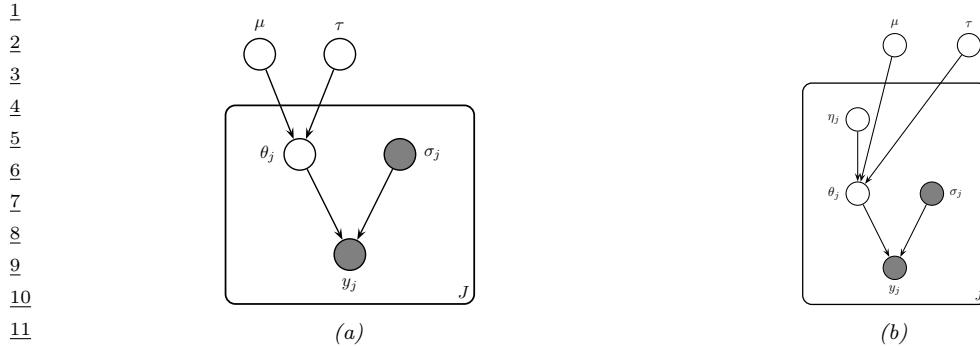


Figure 3.21: A hierarchical Gaussian Bayesian model. (a) Centered parameterization. (b) Non-centered parameterization.

other  $\theta_k$  variables, which can improve the computational efficiency of inference algorithms. as we discuss in Section 12.6.4. This kind of reparameterization is widely used in hierarchical Bayesian models. See Section 12.6.4 for more details.

## 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,  $p(\boldsymbol{\theta}|\hat{\phi}, \mathcal{D})$ , rather than the joint posterior,  $p(\boldsymbol{\theta}, \phi|\mathcal{D})$ .

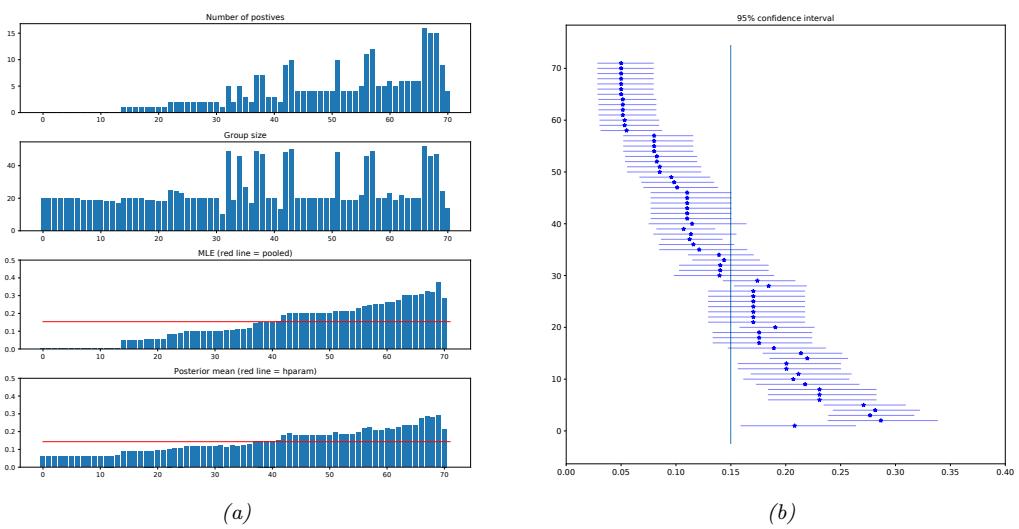
To estimate the hyper-parameters, we can maximize the marginal likelihood:

$$\hat{\phi}_{\text{mml}}(\mathcal{D}) = \underset{\phi}{\operatorname{argmax}} p(\mathcal{D}|\phi) = \underset{\phi}{\operatorname{argmax}} \int p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta}|\phi)d\boldsymbol{\theta} \quad (3.219)$$

This technique is known as **type II maximum likelihood**, since we are optimizing the hyperparameters, rather than the parameters. Once we have estimated  $\hat{\phi}$ , we compute the posterior  $p(\boldsymbol{\theta}|\hat{\phi}, \mathcal{D})$  in the usual way. This is easy to do, if the model is conjugate conditional on the hyperparameters.

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

47



(a)

(b)

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

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	$p(\theta, \phi \mathcal{D}) \propto p(\mathcal{D} \theta)p(\theta \phi)p(\phi)$

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

### 3.6.1 A hierarchical binomial model

In this section, we revisit the hierarchical binomial model from Section 3.5.1, but we use empirical Bayes instead of full Bayesian inference. We can analytically integrate out the  $\theta_j$ ’s, and write down

1 the marginal likelihood directly, as shown in Section 3.2.1.10. The resulting expression is  
 2

$$3 \quad p(\mathcal{D}|\phi) = \prod_j \int \text{Bin}(y_j|N_j, \theta_j) \text{Beta}(\theta_j|a, b) d\theta_j \quad (3.220)$$

$$4 \quad \propto \prod_j \frac{B(a+y_j, b+N_j-y_j)}{B(a, b)} \quad (3.221)$$

$$5 \quad = \prod_j \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \frac{\Gamma(a+y_j)\Gamma(b+N_j-y_j)}{\Gamma(a+b+N_j)} \quad (3.222)$$

6 Various ways of maximizing this marginal likelihood wrt  $a$  and  $b$  are discussed in [Min00].  
 7

8 Having estimated the hyper-parameters  $a$  and  $b$ , we can plug them in to compute the posterior  
 9  $p(\theta_j|\hat{a}, \hat{b}, \mathcal{D})$  for each group, using conjugate analysis in the usual way. We show the results in  
 10 Figure 3.22; they are very similar to the full Bayesian analysis shown in Figure 3.17, but the EB  
 11 method is much faster.

12

### 13 3.6.2 A hierarchical Gaussian model

14 In this section, we revisit the hierarchical Gaussian model from Section 3.5.2.1. However, we fit the  
 15 model using empirical Bayes.  
 16

17 For simplicity, we will assume that  $\sigma_j^2 = \sigma^2$  is the same for all groups. When the variances are  
 18 equal, we can derive the EB estimate in closed form, as we now show. We have  
 19

$$20 \quad p(y_j|\mu, \tau^2, \sigma^2) = \int \mathcal{N}(y_j|\theta_j, \sigma^2) \mathcal{N}(\theta_j|\mu, \tau^2) d\theta_j = \mathcal{N}(y_j|\mu, \tau^2 + \sigma^2) \quad (3.223)$$

21 Hence the marginal likelihood is  
 22

$$23 \quad p(\mathcal{D}|\mu, \tau^2, \sigma^2) = \prod_{j=1}^J \mathcal{N}(y_j|\mu, \tau^2 + \sigma^2) \quad (3.224)$$

24 Thus we can estimate the hyper-parameters using the usual MLEs for a Gaussian. For  $\mu$ , we have  
 25

$$26 \quad \hat{\mu} = \frac{1}{J} \sum_{j=1}^J y_j = \bar{y} \quad (3.225)$$

27 which is the overall mean. For  $\tau^2$ , we can use moment matching, which is equivalent to the MLE for  
 28 a Gaussian. This means we equate the model variance to the empirical variance:  
 29

$$30 \quad \hat{\tau}^2 + \sigma^2 = \frac{1}{J} \sum_{j=1}^J (y_j - \bar{y})^2 \triangleq v \quad (3.226)$$

31 so  $\hat{\tau}^2 = v - \sigma^2$ . Since we know  $\tau^2$  must be positive, it is common to use the following revised estimate:  
 32

$$33 \quad \hat{\tau}^2 = \max\{0, v - \sigma^2\} = (v - \sigma^2)_+ \quad (3.227)$$

34

Given this, the posterior mean becomes

$$\hat{\theta}_j = \lambda\mu + (1 - \lambda)y_j = \mu + (1 - \lambda)(y_j - \mu) \quad (3.228)$$

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

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

### 3.6.3 Hierarchical Bayes for n-gram smoothing

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

$$A_{jk} = (1 - \lambda)f_{jk} + \lambda f_k = (1 - \lambda)\frac{N_{jk}}{N_j} + \lambda\frac{N_k}{N} \quad (3.229)$$

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

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

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

$$\mathbf{A}_j \sim \text{Dir}(\alpha_0 m_1, \dots, \alpha_0 m_K) = \text{Dir}(\alpha_0 \mathbf{m}) = \text{Dir}(\boldsymbol{\alpha}) \quad (3.230)$$

where  $\mathbf{A}_j$  is row  $j$  of the transition matrix,  $\mathbf{m}$  is the prior mean (satisfying  $\sum_k m_k = 1$ ) and  $\alpha_0$  is the prior strength (see Figure 3.23). In terms of the earlier notation, we have  $\boldsymbol{\theta}_j = \mathbf{A}_j$  and  $\boldsymbol{\phi} = (\alpha, \mathbf{m})$ .

The posterior is given by  $\mathbf{A}_j \sim \text{Dir}(\boldsymbol{\alpha} + \mathbf{N}_j)$ , where  $\mathbf{N}_j = (N_{j1}, \dots, N_{jK})$  is the vector that records the number of times we have transitioned out of state  $j$  to each of the other states. From Equation (3.63), the posterior predictive density is

$$p(X_{t+1} = k | X_t = j, \mathcal{D}) = \frac{N_{jk} + \alpha_j m_k}{N_j + \alpha_0} = \frac{f_{jk} N_j + \alpha_j m_k}{N_j + \alpha_0} \quad (3.231)$$

$$= (1 - \lambda_j)f_{jk} + \lambda_j m_k \quad (3.232)$$

where

$$\lambda_j = \frac{\alpha_j}{N_j + \alpha_0} \quad (3.233)$$

This is very similar to Equation (3.229) but not identical. The main difference is that the Bayesian model uses a context-dependent weight  $\lambda_j$  to combine  $m_k$  with the empirical frequency  $f_{jk}$ , rather than a fixed weight  $\lambda$ . This is like *adaptive* deleted interpolation. Furthermore, rather than backing off to the empirical marginal frequencies  $f_k$ , we back off to the model parameter  $m_k$ .

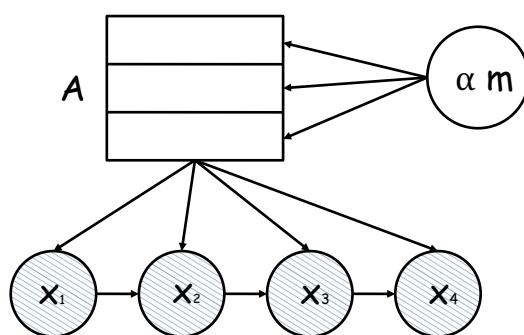


Figure 3.23: A Markov chain in which we put a different Dirichlet prior on every row of the transition matrix  $\mathbf{A}$ , but the hyperparameters of the Dirichlet are shared.

The only remaining question is: what values should we use for  $\alpha$  and  $\mathbf{m}$ ? Let's use empirical Bayes. Since we assume each row of the transition matrix is a priori independent given  $\alpha$ , the marginal likelihood for our Markov model is found by applying Equation (3.66) to each row:

$$p(\mathcal{D}|\alpha) = \prod_j \frac{B(\mathbf{N}_j + \alpha)}{B(\alpha)} \quad (3.234)$$

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]:

$$m_k \propto |\{j : N_{jk} > 0\}| \quad (3.235)$$

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 (3.229), 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 (3.232), 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 (3.235).

47

Although elegant, this Bayesian model does not beat the state-of-the-art language model, known as **interpolated Kneser-Ney** [KN95; CG98]. By using ideas from nonparametric Bayes, one can create a language model that outperforms such heuristics, as discussed in [Teh06; Woo+09]. However, one can get even better results using recurrent neural nets (Section 16.3.3); the key to their success is that they don't treat each symbol "atomically", but instead learn a distributed embedding representation, which encodes the assumption that some symbols are more similar to each other than others.

## 3.7 Model selection and evaluation

All models are wrong, but some are useful. — George Box [BD87, p424].<sup>8</sup>

In this section, we assume we have a set of different models  $\mathcal{M}$ , each of which may fit the data to different degrees, and each of which may make different assumptions. We discuss how to pick the best model from this set, or to identify that none of them may be adequate.

### 3.7.1 Bayesian model selection

The natural way to pick the best model is to pick the most probable model according to Bayes rule:

$$\hat{m} = \operatorname{argmax}_{m \in \mathcal{M}} p(m|\mathcal{D}) \quad (3.236)$$

where

$$p(m|\mathcal{D}) = \frac{p(\mathcal{D}|m)p(m)}{\sum_{m \in \mathcal{M}} p(\mathcal{D}|m)p(m)} \quad (3.237)$$

is the posterior over models. This is called **Bayesian model selection**. If the prior over models is uniform,  $p(m) = 1/|\mathcal{M}|$ , then the MAP model is given by

$$\hat{m} = \operatorname{argmax}_{m \in \mathcal{M}} p(\mathcal{D}|m) \quad (3.238)$$

The quantity  $p(\mathcal{D}|m)$  is given by

$$p(\mathcal{D}|m) = \int p(\mathcal{D}|\boldsymbol{\theta}, m)p(\boldsymbol{\theta}|m)d\boldsymbol{\theta} \quad (3.239)$$

This is known as the **marginal likelihood**, or the **evidence** for model  $m$ . (See Section 3.7.2 for details on how to compute this quantity.) If the model assigns high prior predictive density to the observed data, then we deem it a good model. If, however, the model has too much flexibility, then some prior settings will not match the data; this probability mass will be "wasted", lowering the expected likelihood. This implicit regularization effect is called the **Bayesian Occam's razor**.

<sup>8</sup> 8. George Box is a retired statistics professor at the University of Wisconsin.

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:

$$B_{1,0} \triangleq \frac{p(\mathcal{D}|M_1)}{p(\mathcal{D}|M_0)} = \frac{p(M_1|\mathcal{D})}{p(M_0|\mathcal{D})} / \frac{p(M_1)}{p(M_0)} \quad (3.240)$$

(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.

12

### 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 (3.239).

A particularly simple estimator, is known as the **harmonic mean estimator**, and was proposed in [NR94]. It is defined as follows:

$$p(\mathcal{D}) \approx \left( \frac{1}{S} \sum_{s=1}^S \frac{1}{p(\mathcal{D}|\boldsymbol{\theta}_s)} \right)^{-1} \quad (3.241)$$

This follows from the following identity:

$$\mathbb{E} \left[ \frac{1}{p(\mathcal{D}|\boldsymbol{\theta})} \right] = \int \frac{1}{p(\mathcal{D}|\boldsymbol{\theta})} p(\boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta} \quad (3.242)$$

$$= \int \frac{1}{p(\mathcal{D}|\boldsymbol{\theta})} \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})} d\boldsymbol{\theta} \quad (3.243)$$

$$= \frac{1}{p(\mathcal{D})} \int p(\boldsymbol{\theta}) d\boldsymbol{\theta} = \frac{1}{p(\mathcal{D})} \quad (3.244)$$

(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"<sup>9</sup>). 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.

<sup>46</sup> 9. <https://bit.ly/3t7id0k>.

47

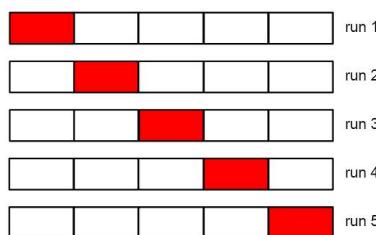


Figure 3.24: Schematic of 5-fold cross validation.

### 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

$$J_{\text{LOO}}(m) \triangleq \sum_{n=1}^N \log p(\mathbf{y}_n | \mathbf{x}_n, \hat{\boldsymbol{\theta}}(\mathcal{D}_{-n}), m) \quad (3.245)$$

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:

$$\log p(\mathcal{D}|m) = \log \prod_{n=1}^N p(\mathcal{D}_n | \mathcal{D}_{1:n-1}, m) = \sum_{n=1}^N \log p(\mathbf{y}_n | \mathbf{x}_n, \mathcal{D}_{1:n-1}, m) \quad (3.246)$$

where

$$p(\mathbf{y}_n | \mathbf{x}_n, \mathcal{D}_{1:n-1}, m) = \int p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta}) p(\boldsymbol{\theta} | \mathcal{D}_{1:n-1}, m) d\boldsymbol{\theta} \quad (3.247)$$

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:

$$p(\mathbf{y} | \mathbf{x}_n, \mathcal{D}_{1:n-1}, m) \approx \int p(\mathbf{y} | \mathbf{x}_n, \boldsymbol{\theta}) \delta(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_m(\mathcal{D}_{1:n-1})) d\boldsymbol{\theta} = p(\mathbf{y} | \mathbf{x}_n, \hat{\boldsymbol{\theta}}_m(\mathcal{D}_{1:n-1})) \quad (3.248)$$

1 Then Equation (3.246) simplifies to  
2

3  
4  $\log p(\mathcal{D}|m) \approx \sum_{n=1}^N \log p(\mathbf{y}_n|\mathbf{x}_n, \hat{\boldsymbol{\theta}}(\mathcal{D}_{1:n-1}), m)$  (3.249)  
5  
6

7 This is very similar to Equation (3.245), 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.  
10

### 11 3.7.4 Pareto-Smoothed Importance Sampling LOO estimate 12

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:  
16

17  
18  $\text{LPPD} \triangleq \sum_{n=1}^N \log p(\mathbf{y}_n|\mathbf{x}_n, \mathcal{D}, m) = \sum_{n=1}^N \log \int p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}, m) p(\boldsymbol{\theta}|\mathcal{D}, m) d\boldsymbol{\theta}$  (3.250)  
19

20 We can approximate LPPD with Monte Carlo:  
21

22  
23  $\text{LPPD}(m) \approx \sum_{n=1}^N \log \left( \frac{1}{S} \sum_{s=1}^S p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}_s, m) \right)$  (3.251)  
24

25 where  $\boldsymbol{\theta}_s \sim p(\boldsymbol{\theta}|\mathcal{D}, m)$  is a posterior sample.  
26

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*,  $\mathbf{y}_*$ :

29  
30  $\text{ELPD} \triangleq \mathbb{E}_{\mathbf{y}_*} \log p(\mathbf{y}_*|\mathbf{x}_*, \mathcal{D}, m) = \int p(\mathbf{y}_*|\mathbf{x}_*, \boldsymbol{\theta}, m) \log \int p(\mathbf{y}_*|\mathbf{x}_*, \boldsymbol{\theta}, m) p(\boldsymbol{\theta}|\mathcal{D}, m) d\boldsymbol{\theta}$  (3.252)  
31

32 Of course, the future data is unknown, but we can use a LOO approximation:  
33

34  
35  $\text{ELPD}_{\text{LOO}} \triangleq \sum_{n=1}^N \log p(\mathbf{y}_n|\mathbf{x}_n, \mathcal{D}_{-n}, m) = \sum_{n=1}^N \log \int p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}, m) p(\boldsymbol{\theta}|\mathcal{D}_{-n}, m) d\boldsymbol{\theta}$  (3.253)  
36

37 This is a Bayesian version of Equation (3.245). We can approximate this integral using Monte Carlo:  
38

39  
40  $\text{ELPD}_{\text{LOO}} \approx \sum_{n=1}^N \log \left( \frac{1}{S} \sum_{s=1}^S p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}_{s,-n}, m) \right)$  (3.254)  
41  
42

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  
47

the target distribution of interest, and let  $g(\boldsymbol{\theta}) = p(\boldsymbol{\theta}|\mathcal{D}, m)$  be the proposal. Define the importance weight for each sample  $s$  when leaving out example  $n$  to be

$$w_{s,-n} = \frac{f(\boldsymbol{\theta}_s)}{g(\boldsymbol{\theta}_s)} = \frac{p(\boldsymbol{\theta}_s|\mathcal{D}_{-n})}{p(\boldsymbol{\theta}_s|\mathcal{D})} = \frac{p(\mathcal{D}_{-n}|\boldsymbol{\theta}_s)p(\boldsymbol{\theta}_s)}{p(\mathcal{D}_{-n})} \frac{p(\mathcal{D})}{p(\mathcal{D}|\boldsymbol{\theta}_s)p(\boldsymbol{\theta}_s)} \quad (3.255)$$

$$\propto \frac{p(\mathcal{D}_{-n}|\boldsymbol{\theta}_s)}{p(\mathcal{D}|\boldsymbol{\theta}_s)} = \frac{p(\mathcal{D}_{-n}|\boldsymbol{\theta}_s)}{p(\mathcal{D}_{-n}|\boldsymbol{\theta})p(\mathcal{D}_n|\boldsymbol{\theta}_s)} = \frac{1}{p(\mathcal{D}_n|\boldsymbol{\theta}_s)} \quad (3.256)$$

We then normalize the weights to get

$$\hat{w}_{s,-n} = \frac{w_{s,-n}}{\sum_{s'=1}^S w_{s',-n}} \quad (3.257)$$

and use them to get the estimate

$$\text{ELPD}_{\text{IS-LOO}} = \sum_{n=1}^N \log \left( \sum_{s=1}^S \hat{w}_{s,-n} p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta}_s, m) \right) \quad (3.258)$$

Unfortunately, the importance weights may have high variance, where some weights are much larger than others. To reduce this effect, we fit a Pareto distribution (see supplementary) to each set of weights for each sample, and use this to smooth the weights. This technique is called **Pareto smoothed importance sampling** or **PSIS** [VGG17]. The Pareto distribution has the form

$$p(r|u, \sigma, k) = \sigma^{-1} (1 + k(r - u)\sigma^{-1})^{-1/k-1} \quad (3.259)$$

where  $u$  is the location,  $\sigma$  is the scale, and  $k$  is the shape. The parameter values  $k_n$  (for each data point  $n$ ) can be used to assess how well this approximation works. If we find  $k_n > 0.5$  for any given point, it is likely an outlier, and the resulting LOO estimate is likely to be quite poor.

### 3.7.5 Information criteria

An alternative approach to cross validation is to score models using the negative log likelihood (or LPPD) on the training set plus a **complexity penalty** term:

$$\mathcal{L}(m) = -\log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, m) + C(m) \quad (3.260)$$

This is called an **information criterion**. Different methods use different complexity terms  $C(m)$ , as we discuss below. Note also that it is conventional, when working with information criteria, to scale the NLL by -2 to get the **deviance**:

$$\text{deviance}(m) = -2 \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, m) \quad (3.261)$$

This makes the math “prettier” for certain Gaussian models.

#### 3.7.5.1 Minimum description length (MDL)

We can think about the problem of scoring different models in terms of information theory (Chapter 5). The goal is for the sender to communicate the data to the receiver. First the sender needs to specify

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

$$-L(m) = -\log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, m) = -\sum_n \log p(\mathbf{y}_n|\mathbf{x}_n, \hat{\boldsymbol{\theta}}, m) \quad (3.262)$$

bits. (We are ignoring the cost of sending the input features  $\mathbf{x}_n$ , if present.) The total cost is

$$\mathcal{L}_{\text{MDL}}(m) = -\log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, m) + C(m) \quad (3.263)$$

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

$$\mathcal{L}_{\text{BIC}}(m) = -2 \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, m) + D_m \log N \quad (3.264)$$

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 (7.22)) the following:

$$\log p(\mathcal{D}|m) \approx \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}_{\text{map}}) + \log p(\hat{\boldsymbol{\theta}}_{\text{map}}) - \frac{1}{2} \log |\mathbf{H}| \quad (3.265)$$

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 (3.265) 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

$$\log p(\mathcal{D}|m) \approx \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}) - \frac{1}{2} \log |\mathbf{H}| \quad (3.266)$$

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

$$\log |\mathbf{H}| = \log |N\hat{\mathbf{H}}| = \log(N^D|\hat{\mathbf{H}}|) = D \log N + \log |\hat{\mathbf{H}}| \quad (3.267)$$

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:

$$J_{\text{BIC}}(m) = \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, m) - \frac{D_m}{2} \log N \quad (3.268)$$

We can also define the **BIC loss**, that we want to minimize, by multiplying by -2:

$$\mathcal{L}_{\text{BIC}}(m) = -2 \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, m) + D_m \log N \quad (3.269)$$

---

### 3.7.5.3 Akaike information criterion

The **Akaike information criterion** [Aka74] is closely related to BIC. It has the form

$$\mathcal{L}_{\text{AIC}}(m) = -2 \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, m) + 2D_m \quad (3.270)$$

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 (3.251)). For the complexity term, WAIC uses the variance of the predictive distribution:

$$C(m) = \sum_{n=1}^N \mathbb{V}_{\boldsymbol{\theta}|\mathcal{D},m}[\log p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}, m)] \approx \sum_{n=1}^N \mathbb{V}\{\log p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}_s, m) : s = 1 : S\} \quad (3.271)$$

The intuition for this is as follows: if, for a given datapoint  $n$ , the different posterior samples  $\boldsymbol{\theta}_s$  make very different predictions, then the model is uncertain, and likely too flexible. The complexity term essentially counts how often this occurs. The final WAIC loss is

$$\mathcal{L}_{\text{WAIC}}(m) = -2\text{LPPD}(m) + 2C(m) \quad (3.272)$$

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

Bayesian inference and decision making is optimal, but only if the modeling assumptions are correct. 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**.

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

1 assess the quality of our model, we can compute how “typical” our observed data  $\mathcal{D}$  is compared to  
 2 the model’s hallucinations. To perform this comparison, we create one or more scalar **test statistics**,  
 3  $\text{test}(\tilde{\mathcal{D}}^s)$ , and compare them to the test statistics on the actual data,  $\text{test}(\mathcal{D})$ . These statistics should  
 4 measure features of interest (since it will not, in general, be possible to capture every aspect of the  
 5 data with a given model). If there is a large difference between the distribution of  $\text{test}(\tilde{\mathcal{D}}^s)$  across  
 6 different  $s$  and the value of  $\text{test}(\mathcal{D})$ , it suggests the model is not a good one. This approach called a  
 7 **posterior predictive check** [Rub84].  
 8

### 10 3.7.6.1 Example: 1d Gaussian

11 To make things clearer, let us consider an example from [Gel+04]. In 1882, Newcomb measured the  
 12 speed of light using a certain method and obtained  $N = 66$  measurements, shown in Figure 3.25(a).  
 13 There are clearly two outliers in the left tails, suggesting that the distribution is not Gaussian. Let  
 14 us nonetheless fit a Gaussian to it. For simplicity, we will just compute the MLE, and use a plug-in  
 15 approximation to the posterior predictive density:  
 16

$$17 \quad p(\tilde{y}|\mathcal{D}) \approx \mathcal{N}(\tilde{y}|\hat{\mu}, \hat{\sigma}^2), \quad \hat{\mu} = \frac{1}{N} \sum_{n=1}^N y_n, \quad \hat{\sigma}^2 = \frac{1}{N} \sum_{n=1}^N (y_n - \hat{\mu})^2 \quad (3.273)$$

20 Let  $\tilde{\mathcal{D}}^s$  be the  $s$ ’th dataset of size  $N = 66$  sampled from this distribution, for  $s = 1 : 1000$ . The  
 21 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  
 22 samples contain the large negative examples that were seen in the real data. This suggests the model  
 23 cannot capture the long tails present in the data. (We are assuming that these extreme values are  
 24 scientifically interesting, and something we want the model to capture.)  
 25

A more formal way to test fit is to define a test statistic. Since we are interested in small values,  
 26 let us use  
 27

$$28 \quad \text{test}(\mathcal{D}) = \min\{y : y \in \mathcal{D}\} \quad (3.274)$$

30 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,  
 31  $\text{test}(\mathcal{D}) = -44$ , but the test statistics of the generated data,  $\text{test}(\tilde{\mathcal{D}})$ , are much larger. Indeed, we see  
 32 that  $-44$  is in the left tail of the predictive distribution,  $p(\text{test}(\tilde{\mathcal{D}})|\mathcal{D})$ .  
 33

### 34 3.7.7 Bayesian p-values

35 If some test statistic of the oberved data,  $\text{test}(\mathcal{D})$ , occurs in the left or right tail of the predictive  
 36 distribution, then it is very unlikely under the model. We can quantify this using a **Bayesian**  
 37 **p-value**, also called a **posterior-predictive p-value**:  
 38

$$39 \quad p_B = P(\text{test}(\tilde{\mathcal{D}}) \geq \text{test}(\mathcal{D})|\mathcal{D}) \quad (3.275)$$

40 In contrast, a frequentist p-value is defined as  
 41

$$42 \quad p_C = P(\text{test}(\tilde{\mathcal{D}}) \geq \text{test}(\mathcal{D})|\theta^*) \quad (3.276)$$

44 where  $\theta^*$  is the true but unknown parameter. The key difference between the Bayesian and classical  
 45 approach is that the Bayesian always conditions on what is known (namely the data  $\mathcal{D}$ ), and never  
 46 conditions on what is unknown (namely  $\theta^*$ ).  
 47

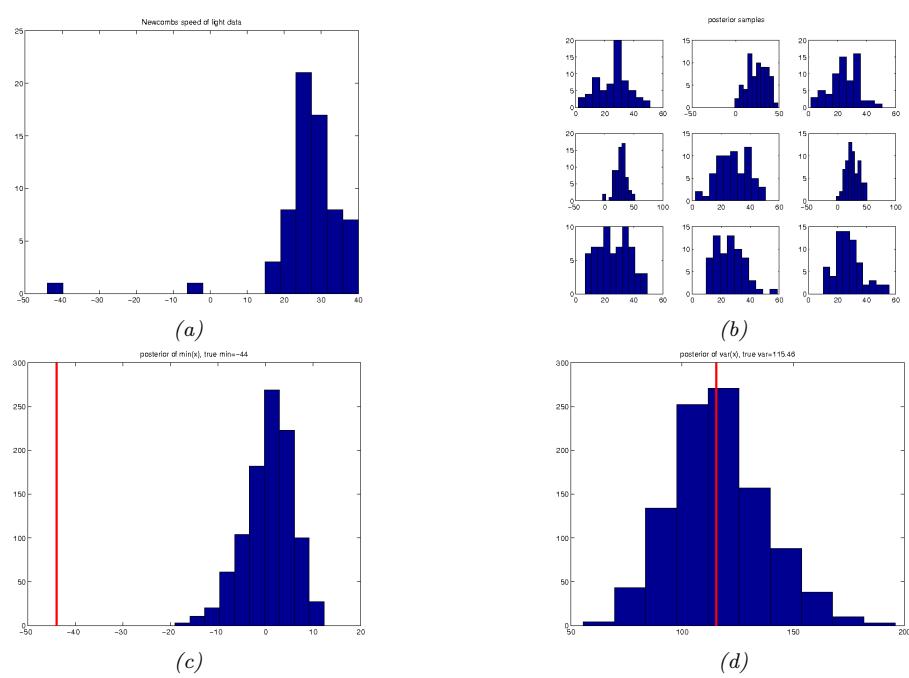


Figure 3.25: (a) Histogram of Newcomb’s data. (b) Histograms of data sampled from Gaussian model. (c) Histogram of test statistic on data sampled from the model, which represents  $p(\text{test}(\tilde{\mathcal{D}}^s)|\mathcal{D})$ , where  $\text{test}(\mathcal{D}) = \min\{y \in \mathcal{D}\}$ . The 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](#).

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

$$p_B = \int \mathbb{I}(\text{test}(\tilde{\mathcal{D}}) > \text{test}(\mathcal{D})) p(\tilde{\mathcal{D}}|\theta) p(\theta|\mathcal{D}) d\theta \approx \frac{1}{S} \sum_{s=1}^S \mathbb{I}(\text{test}(\tilde{\mathcal{D}}^s) > \text{test}(\mathcal{D})) \quad (3.277)$$

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

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

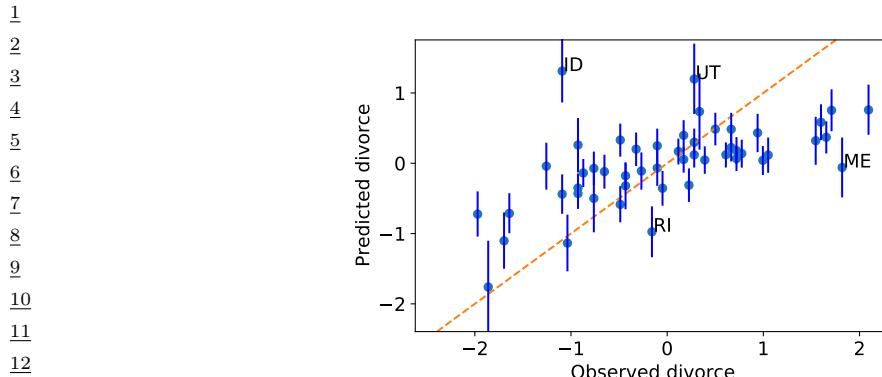


Figure 3.26: Posterior predictive distribution for divorce rate vs actual divorce rate for 50 US states. Both axes are standardized (i.e., z-scores). A few outliers are annotated. Adapted from Figure 5.5 of [McE20]. Generated by [linreg\\_divorce\\_ppc\\_numpyro.py](#).

### 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:

$$R(d|\mathbf{x}) \triangleq \mathbb{E}_{p(h|\mathbf{x})} [\ell(h, d)] = \sum_{h \in \mathcal{H}} \ell(h, d)p(h|\mathbf{x}) \quad (3.278)$$

The **optimal policy** (also called the **Bayes estimator**) specifies what action to take for each possible observation so as to minimize the risk:

$$\pi^*(\mathbf{x}) = \operatorname{argmin}_{d \in \mathcal{A}} \mathbb{E}_{p(h|\mathbf{x})} [\ell(h, d)] \quad (3.279)$$

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:

$$\pi^*(\mathbf{x}) = \operatorname{argmax}_{d \in \mathcal{A}} \mathbb{E}_h [U(h, d)] \quad (3.280)$$

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.<sup>10</sup>

10. The prior was chosen to match the prevalence in New York City in Spring 2020, and the likelihood corresponds to a PCR test. For details, see <https://nyti.ms/31MTZgV>.

	State	Nothing	Drugs
1	No COVID-19, young	0	8
2	COVID-19, young	60	8
3	No COVID-19, old	0	8
4	COVID-19, old	10	8
5			
6			

7    Table 3.2: Hypothetical loss matrix (in QALY units) for a decision maker, where there are 4 states of nature,  
8    and 2 possible actions.  
9

10	test	age	pr(covid)	cost-noop	cost-drugs	action
11	0	0	0.01	0.84	8.00	0
12	0	1	0.01	0.14	8.00	0
13	1	0	0.80	47.73	8.00	1
14	1	1	0.80	7.95	8.00	0
15						
16						

17    Table 3.3: Expected loss and corresponding optimal policy for treating COVID-19 patients for each possible  
18    observation.  
19

20

21

22    To specify the utility function, we need to pick some units. In economics, we usually use dollars,  
23    but in medical circles, it is more common to use a unit called **quality-adjusted life years** or  
24    **QALY**. Suppose that the expected QALY for a young person is 60, and for an old person is 10. Let  
25    us assume the drug costs the equivalent of 8 QALY, due to induced pain and suffering from side  
26    effects. Then we get the loss matrix shown in Table 3.2.<sup>11</sup>  
27

28    We have now fully specified the model. We can compute the belief state  $p(h|x) = p(\text{covid}|x)$   
29    using Bayes rule,  $p(h = (c, a)|x) \propto p(c)p(x|c)$ , where the age  $a$  is observed. Given this belief state,  
30    and the loss matrix in Table 3.2, we can compute the expected loss for each possible observation, as  
31    shown in Table 3.3. (See [dtheory.ipynb](#) for the code.) For this we see that the optimal policy is to  
32    only give the drug to young people who test positive. If, however, we reduce the cost of the drug  
33    from 8 units to 5, then the optimal policy changes: in this case, we should give the drug to everyone  
34    who tests positive. The policy can also change depending on the reliability of the test, which affects  
35    the confidence in our diagnosis.  
36

### 37    3.8.3 One-shot decision problems 38

39    In machine learning, there are several canonical kinds of **one-shot decision problems**, where we  
40    get an input  $x$  and need to make a decision. Often this decision corresponds to predicting an output  
41    from some set,  $y \in \mathcal{Y}$ . The optimal prediction will depend on the loss function, as we illustrate below.  
42

43    <sup>11</sup> These numbers reflect relative costs and benefits, and will depend on many factors. The numbers can be  
44    derived by asking the decision maker about their **preferences** about different possible outcomes. It is a theorem  
45    of decision theory that any consistent set of preferences can be converted into an ordinal cost scale (see e.g.,  
46    [https://en.wikipedia.org/wiki/Preference\\_\(economics\)](https://en.wikipedia.org/wiki/Preference_(economics))).  
47

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:

	$\hat{y} = 0$	$\hat{y} = 1$	
$y^* = 0$	0	1	
$y^* = 1$	1	0	

(3.281)

10 We can write this more concisely as follows:

$$\ell_{01}(y^*, \hat{y}) = \mathbb{I}(y^* \neq \hat{y}) \quad (3.282)$$

13 In this case, the posterior expected loss is

$$R(\hat{y}|\mathbf{x}) = p(\hat{y} \neq y^*|\mathbf{x}) = 1 - p(y^* = \hat{y}|\mathbf{x}) \quad (3.283)$$

16 Hence the action that minimizes the expected loss is to choose the most probable label:

$$\pi(\mathbf{x}) = \operatorname{argmax}_{y \in \mathcal{Y}} p(y|\mathbf{x}) \quad (3.284)$$

20 This corresponds to the **mode** of the posterior distribution, also known as the **maximum a  
21 posteriori or MAP estimate**.

22 We can generalize the loss function to associate different costs for false positives and false negatives.  
23 We can also allow for a “**reject action**”, in which the decision maker abstains from classifying when  
24 it is not sufficiently confident. See [Mur22, Sec 5.1.2.3] for details.

26 **3.8.3.2 L2 loss for regression**

27 Now suppose the hidden state of nature is a scalar  $h \in \mathbb{R}$ , and the corresponding action is also a  
28 scalar,  $y \in \mathbb{R}$ . The most common loss for continuous states and actions is the  **$\ell_2$  loss**, also called  
29 **squared error** or **quadratic loss**, which is defined as follows:

$$\ell_2(h, y) = (h - y)^2 \quad (3.285)$$

32 In this case, the risk is given by

$$R(y|\mathbf{x}) = \mathbb{E}[(h - y)^2|\mathbf{x}] = \mathbb{E}[h^2|\mathbf{x}] - 2y\mathbb{E}[h|\mathbf{x}] + a^2 \quad (3.286)$$

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:

$$\frac{\partial}{\partial y} R(y|\mathbf{x}) = -2\mathbb{E}[h|\mathbf{x}] + 2y = 0 \Rightarrow \pi(\mathbf{x}) = \mathbb{E}[h|\mathbf{x}] = \int h p(h|\mathbf{x}) dh \quad (3.287)$$

40 This is often called the **minimum mean squared error** estimate or **MMSE** estimate.

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).



# 4 Probabilistic graphical models

## 4.1 Introduction

I basically know of two principles for treating complicated systems in simple ways: the first is the principle of modularity and the second is the principle of abstraction. I am an apologist for computational probability in machine learning because I believe that probability theory implements these two principles in deep and intriguing ways — namely through factorization and through averaging. Exploiting these two mechanisms as fully as possible seems to me to be the way forward in machine learning. — Michael Jordan, 1997 (quoted in [Fre98]).

**Probabilistic graphical models (PGMs)** provide a convenient formalism for defining joint distributions on sets of random variables. In such graphs, the nodes represent random variables, and the (lack of) edges represent **conditional independence (CI)** assumptions between these variables. A better name for these models would be “independence diagrams”, but the term “graphical models” is now entrenched.

There are several kinds of graphical model, depending on whether the graph is directed, undirected, or some combination of directed and undirected, as we discuss in the sections below. More details on graphical models can be found in e.g., [KF09a].

## 4.2 Directed graphical models (Bayes nets)

In this section, we discuss **directed graphical models (DGM)**, which are based on **directed acyclic graphs** or **DAGs** (graphs that do not have any directed cycles). PGMs based on a DAG are often called **Bayesian networks** or **Bayes nets** for short; however, there is nothing inherently “Bayesian” about Bayesian networks: they are just a way of defining probability distributions. They are also sometimes called **belief networks**. The term “belief” here refers to subjective probability. However, the probabilities used in these models are no more (and no less) subjective than in any other kind of probabilistic model.

### 4.2.1 Representing the joint distribution

The key property of a DAG is that the nodes can be ordered such that parents come before children. This is called a **topological ordering**. Given such an order, we define the **ordered Markov property** to be the assumption that a node is conditionally independent of all its predecessors in

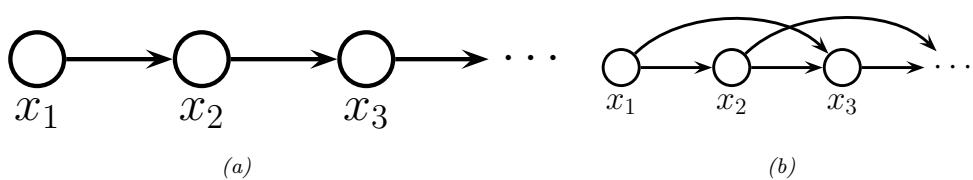


Figure 4.1: Illustration of first and second order Markov models.

the ordering given its parents, i.e.,

$$x_i \perp \mathbf{x}_{\text{pred}(i) \setminus \text{pa}(i)} | \mathbf{x}_{\text{pa}(i)} \quad (4.1)$$

where  $\text{pa}(i)$  are the parents of node  $i$ , and  $\text{pred}(i)$  are the predecessors of node  $i$  in the ordering. Consequently, we can represent the joint distribution as follows (assuming we use node ordering  $1 : V$ ):

$$p(\mathbf{x}_{1:V}) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)\dots p(x_V|x_1, \dots, x_{V-1}) \stackrel{*}{=} \prod_{i=1}^V p_{\theta_i}(x_i | \mathbf{x}_{\text{pa}(i)}) \quad (4.2)$$

where the equation marked  $*$  follows from the conditional independence assumptions, and where  $p_{\theta_i}(x_i | \mathbf{x}_{\text{pa}(i)})$  is the **conditional probability distribution** or **CPD** for node  $i$ .

The key advantage of the representation used in Equation (4.2) is that the number of parameters used to specify the joint distribution is substantially less, by virtue of the conditional independence assumptions that we have encoded in the graph, than an unstructured joint distribution. To see this, suppose all the variables are discrete and have  $K$  states each. Then an unstructured joint distribution needs  $O(K^V)$  parameters to specify the probability of every configuration. By contrast, with a DAG in which each node has at most  $P$  parents, we only need  $O(VK^{P+1})$  parameters, which can be exponentially fewer if the DAG is sparse.

We give some examples of DGMs in Section 4.2.2, and in Section 4.2.3, we discuss how to read off other conditional independence properties from the graph.

### 4.2.2 Examples

In this section, we give several examples of models that can be usefully represented as PGM-D's.

#### 4.2.2.1 Markov models

We can represent the conditional independence assumptions of a first-order Markov model using the chain-structured DGM shown in Figure 4.1(a). Consider a variable at a single time step  $t$ , which we call the “present”. From the diagram, we see that information cannot flow from the past,  $\mathbf{x}_{1:t-1}$ , to the future,  $\mathbf{x}_{t+1:T}$ , except via the present,  $\mathbf{x}_t$ . (We formalize this in Section 4.2.3.) This means that the  $\mathbf{x}_t$  is a sufficient statistic for the past, so the model is first-order Markov. This implies that the corresponding joint distribution can be written as follows:

$$p(\mathbf{x}_{1:T}) = p(\mathbf{x}_1)p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_3|\mathbf{x}_2)\dots p(\mathbf{x}_T|\mathbf{x}_{T-1}) \quad (4.3)$$

For discrete random variables, we can represent corresponding CPDs,  $p(x_t = k|x_{t-1} = j)$ , as a 2d table, known as a **conditional probability table** or **CPT**,  $p(x_t = k|x_{t-1} = j) = \theta_{jk}$ , where  $0 \leq \theta_{jk} \leq 1$  and  $\sum_{k=1}^K \theta_{jk} = 1$  (i.e., each row sums to 1).

The first-order Markov assumption is quite restrictive. If we want to allow for dependencies two steps into the past, we can create a Markov model of order 2. This is shown in Figure 4.1(b). The corresponding joint distribution has the form

$$p(\mathbf{x}_{1:T}) = p(\mathbf{x}_1, \mathbf{x}_2)p(\mathbf{x}_3|\mathbf{x}_1, \mathbf{x}_2)p(\mathbf{x}_4|\mathbf{x}_2, \mathbf{x}_3)\cdots p(\mathbf{x}_T|\mathbf{x}_{T-2}, \mathbf{x}_{T-1}) \quad (4.4)$$

As we increase the order of the Markov model, we need to add more edges. In the limit, the DAG becomes fully connected (subject to being acyclic), as shown in Figure 23.1. However, in this case, there are no useful conditional independencies, so the graphical model has no value.

#### 4.2.2.2 The “Student” network

Figure 4.2 shows a model for capturing the inter-dependencies between 5 discrete random variables related to a hypothetical student taking a class: D = difficulty of class (easy, hard), I = intelligence (low, high), G = grade (A, B, C), S = SAT score (bad, good), L = letter of recommendation (bad, good). (This is a simplification of the “**Student network**” from [KF09a, p.281].) The chain rule tells us that we can represent the joint as follows:

$$p(D, I, G, L, S) = p(L|S, G, D, I) \times p(S|G, D, I) \times p(G|D, I) \times p(D|I) \times p(I) \quad (4.5)$$

where we have ordered the nodes topologically as I, D, G, S, L. Note that L is conditionally independent of all the other nodes earlier in this ordering given its parent G, so we can replace  $p(L|S, G, D, I)$  by  $p(L|G)$ . We can simplify the other terms in a similar way to get

$$p(D, I, G, L, S) = p(L|G) \times p(S|I) \times p(G|D, I) \times p(D) \times p(I) \quad (4.6)$$

The ability to simplify a joint distribution in a product of small local pieces is the key idea behind graphical models.

In addition to the graph structure, we need to specify the conditional probability distributions (CPDs) at each node. For discrete random variables, we can represent the CPD as a table, which means we have a separate row (i.e., a separate categorical distribution) for each **conditioning case**, i.e., for each combination of parent values. This is known as a **conditional probability table** or **CPT**. We can represent the  $i$ 'th CPT as a tensor

$$\theta_{ijk} \triangleq p(x_i = k|x_{\text{pa}(i)} = j) \quad (4.7)$$

Thus  $\theta_i$  is a **row stochastic matrix**, that satisfies the properties  $0 \leq \theta_{ijk} \leq 1$  and  $\sum_{k=1}^{K_i} \theta_{ijk} = 1$  for each row  $j$ . Here  $i$  indexes nodes,  $i \in [V]$ ;  $k$  indexes node states,  $k \in [K_i]$ , where  $K_i$  is the number of states for node  $i$ ; and  $j$  indexes joint parent states,  $j \in [J_i]$ , where  $J_i = \prod_{p \in \text{pa}(i)} K_p$ .

The CPTs for the student network are shown next to each node in Figure 4.2. For example, we see that if the class is hard ( $D = 1$ ) and the student is dumb ( $I = 0$ ), the distribution over grades A, B and C we expect is  $p(G|D = 1, I = 0) = [0.05, 0.25, 0.7]$ ; but if the student is intelligent, we get  $p(G|D = 1, I = 1) = [0.5, 0.3, 0.2]$ .

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47

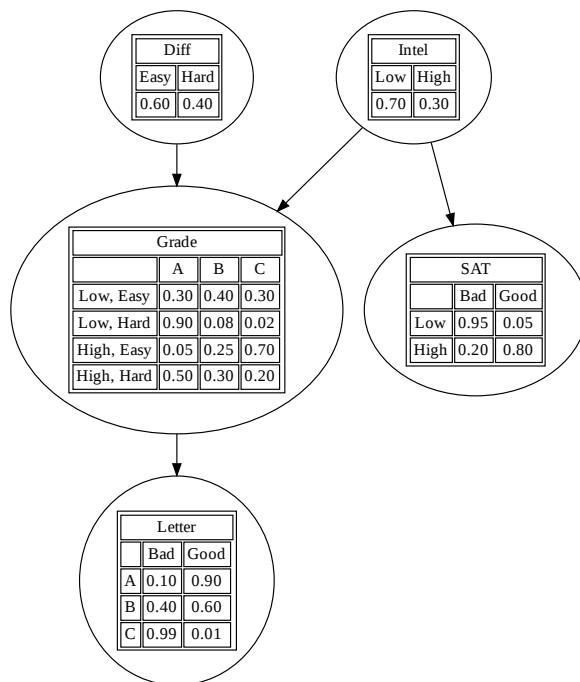


Figure 4.2: The (simplified) student network. “Diff” is the difficulty of the class. “Intel” is the intelligence of the student. “Grade” is the grade of the student in this class. “SAT” is the score of the student on the SAT exam. “Letter” is whether the teacher writes a good or bad letter of recommendation. The circles (nodes) represent random variables, the edges represent direct probabilistic dependencies. The tables inside each node represent the conditional probability distribution of the node given its parents. Generated by [student\\_pgm.ipynb](#).

The number of parameters in a CPT is  $O(K^{p+1})$ , where  $K$  is the number of states per node, and  $p$  is the number of parents. Later we will consider more parsimonious representations, with fewer learnable parameters. (We discuss parameter learning in Section 4.2.6.)

Once we have specified the model, we can use it to answer probabilistic queries, as we discuss in Section 4.2.5. As an example, suppose we observe that the student gets a grade of C. The posterior probability that the student is intelligent is just  $p(I = \text{Intelligent} | G = C) = 0.08$ , since it is more likely that the low grade is explained by the class being hard (indeed,  $p(D = H | G = C) = 0.63$ ). However, now suppose we also observe that the student gets a good SAT score. Now the posterior probability that the student is intelligent has jumped to  $p(I = \text{Intelligent} | G = C, \text{SAT} = \text{Good}) = 0.58$ , and the probability that the class is hard has changed to  $p(D = \text{Hard} | G = C, \text{SAT} = \text{Good}) = 0.76$ , as shown in Figure 4.7. This negative mutual interaction between multiple causes of some observations is called the **explaining away** effect, also known as **Berkson’s paradox** (see Section 4.2.3.2 for details).

1

### 4.2.2.3 Gaussian Bayes nets

2

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

4

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

5

As we show below, multiplying all these CPDs together results in a large joint Gaussian distribution of  
6 the form  $p(\mathbf{x}) = \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}, \boldsymbol{\Sigma})$ , where  $\mathbf{x} \in \mathbb{R}^V$ . This is called a **directed Gaussian graphical model**  
7 or a **Gaussian Bayes net**.

8

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

10

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

11

where  $z_i \sim \mathcal{N}(0, 1)$ ,  $\sigma_i$  is the conditional standard deviation of  $x_i$  given its parents,  $w_{i,j}$  is the strength  
12 of the  $j \rightarrow i$  edge, and  $\mu_i$  is the local mean.<sup>1</sup>

13

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

16

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

17

where  $\mathbf{W}$  is the matrix of regression weights. Now let  $\mathbf{e}$  be a vector of noise terms:  $\mathbf{e} \triangleq \mathbf{S}\mathbf{z}$ . We can  
18 rearrange this to get  $\mathbf{e} = (\mathbf{I} - \mathbf{W})(\mathbf{x} - \boldsymbol{\mu})$ . Since  $\mathbf{W}$  is lower triangular (because  $w_{j,i} = 0$  if  $j < i$  in  
19 the topological ordering), we have that  $\mathbf{I} - \mathbf{W}$  is lower triangular with 1s on the diagonal. Hence

20

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

21

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

22

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

23

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

24

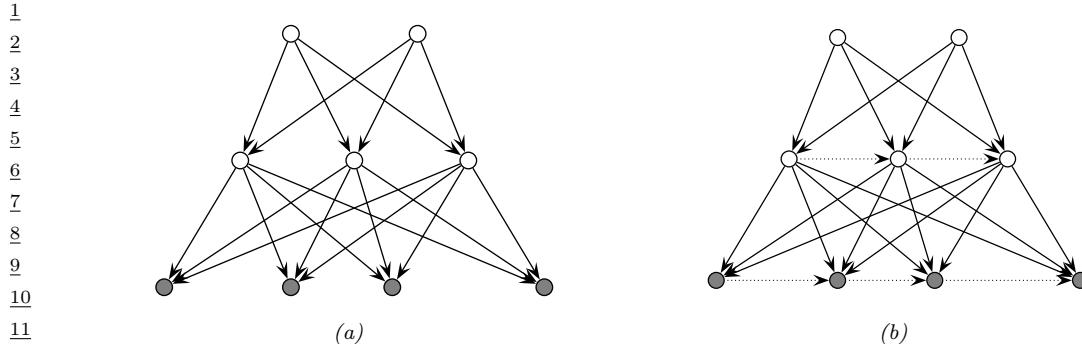
$$\boldsymbol{\Sigma} = \text{Cov}[\mathbf{x}] = \text{Cov}[\mathbf{x} - \boldsymbol{\mu}] \quad (4.13)$$

25

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

26

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



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

#### 4.2.2.4 Sigmoid belief nets

In this section, we consider a **deep generative model** of the form shown in Figure 4.3a. This corresponds to the following joint distribution:

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{z}_2)p(\mathbf{z}_1|\mathbf{z}_2)p(\mathbf{x}|\mathbf{z}_1) = \prod_{k=1}^{K_2} p(z_{2,k}) \prod_{k=1}^{K_1} p(z_{1,k}|\mathbf{z}_2) \prod_{d=1}^D p(x_d|\mathbf{z}_1) \quad (4.15)$$

where the  $\mathbf{x}$  nodes are the leaves the the  $\mathbf{z}_\ell$  nodes are the internal hidden nodes. (We assume there are  $K_\ell$  hidden nodes at level  $\ell$ , 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  $\Sigma$  is much messier, as can be seen by looking at [Bis06, p370].

We can also add directed connections between the hidden variables within a layer, as shown in Figure 4.3b. This is called a **deep autoregressive network** or **DARN** model [Gre+14], which combines ideas from latent variable modeling and autoregressive modeling.

We discuss other forms of hierarchical generative models in Chapter 22.

### 4.2.3 Conditional independence properties

In this section, we discuss the CI properties of a DAG. That is, we discuss how to determine if  $\mathbf{X}_A \perp_G \mathbf{X}_B | \mathbf{X}_C$  is entailed by the graph  $G$ , where  $\mathbf{X}_A$ ,  $\mathbf{x}_B$  and  $\mathbf{X}_C$  are sets of variables. Intuitively one might guess that it is sufficient to see if  $A$  and  $B$  are separated or not after we remove nodes in  $C$  from the graph (i.e., if  $C$  blocks the paths between  $A$  and  $B$ ). This intuition is correct in the case of undirected graphs (see Section 4.3.3), but for DAGs, we need to pay attention to the orientation of the edges, as we explain below.

#### 4.2.3.1 Global Markov properties (d-separation)

First, we introduce some definitions. We say an *undirected path*  $P$  is **d-separated** by a set of nodes  $E$  (containing the evidence) iff at least one of the following conditions hold:

1.  $P$  contains a chain or **pipe**,  $s \rightarrow m \rightarrow t$  or  $s \leftarrow m \leftarrow t$ , where  $m \in E$
2.  $P$  contains a tent or **fork**,  $s \swarrow^m \searrow t$ , where  $m \in E$
3.  $P$  contains a **collider** or **v-structure**,  $s \searrow_m \swarrow t$ , where  $m$  is not in  $E$  and neither is any descendant of  $m$ .

Next, we say that a *set of nodes*  $A$  is d-separated from a different set of nodes  $B$  given a third observed set  $E$  iff each undirected path from every node  $a \in A$  to every node  $b \in B$  is d-separated by  $E$ . Finally, we define the CI properties of a DAG as follows:

$$\mathbf{X}_A \perp_G \mathbf{X}_B | \mathbf{X}_E \iff A \text{ is d-separated from } B \text{ given } E \quad (4.18)$$

This is called the (directed) **global Markov property**.

The **Bayes ball algorithm** [Sha98] is a simple way to see if  $A$  is d-separated from  $B$  given  $E$ , based on the above definition. The idea is this. We “shade” all nodes in  $E$ , indicating that they are observed. We then place “balls” at each node in  $A$ , let them “bounce around” according to some rules, and then ask if any of the balls reach any of the nodes in  $B$ . The three main rules are shown in Figure 4.4. Notice that balls can travel opposite to edge directions. We see that a ball can pass through a chain, but not if it is shaded in the middle. Similarly, a ball can pass through a fork, but not if it is shaded in the middle. However, a ball cannot pass through a v-structure, unless it is shaded in the middle.

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

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

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

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

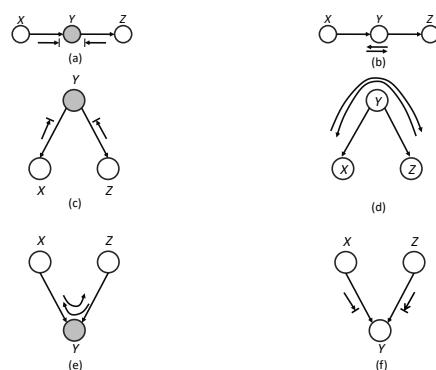


Figure 4.4: Bayes ball rules. A shaded node is one we condition on. If there is an arrow hitting a bar, it means the ball cannot pass through; otherwise the ball can pass through.

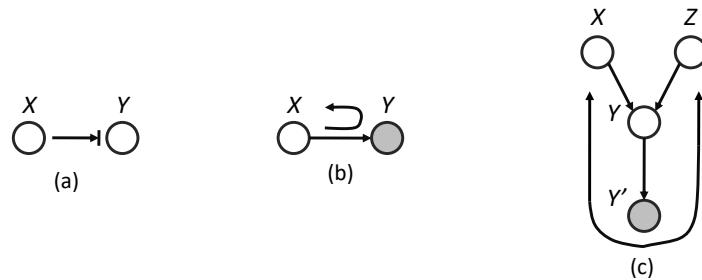


Figure 4.5: (a-b) Bayes ball boundary conditions. (c) Example of why we need boundary conditions.  $y'$  is an observed child of  $y$ , rendering  $y$  “effectively observed”, so the ball bounces back up on its way from  $x$  to  $z$ .

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

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

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

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

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

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

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

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

	X	Y	Z
1	D	I	
2	D	I	S
3	D	S	
4	D	S	I
5	D	S	L, I
6	D	S	G, I
7	D	S	G, L, I
8	D	L	G
9	D	L	G, S
10	D	L	G, I
11	D	L	I, G, S
12			
13			
14			

Table 4.1: Conditional independence relationships implied by the student DAG (Figure 4.2). Each line has the form  $X \perp Y|Z$ . Generated by [student\\_pgmpy.ipynb](#).

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

$$p(x, z|y) = \frac{p(x)p(z)p(y|x, z)}{p(y)} \quad (4.24)$$

so  $X \not\perp Z|Y$ . However, in the unconditional distribution, we have

$$p(x, z) = p(x)p(z) \quad (4.25)$$

so we see that  $X$  and  $Z$  are marginally independent. So we see that conditioning on a common child at the bottom of a v-structure makes its parents become dependent. This important effect is called **explaining away**, **inter-causal reasoning**, or **Berkson's paradox** (see Section 4.2.3.2 for a discussion).

Finally, Bayes Ball also needs the “boundary conditions” shown in Figure 4.5(a-b). These rules say that a ball hitting a hidden leaf stops, but a ball hitting an observed leaf “bounces back”. To understand where this rule comes from, consider Figure 4.5(c). Suppose  $Y'$  is a (possibly noisy) copy of  $Y$ . If we observe  $Y'$ , we effectively observe  $Y$  as well, so the parents  $X$  and  $Z$  have to compete to explain this. So if we send a ball down  $X \rightarrow Y \rightarrow Y'$ , it should “bounce back” up along  $Y' \rightarrow Y \rightarrow Z$ , in order to pass information between the parents. However, if  $Y$  and *all* its children are hidden, the ball does not bounce back.

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

#### 4.2.3.2 Explaining away (Berkson's paradox)

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

As a simple example (from [PM18b, p198]), consider tossing two coins 100 times. Suppose you only record the outcome of the experiment if at least one coin shows up heads. You should expect

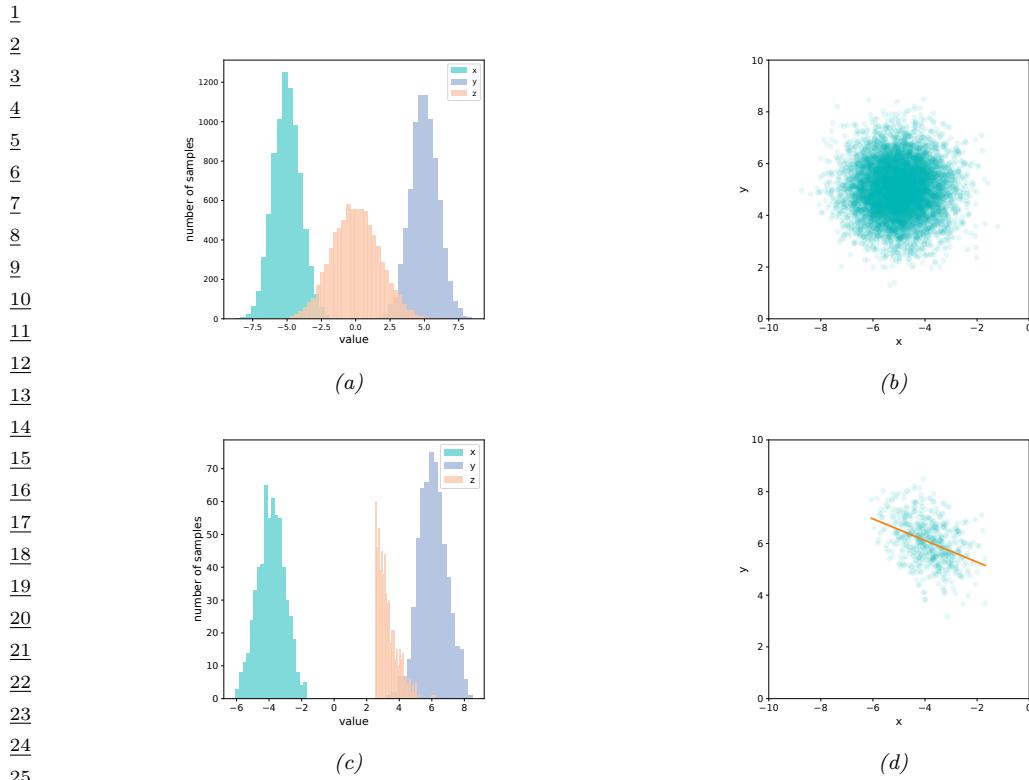


Figure 4.6: Samples from a jointly Gaussian DGM,  $p(x, y, z) = \mathcal{N}(x| -5, 1)\mathcal{N}(y|5, 1)\mathcal{N}(z|x + y, 1)$ . (a) Unconditional marginal distributions,  $p(x)$ ,  $p(y)$ ,  $p(z)$ . (b) Unconditional joint distribution,  $p(x, y)$ . (c) Conditional marginal distribution,  $p(x|z > 2.5)$ ,  $p(y|z > 2.5)$ ,  $p(z|z > 2.5)$ . (d) Conditional joint distribution,  $p(x, y|z > 2.5)$ . Adapted from [Clo20]. Generated by `berksons_gaussian.py`.

30

31

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

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

$$39 \quad p(x, y, z) = \mathcal{N}(x| -5, 1)\mathcal{N}(y|5, 1)\mathcal{N}(z|x + y, 1) \quad (4.26)$$

40

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

47

1

### 4.2.3.3 Other Markov properties

2 From the d-separation criterion, one can conclude that

3

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

4 where the **non-descendants** of a node  $\text{nd}(t)$  are all the nodes except for its descendants,  $\text{nd}(t) =$   
5  $\{1, \dots, D\} \setminus \{t \cup \text{desc}(t)\}$ . Equation (4.27) is called the (directed) **local Markov property**. For  
6 example, in Figure 4.20(a), we have  $\text{nd}(3) = \{1, 2, 4\}$ , and  $\text{pa}(3) = 1$ , so  $3 \perp 2, 4 | 1$ .

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

9

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

10 which follows since  $\text{pred}(t) \subseteq \text{nd}(t)$ . This is called the **ordered Markov property**, which justifies  
11 Equation (4.2). For example, in Figure 4.20(a), if we use the ordering  $1, 2, \dots, 7$ , we find  $\text{pred}(3) =$   
12  $\{1, 2\}$  and  $\text{pa}(3) = 1$ , so  $3 \perp 2 | 1$ .

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

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

21

### 4.2.3.4 Markov blankets and full conditionals

22 The smallest set of nodes that renders a node  $t$  conditionally independent of all the other nodes  
23 in the graph is called  $t$ 's **Markov blanket**; we will denote this by  $\text{mb}(t)$ . One can show that the  
24 Markov blanket of a node in a DGM is equal to the parents, the children, and the **co-parents**, i.e.,  
25 other nodes who are also parents of its children:

26

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

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

28

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

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

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

33

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

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

35

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

1

### 2 4.2.3.5 I-maps

3 We have discussed how to “read off” the CI statements encoded by a graph  $G$ . In this section, we  
4 discuss how this relates to the CI properties of a distribution  $p$ .

5 We will write  $\mathbf{x}_A \perp_G \mathbf{x}_B | \mathbf{x}_C$  if  $A$  is independent of  $B$  given  $C$  in the graph  $G$ . Let  $I(G)$  be the set  
6 of all such CI statements encoded by the graph. We say that  $G$  is an **I-map** (independence map)  
7 for  $p$ , or that  $p$  is **Markov** wrt  $G$ , iff  $I(G) \subseteq I(p)$ , where  $I(p)$  is the set of all CI statements that  
8 hold for distribution  $p$ . In other words, the graph is an I-map if it does not make any assertions of  
9 CI that are not true of the distribution. This allows us to use the graph as a safe proxy for  $p$  when  
10 reasoning about  $p$ ’s CI properties. This is helpful for designing algorithms that work for large classes  
11 of distributions, regardless of their specific numerical parameters. Note that the fully connected  
12 graph is an I-map of all distributions, since it makes no CI assertions at all (since it is not missing  
13 any edges). We therefore say  $G$  is a **minimal I-map** of  $p$  if  $G$  is an I-map of  $p$ , and if there is no  
14  $G' \subseteq G$  which is an I-map of  $p$ .

15

### 16 4.2.4 Generation (sampling)

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

### 23 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**,  $R = \{1, \dots, V\} \setminus \{Q, V\}$ :

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

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:

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

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 simpifies to

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

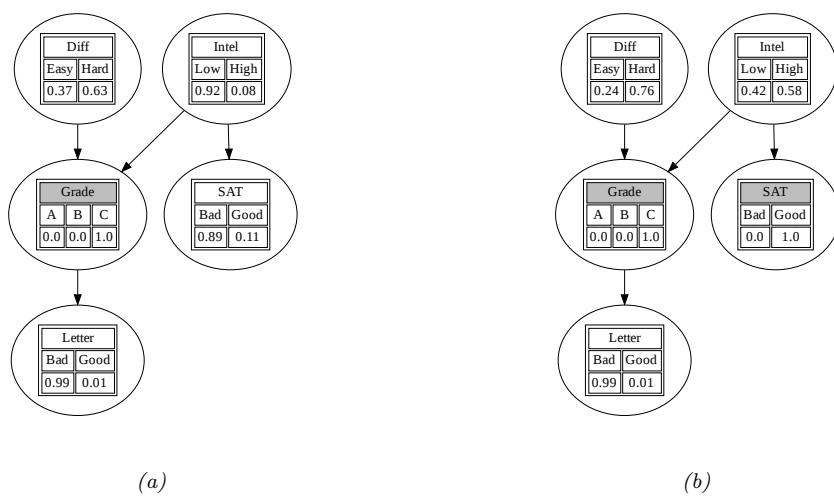


Figure 4.7: Illustration of belief updating in the “Student” PGM. The histograms show the marginal distribution of each node. Nodes with shaded titles are clamped to an observed value. (a) Posterior after conditioning on Grade=C. (b) Posterior after also conditioning on SAT=Good. Generated by [student\\_pgmpy.ipynb](#).

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

#### 4.2.5.1 Example: inference in the Student network

As an example of inference in PGMs, consider the Student network from Section 4.2.2. Suppose we observe that the student gets a grade of C. The posterior marginals are shown in Figure 4.7a. We see that the low grade could be explained by the class being hard (since  $p(D = \text{Hard}|G = C) = 0.63$ ) 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.)

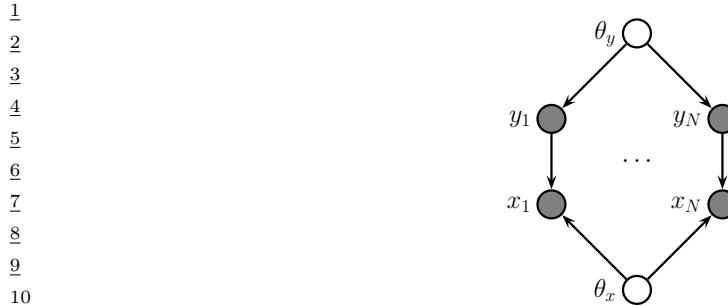


Figure 4.8: A PGM-D representing the joint distribution  $p(\mathbf{y}_{1:N}, \mathbf{x}_{1:N}, \boldsymbol{\theta}_y, \boldsymbol{\theta}_x)$ . Here  $\boldsymbol{\theta}_x$  and  $\boldsymbol{\theta}_y$  are global parameter nodes that are shared across the examples, whereas  $\mathbf{x}_n$  and  $\mathbf{y}_n$  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  $\mathbf{y}_n$ , and 2 **global variables**, corresponding to the parameters, which are shared across data samples. The local variables are observed (in the training set), so they are represented by solid (shaded) nodes. The global variables are not observed, and hence are represented by empty (unshaded) nodes. (The model represents a generative classifier, so the edge is from  $\mathbf{y}_n$  to  $\mathbf{x}_n$ ; if we are fitting a discriminative classifier, the edge would be from  $\mathbf{x}_n$  to  $\mathbf{y}_n$ , and there would be no  $\boldsymbol{\theta}_y$  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:

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

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

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

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

1 data that is sufficient for  $\theta_x$ .

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

$$\underline{5} \quad p(\boldsymbol{\theta}, \mathcal{D}) = \prod_{i=1}^V p(\boldsymbol{\theta}_i) p(\mathcal{D}_i | \boldsymbol{\theta}_i) \quad (4.39)$$

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

$$\underline{8} \quad \hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{i=1}^V p(\mathcal{D}_i | \boldsymbol{\theta}_i) \quad (4.40)$$

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

#### 10 4.2.6.2 Example: computing the MLE for CPTs

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

$$\underline{13} \quad p(\mathcal{D} | \boldsymbol{\theta}) = \prod_{n=1}^N \prod_{i=1}^V p(x_{ni} | \mathbf{x}_{n,\text{pa}(i)}, \boldsymbol{\theta}_i) \quad (4.41)$$

$$\underline{14} \quad = \prod_{n=1}^N \prod_{i=1}^V \prod_{j=1}^{J_i} \prod_{k=1}^{K_i} \theta_{ijk}^{\mathbb{I}(x_{ni}=k, \mathbf{x}_{n,\text{pa}(i)}=j)} \quad (4.42)$$

15 where

$$\underline{16} \quad \theta_{ijk} \triangleq p(x_i = k | \mathbf{x}_{\text{pa}(i)} = j) \quad (4.43)$$

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

$$\underline{19} \quad N_{ijk} \triangleq \sum_{n=1}^N \mathbb{I}(x_{n,i} = k, x_{n,\text{pa}(i)} = j) \quad (4.44)$$

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

$$\underline{21} \quad \hat{\theta}_{ijk} = \frac{N_{ijk}}{\sum_{k'} N_{ijk'}} \quad (4.45)$$

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

26

	I	D	G	S	L
1	0	0	2	0	0
2	0	1	2	0	0
3	0	0	1	1	1
4	1	1	1	1	0
5	1	0	0	1	1
6	0	0	0	0	1
7	1	1	2	1	1
8					
9					
10					

Table 4.2: Some fully observed training data for the student network.

I	D	$N_{i,j,k}$	$\hat{\theta}_{i,j,k}$	$\bar{\theta}_{i,j,k}$
0	0	[1, 1, 1]	[ $\frac{1}{3}, \frac{1}{3}, \frac{1}{3}$ ]	[ $\frac{2}{6}, \frac{2}{6}, \frac{2}{6}$ ]
0	1	[0, 0, 1]	[ $\frac{0}{4}, \frac{0}{4}, \frac{1}{4}$ ]	[ $\frac{1}{4}, \frac{1}{4}, \frac{4}{4}$ ]
1	0	[1, 0, 0]	[ $\frac{1}{4}, \frac{0}{4}, \frac{0}{4}$ ]	[ $\frac{2}{4}, \frac{1}{4}, \frac{1}{4}$ ]
1	1	[0, 1, 1]	[ $0, \frac{1}{2}, \frac{1}{2}$ ]	[ $\frac{1}{5}, \frac{2}{5}, \frac{2}{5}$ ]

Table 4.3: Sufficient statistics  $N_{ijk}$  and corresponding MLE  $\hat{\theta}_{ijk}$  and posterior mean  $\bar{\theta}_{ijk}$  for node  $i = G$  in the student network. Each row corresponds to a different joint configuration of its parent nodes, corresponding to state  $j$ . The index  $k$  refers to the 3 possible values of the child node  $G$ .

In Table 4.3, we list the sufficient statistics  $N_{ijk}$  and the MLE  $\hat{\theta}_{ijk}$  for node  $i = G$ , with parents  $(I, D)$ . A similar process can be used for the other nodes. Thus we see that fitting a DGM with tabular CPDs reduces to a simple counting problem.

However, we notice there are a lot of zeros in the sufficient statistics, due to the small sample size, resulting in extreme estimates for some of the probabilities  $\hat{\theta}_{ijk}$ . We discuss a (Bayesian) solution to this in Section 4.2.6.3.

### 4.2.6.3 Example: Computing the posterior for CPTs

In Section 4.2.6.2 we discussed how to compute the MLE for the CPTs in a discrete Bayes net. We also observed that this can suffer from the zero-count problem. In this section, we show how a Bayesian approach can solve this problem.

Let us put a separate Dirichlet prior on each row of each CPT, i.e.,  $\theta_{ij} \sim \text{Dir}(\alpha_{ij})$ . Then we can compute the posterior by simply adding the pseudo counts to the empirical counts to get  $\theta_{ij} | \mathcal{D} \sim \text{Dir}(N_{ij} + \alpha_{ij})$ , where  $N_{ijk}$  is the number of times that node  $i$  is in state  $k$  while its parents are in state  $j$ . Hence the posterior mean estimate is given by

$$\bar{\theta}_{ijk} = \frac{N_{ijk} + \alpha_{ijk}}{\sum_{k'} (N_{ijk'} + \alpha_{ijk'})} \quad (4.46)$$

The MAP estimate has the same form, except we use  $\alpha_{ijk} - 1$  instead of  $\alpha_{ijk}$ .

In Table 4.3, we illustrate this approach applied to the  $G$  node in the student netowkr, where we use a uniform Dirichlet prior,  $\alpha_{ijk} = 1$ .

47

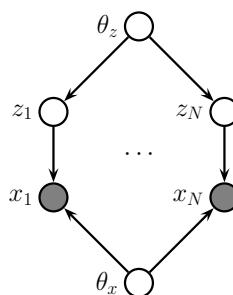


Figure 4.9: A PGM-D representing the joint distribution  $p(\mathbf{z}_{1:N}, \mathbf{x}_{1:N}, \boldsymbol{\theta}_z, \boldsymbol{\theta}_x)$ . The local variables  $\mathbf{z}_n$  are hidden, whereas  $\mathbf{x}_n$  are observed. This is typical for learning unsupervised latent variable models.

#### 4.2.6.4 Learning from incomplete data

In Section 4.2.6.1, we explained that when we have complete data, the likelihood (and posterior) factorizes over CPDs, so we can estimate each CPD independently. Unfortunately, this is no longer the case when we have incomplete or missing data. To see this, consider Figure 4.9. The likelihood of the observed data can be written as follows:

$$p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{\mathbf{z}_{1:N}} \left[ \prod_{n=1}^N p(\mathbf{z}_n|\boldsymbol{\theta}_z) p(\mathbf{x}_n|\mathbf{z}_n, \boldsymbol{\theta}_x) \right] \quad (4.47)$$

$$= \prod_{n=1}^N \sum_{\mathbf{z}_n} p(\mathbf{z}_n|\boldsymbol{\theta}_z) p(\mathbf{x}_n|\mathbf{z}_n, \boldsymbol{\theta}_x) \quad (4.48)$$

Thus the log likelihood is given by

$$LL(\boldsymbol{\theta}) = \sum_n \log \sum_{\mathbf{z}_n} p(\mathbf{z}_n|\boldsymbol{\theta}_z) p(\mathbf{x}_n|\mathbf{z}_n, \boldsymbol{\theta}_x) \quad (4.49)$$

The log function does not distribute over the  $\sum_{\mathbf{z}_n}$  operation, so the objective does not decompose over nodes.<sup>2</sup> Consequently, we can no longer compute the MLE or the posterior by solving separate problems per node.

To solve this, we will resort to optimization methods. (We focus on the MLE case, and leave discussion of Bayesian inference for latent variable models to Part II.) In this section below, we discuss how to use EM and SGD to find a local optimum of the (non convex) log likelihood objective.

#### 4.2.6.5 Using EM to fit CPTs in the incomplete data case

A popular method for estimating the parameters of a PGM-D in the presence of missing data is to use the EM algorithm, as proposed in [Lau95]. We describe EM in detail in Section 6.7.3,

<sup>2</sup> We can also see this from the graphical model:  $\boldsymbol{\theta}_x$  is no longer independent of  $\boldsymbol{\theta}_z$ , because there is a path that connects them via the hidden nodes  $\mathbf{z}_n$ . (See Section 4.2.3 for an explanation of how to “read off” such CI properties from a DGM.)

1 but the basic idea is to alternate between inferring the latent variables  $\mathbf{z}_n$  (the E or expectation  
 2 step), and estimating the parameters given this completed dataset (the M or maximization step).  
 3 Rather than returning the full posterior  $p(\mathbf{z}_n|\mathbf{x}_n, \boldsymbol{\theta}^{(t)})$  in the E step, we instead return the expected  
 4 sufficient statistics (ESS), which takes much less space. In the M step, we maximize the expected  
 5 value of the log likelihood of the fully observed data using these ESS.  
 6

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

$$9 \quad 10 \quad \log p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{i=1}^V \sum_{j=1}^{J_i} \sum_{k=1}^{K_i} N_{ijk} \log \theta_{ijk} \quad (4.50)$$

12 and hence the expected complete data log-likelihood has the form  
 13

$$14 \quad 15 \quad \mathbb{E} [\log p(\mathcal{D}|\boldsymbol{\theta})] = \sum_i \sum_j \sum_k \bar{N}_{ijk} \log \theta_{ijk} \quad (4.51)$$

16 where

$$18 \quad 19 \quad \bar{N}_{ijk} = \sum_{n=1}^N \mathbb{E} [\mathbb{I}(x_{ni} = k, \mathbf{x}_{n,\text{pa}(i)} = j)] = \sum_{n=1}^N p(x_{ni} = k, \mathbf{x}_{n,\text{pa}(i)} = j | \mathcal{D}_n, \boldsymbol{\theta}^{\text{old}}) \quad (4.52)$$

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

25 Given these ESS, the M step has the simple form

$$27 \quad 28 \quad \hat{\theta}_{ijk} = \frac{\bar{N}_{ijk}}{\sum_{k'} \bar{N}_{ijk'}} \quad (4.53)$$

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

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

34

### 35 4.2.6.6 Using SGD to fit CPTs in the incomplete data case

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

$$40 \quad 41 \quad p(\mathbf{x}_n|\boldsymbol{\theta}) = \sum_{\mathbf{z}_n} p(\mathbf{z}_n|\boldsymbol{\theta}_z) p(\mathbf{x}_n|\mathbf{z}_n, \boldsymbol{\theta}_x) \quad (4.54)$$

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

$$45 \quad 46 \quad LL(\boldsymbol{\theta}) = \sum_n \log p(\mathbf{x}_n|\boldsymbol{\theta}) \quad (4.55)$$

47

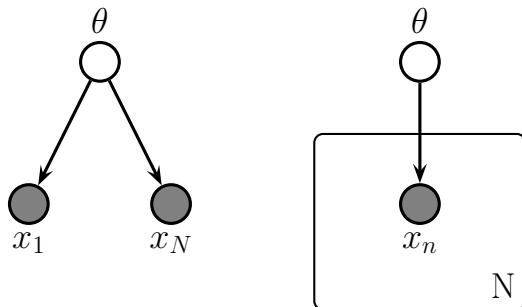


Figure 4.10: Left: data points  $\mathbf{x}_n$  are conditionally independent given  $\theta$ . Right: Same model, using plate notation. This represents the same model as the one on the left, except the repeated  $\mathbf{x}_n$  nodes are inside a box, known as a plate; the number in the lower right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

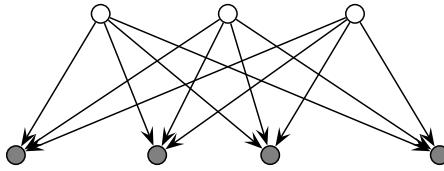
### 4.2.7 Plate notation

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  $\theta$ . We denote this by

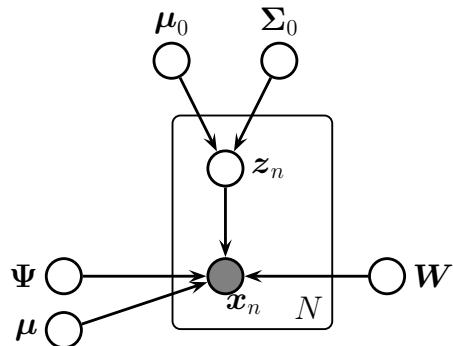
$$\mathbf{x}_n \sim p(\mathbf{x} | \theta) \quad (4.60)$$

The corresponding joint distribution over the parameters and data has the form

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



(a)



(b)

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

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

$$p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{n=1}^N p(\mathbf{x}_n|\boldsymbol{\theta}) \quad (4.62)$$

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  $\mathbf{x}$  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

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

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) \quad (4.63)$$

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

where  $\mathbf{W}$  is a  $D \times L$  matrix, known as the factor loading matrix, and  $\boldsymbol{\Psi}$  is a diagonal  $D \times D$  covariance matrix.

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

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

47

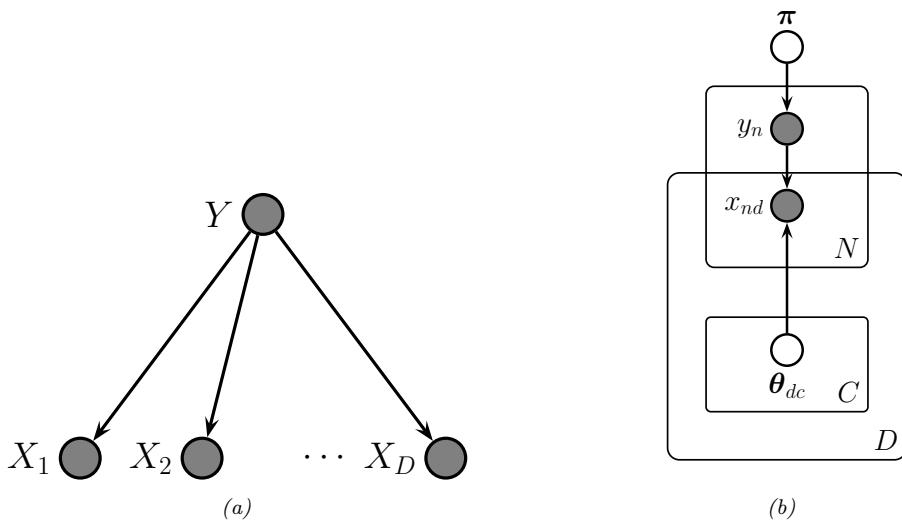


Figure 4.12: (a) Naive Bayes classifier as a PGM-D. (b) Model augmented with plate notation.

#### 4.2.7.2 Example: Naive Bayes classifier

In some models, we have doubly indexed variables. For example, consider a **naive Bayes classifier**. This is a simple generative classifier, defined as follows:

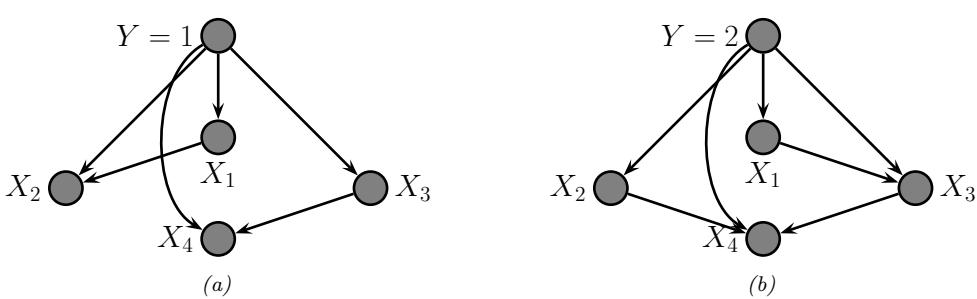
$$p(\mathbf{x}, y|\boldsymbol{\theta}) = p(y|\boldsymbol{\pi}) \prod_{d=1}^D p(x_d|y, \boldsymbol{\theta}_d) \quad (4.65)$$

The fact that the features  $\mathbf{x}_{1:D}$  are considered conditionally independent given the class label  $y$  is where the term “naive” comes from. Nevertheless, this model often works surprisingly well, and is extremely easy to fit.

We can represent the conditional independence assumption as shown in Figure 4.12a. We can represent the repetition over the dimension  $d$  with a plate. When we turn to inferring the parameters  $\boldsymbol{\theta} = (\boldsymbol{\pi}, \boldsymbol{\theta}_{1:D, 1:C})$ , we also need to represent the repetition over data cases  $n$ . This is shown in Figure 4.12b. Note that the parameter  $\boldsymbol{\theta}_{dc}$  depends on  $d$  and  $c$ , whereas the feature  $x_{nd}$  depends on  $n$  and  $d$ . This is shown using **nested plates** to represent the shared  $d$  index.

#### 4.2.7.3 Relaxing the naive Bayes assumption

We see from Figure 4.12a that the observed features are conditionally independent given the class label. We can of course allow for dependencies between the features, as illustrated in Figure 4.13. (We omit parameter nodes for simplicity.) If we enforce that the edges between the features forms a tree the model is known as a **tree-augmented naive Bayes classifier** [FGG97], or **TAN** model. (Trees are a restricted form of graphical that have various computational advantages that we discuss later.) Note that the topology of the tree can change depending on the value of the class node  $y$ ;



*Figure 4.13: Tree-augmented naive Bayes classifier for  $D = 4$  features. The tree topology can change depending on the value of  $u$ , as illustrated.*

<sup>15</sup> in this case, the model is known as a **Bayesian multi net**, and can be thought of as a supervised  
<sup>16</sup> mixture of trees.

### 4.3 Undirected graphical models (Markov random fields)

Directed graphical models (Section 4.2) are very useful. However, for some domains, being forced to choose a direction for the edges, as required by a DAG, is rather awkward. For example, consider modeling an image. It is reasonable to assume that the intensity values of neighboring pixels are correlated. We can model this using a DAG with a 2d lattice topology as shown in Figure 4.14(a). This is known as a **Markov mesh** [AHK65]. However, its conditional independence properties are rather unnatural.

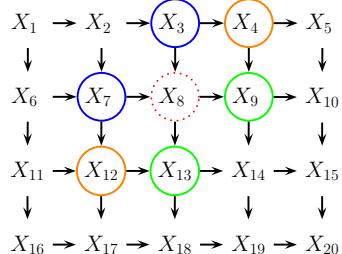
An alternative is to use an **undirected graphical model (UGM)**, also called a **Markov random field (MRF)** or **Markov network**. These do not require us to specify edge orientations, and are much more natural for some problems such as image analysis and spatial statistics. For example, an undirected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

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

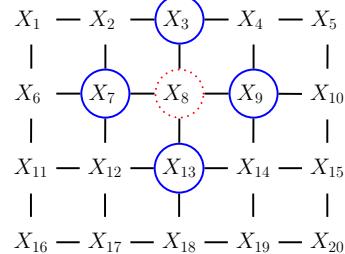
#### **4.3.1 Representing the joint distribution**

<sup>41</sup> Since there is no topological ordering associated with an undirected graph, we can't use the chain  
<sup>42</sup> rule to represent  $p(\mathbf{x}_{1:V})$ . So instead of associating CPDs with each node, we associate **potential**  
<sup>43</sup> **functions** or **factors** with each **maximal clique** in the graph.<sup>3</sup> We will denote the potential  
<sup>44</sup>

<sup>45</sup> 3. A **clique** is a set of nodes that are all neighbors of each other. A **maximal clique** is a clique which cannot be  
<sup>46</sup> made any larger without losing the clique property.



(a)



(b)

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

function for clique  $c$  by  $\psi_c(\mathbf{x}_c; \boldsymbol{\theta}_c)$ , where  $\boldsymbol{\theta}_c$  are its parameters. A potential function can be any non-negative function of its arguments (we give some examples below). We can use these functions to define the joint distribution as we explain in Section 4.3.1.1.

#### 4.3.1.1 Hammersley-Clifford theorem

Suppose a joint distribution  $p$  satisfies the CI properties implied by the undirected graph  $G$ . (We discuss how to derive these properties in Section 4.3.3.) Then the **Hammersley-Clifford theorem** tells us that  $p$  can be written as follows:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{c \in \mathcal{C}} \psi_c(\mathbf{x}_c; \boldsymbol{\theta}_c) \quad (4.66)$$

where  $\mathcal{C}$  is the set of all the (maximal) cliques of the graph  $G$ , and  $Z(\boldsymbol{\theta})$  is the **partition function** given by

$$Z(\boldsymbol{\theta}) \triangleq \sum_{\mathbf{x}} \prod_{c \in \mathcal{C}} \psi_c(\mathbf{x}_c; \boldsymbol{\theta}_c) \quad (4.67)$$

Note that the partition function is what ensures the overall distribution sums to 1.<sup>4</sup>

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.)

<sup>4</sup> The partition function is denoted by  $Z$  because of the German word *Zustandssumme*, which means “sum over states”. This reflects the fact that a lot of pioneering work on MRFs was done by German (and Austrian) physicists, such as Boltzmann.

1    2 **4.3.1.2 Gibbs distribution**

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

4

$$\frac{5}{6} \quad p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp(-\mathcal{E}(\mathbf{x}; \boldsymbol{\theta})) \quad (4.68)$$

7

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

9

10

$$\frac{11}{c} \quad \mathcal{E}(\mathbf{x}; \boldsymbol{\theta}) = \sum_c \mathcal{E}(\mathbf{x}_c; \boldsymbol{\theta}_c) \quad (4.69)$$

12

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

14

15

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

16

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

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

23

24 **4.3.2 Examples**

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

27

28 **4.3.2.1 Ising models**

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

31

$$\frac{32}{33} \quad p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{i \sim j} \psi_{ij}(x_i, x_j; \boldsymbol{\theta}) \quad (4.71)$$

34

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

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

41

$$\frac{42}{43} \quad \psi_{ij}(x_i, x_j; \boldsymbol{\theta}) = \begin{cases} e^{J_{ij}} & \text{if } x_i = x_j \\ e^{-J_{ij}} & \text{if } x_i \neq x_j \end{cases} \quad (4.72)$$

44

45    where  $J_{ij}$  is the coupling strength between nodes  $i$  and  $j$ . This is known as the **Ising model**. If  
46    two nodes are not connected in the graph, we set  $J_{ij} = 0$ . We assume that the weight matrix is  
47

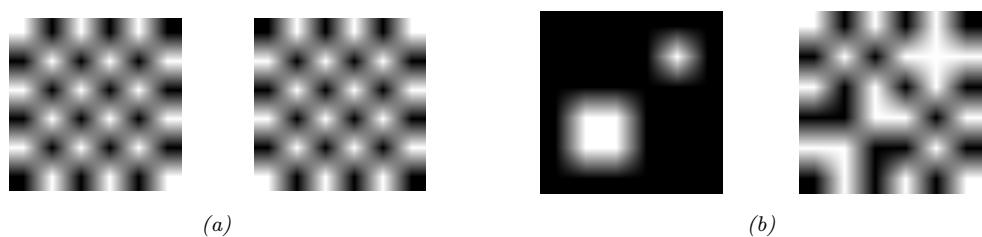


Figure 4.15: (a) The two ground states for a small ferromagnetic Ising model where  $J = 1$ . (b) Two different states for a small Ising model which have the same energy. Left:  $J = 1$ , so neighboring pixels have similar values. Right:  $J = -1$ , so neighboring pixels have different values. From Figures 31.7 and 31.8 of [Mac03].

symmetric, so  $J_{ij} = J_{ji}$ . Often we also assume all edges have the same strength, so  $J_{ij} = J$  for each  $(i, j)$  edge. Thus

$$\psi_{ij}(x_i, x_j; J) = \begin{cases} e^J & \text{if } x_i = x_j \\ e^{-J} & \text{if } x_i \neq x_j \end{cases} \quad (4.73)$$

It is more common to define the Ising model as an energy based model, as follows:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(J)} \exp(-\mathcal{E}(\mathbf{x}; J)) \quad (4.74)$$

$$\mathcal{E}(\mathbf{x}; J) = -J \sum_{i \sim j} x_i x_j \quad (4.75)$$

where  $\mathcal{E}(\mathbf{x}; J)$  is the energy, and where we exploited the fact that  $x_i x_j = -1$  if  $x_i \neq x_j$ , and  $x_i x_j = +1$  if  $x_i = x_j$ . The magnitude of  $J$  controls the degree of coupling strength between neighboring sites, which depends on the (inverse) temperature of the system (colder = more tightly coupled = larger magnitude  $J$ ).

If all the edge weights are positive,  $J > 0$ , then neighboring spins are likely to be in the same state, since if  $x_i = x_j$ , the energy term gets a contribution of  $-J < 0$ , and lower energy corresponds to higher probability. In the machine learning literature, this is called an **associative Markov network**. In the physics literature, this is called a **ferromagnetic** model. If the weights are sufficiently strong, the corresponding probability distribution will have two modes, corresponding to the all +1's state and the all -1's state. These are called the **ground states** of the system. See Figure 4.15a.

If all of the weights are negative,  $J < 0$ , then the spins want to be different from their neighbors (see Figure 4.15b). This is called an **antiferromagnetic** system, and results in a **frustrated system**, since it is not possible for all neighbors to be different from each other in a 2d lattice. Thus the corresponding probability distribution will have multiple modes, corresponding to different “solutions” to the problem.

Figure 4.16 shows some samples from the Ising model for varying  $J > 0$ . (The samples were created using the Gibbs sampling method discussed in Section 12.3.3.) As the temperature reduces, the distribution becomes less entropic, and the “clumpiness” of the samples increases. One can show that, as the lattice size goes to infinity, there is a **critical temperature**  $J_c$  below which many large

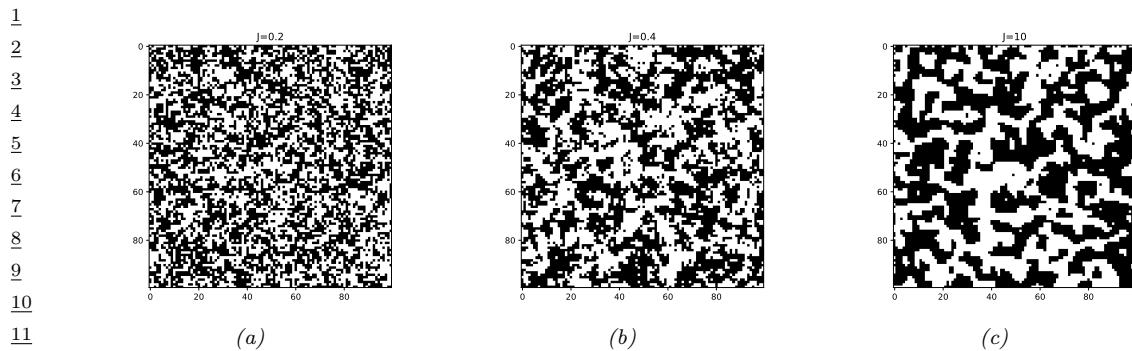


Figure 4.16: Samples from an associative Ising model with varying  $J > 0$ . Generated by `gibbs_demo_ising.py`.

<sup>15</sup> clusters occur, and above which many small clusters occur. In the case of an isotropic square lattice  
<sup>16</sup> model, one can show [Geo88] that

$$\frac{18}{19} \quad J_c = \frac{1}{2} \log(1 + \sqrt{2}) \approx 0.44 \quad (4.76)$$

20 This rapid change in global behavior as we vary a parameter of the system is called a **phase**  
21 **transition**. This can be used to explain how natural systems, such as water, can suddenly go from  
22 solid to liquid, or from liquid to gas, when the temperature changes slightly. See e.g., [Mac03,  
23 ch 31] for further details on the statistical mechanics of Ising models.

In addition to pairwise terms, it is standard to add **unary terms**,  $\psi_i(x_i)$ . In statistical physics, this is called an **external field**. The resulting model is as follows:

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

<sup>30</sup> The  $\psi_i$  terms can be thought of as a local bias term, that are independent of the contributions of the  
<sup>31</sup> neighboring nodes. For binary nodes, we can define this as follows:

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

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

$$\frac{37}{38} \quad \mathcal{E}(x|\theta) = -\alpha \sum_i x_i - J \sum_{i \sim j} x_i x_j \quad (4.79)$$

40 1.3.3.3 Potts models

<sup>42</sup> 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

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

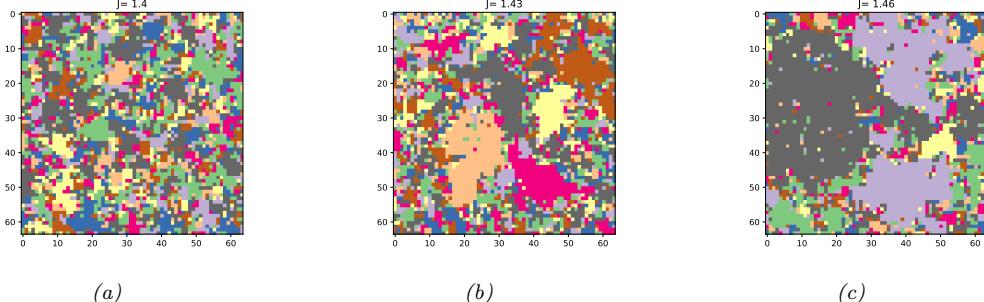


Figure 4.17: Visualizing a sample from a 10-state Potts model of size  $128 \times 128$ . The critical value is  $J_c = \log(1 + \sqrt{10}) = 1.426$ . for different association strengths: (a)  $J = 1.40$ , (b)  $J = 1.43$ , (c)  $J = 1.46$ . Generated by [gibbs\\_demo\\_potts\\_jax.ipynb](#).

where  $J_{ij}(k, k')$  is the energy if one node has state  $k$  and its neighbor has state  $k'$ . A common special case is

$$\psi_{ij}(x_i = k, x_j = k') = \begin{cases} e^J & \text{if } k = k' \\ e^0 & \text{if } k \neq k' \end{cases} \quad (4.81)$$

This is called the **Potts model**. The Potts model reduces to the Ising model if we define  $J_{\text{potts}} = 2J_{\text{Ising}}$ .

If  $J > 0$ , then neighboring nodes are encouraged to have the same label; this is an example of an associative Markov model. Some samples from this model are shown in Figure 4.17. The phase transition for a 2d Potts model occurs at the following value (see [MS96]):

$$J_c = \log(1 + \sqrt{K}) \quad (4.82)$$

We can extend this model to have local evidence for each node. If we write this as an energy based model, we have

$$\mathcal{E}(\mathbf{x}|\boldsymbol{\theta}) = - \sum_i \sum_{k=1}^K \alpha_k \mathbb{I}(x_i = k) - J \sum_{i \sim j} \mathbb{I}(x_i = x_j) \quad (4.83)$$

### 4.3.2.3 Boltzmann machines

MRFs in which all the variables are visible are limited in their expressive power, since the only way to model correlation between the variables is by directly adding an edge. An alternative approach is to introduce latent variables. A **Boltzmann machine** [AHS85] is like an Ising model (Section 4.3.2.1) with latent variables. In addition, the graph structure can be arbitrary (not just a lattice), and the binary states are  $x_i \in \{0, 1\}$  instead of  $x_i \in \{-1, +1\}$ . We usually partition the nodes into hidden nodes  $\mathbf{z}$  and visible nodes  $\mathbf{x}$ , as shown in Figure 4.18(a).

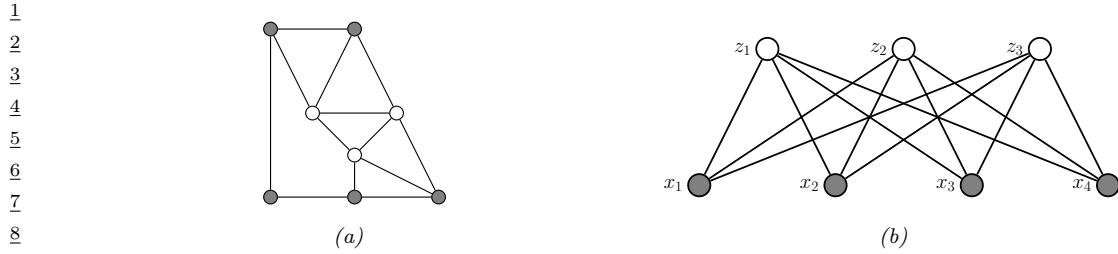


Figure 4.18: (a) A general Boltzmann machine, with an arbitrary graph structure. The shaded (visible) nodes are partitioned into input and output, although the model is actually symmetric and defines a joint distribution on all the nodes. (b) A restricted Boltzmann machine with a bipartite structure.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

#### 4.3.2.4 Restricted Boltzmann machines (RBMs)

Unfortunately, exact inference (and hence learning) in Boltzmann machines is intractable, and even approximate inference (e.g., Gibbs sampling, Section 12.3) can be slow. However, suppose we restrict the architecture so that the nodes are arranged in two layers, and so that there are no connections between nodes within the same layer (see Figure 4.18(b)). This model is known as a **restricted Boltzmann machine (RBM)** [HT01; HS06], or a **harmonium** [Smo86]. The RBM supports efficient approximate inference, since the hidden nodes are conditionally independent given the visible nodes, i.e.,  $p(\mathbf{z}|\mathbf{x}) = \prod_{k=1}^K p(z_k|\mathbf{x})$ . Note this is in contrast to a directed two-layer models, where the explaining away effect causes the latent variables to become “entangled” in the posterior even if they are independent in the prior.

Typically the hidden and visible nodes in an RBM are binary, so the energy terms have the form  $w_{dk}x_dz_k$ . If  $z_k = 1$ , then the  $k$ 'th hidden unit adds a term of the form  $\mathbf{w}_k^\top \mathbf{x}$  to the energy; this can be thought of as a “soft constraint”. If  $z_k = 0$ , the hidden unit is not active, and does not have an opinion about this data example. By turning on different combinations of constraints, we can create complex distributions on the visible data. This is an example of a **product of experts** (Section 25.1.1), since  $p(\mathbf{x}|\mathbf{z}) = \prod_{k:z_k=1} \exp(\mathbf{w}_k^\top \mathbf{x})$ .

This can be thought of as a mixture model with an exponential number of hidden components, corresponding to  $2^K$  settings of  $\mathbf{z}$ . That is,  $\mathbf{z}$  is a **distributed representation**, whereas a standard mixture model uses a **localist representation**, where  $z \in \{1, K\}$ , and each setting of  $z$  corresponds to a complete prototype or exemplar  $\mathbf{w}_k$  to which  $\mathbf{x}$  is compared, giving rise to a model of the form  $p(\mathbf{x}|z=k) \propto \exp(\mathbf{w}_k^\top \mathbf{x})$ .

Many different kinds of RBMs have been defined, which use different pairwise potential functions. See Table 4.4 for a summary. All of these are special cases of the **exponential family harmonium** [WRZH04]. See the supplementary material for more details.

47

Visible	Hidden	Name	Reference
Binary	Binary	Binary RBM	[HS06]
Gaussian	Binary	Gaussian RBM	[WS05]
Categorical	Binary	Categorical RBM	[SMH07]
Multiple categorical	Binary	Replicated softmax/ undirected LDA	[SH10]
Gaussian	Gaussian	Undirected PCA	[MM01]
Binary	Gaussian	Undirected binary PCA	[WS05]

Table 4.4: Summary of different kinds of RBM.

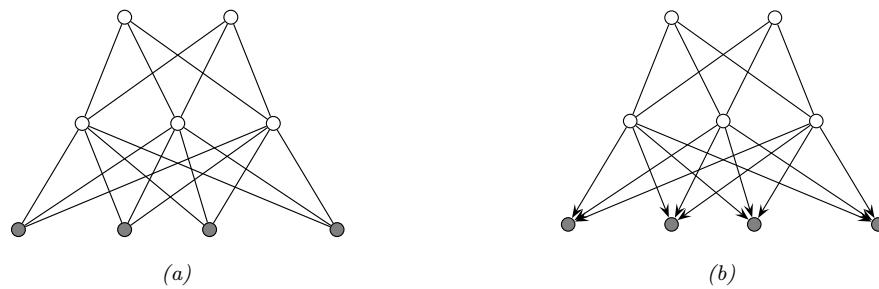


Figure 4.19: (a) Deep Boltzmann machine. (b) Deep belief network. The top two layers define the prior in terms on an RBM. The remaining layers are a directed graphical model that “decodes” the prior into observable data.

### 4.3.2.5 Deep Boltzmann machines

We can make a “deep” version of an RBM by stacking multiple layers; this is called a **deep Boltzmann machine** [SH09]. For example, the two layer model in Figure 4.19(a) has the form

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

where  $\mathbf{x}$  are the visible nodes at the bottom, and we have dropped bias terms for brevity.

### 4.3.2.6 Deep belief networks (DBNs)

We can use an RBM as a prior over a latent distributed code, and then use a PGM-D “decoder” to convert this into the observed data, as shown in Figure 4.19(b). The corresponding joint distribution has the form

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

In other words, it is an RBM on top of a PGM-D. This combination has been called a **deep belief network (DBN)** [HOT06]. However, this name is confusing, since it is not actually a belief net. We will therefore call it a **deep Boltzmann network** (which conveniently has the same DBN abbreviation).

1 DBNs can be trained in a simple greedy fashion, and support fast bottom-up inference (see [HOT06]  
2 for details). DBNs played an important role in the history of deep learning, since they were one of the  
3 first deep models that could be successfully trained. However, they are no longer widely used, since  
4 the advent of better ways to train fully supervised DNNs (such as using ReLU units and the Adam  
5 optimizer), and the advent of efficient ways to train deep PGM-D's, such as the VAE (Section 22.2).  
6

78

#### 9 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

$$\underline{12} \quad \psi_c(\mathbf{x}_c; \boldsymbol{\theta}_c) = \exp(\boldsymbol{\theta}_c^\top \boldsymbol{\phi}(\mathbf{x}_c)) \quad (4.86)$$

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

$$\underline{16} \quad p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left(\sum_c \boldsymbol{\theta}_c^\top \boldsymbol{\phi}(\mathbf{x}_c)\right) = \frac{1}{Z(\boldsymbol{\theta})} \exp(\boldsymbol{\theta}^\top \boldsymbol{\phi}(\mathbf{x})) \quad (4.87)$$

17

18 which we see is equivalent to the exponential family.  
19

20 In Section 2.5.7, we show that the exponential family is the distribution with maximum entropy,  
21 subject to the constraints that the expected value of the features (sufficient statistics)  $\boldsymbol{\phi}(\mathbf{x})$  match  
22 the empirical expectations. Consequently, the model in Equation (4.87) is often called a **maximum**  
23 **entropy** or **maxent** model.

24 For example, in a Gaussian model, we have

25

$$\underline{26} \quad \boldsymbol{\phi}([x_i, x_j]) = [x_i, x_j, x_i x_j] \quad (4.88)$$

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

28

$$\underline{29} \quad \boldsymbol{\phi}([x_i, x_j]) = [x_i, x_j, x_i x_j] \quad (4.89)$$

30

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

32 If the features  $\boldsymbol{\phi}$  are structured in a hierarchical way (capturing first order interactions, and second  
33 order interactions, etc.), and all the variables  $\mathbf{x}$  are categorical, the resulting model is known in  
34 statistics as a **log-linear model**. However, in the ML community, the term “log-linear model” is  
35 often used to describe any model of the form Equation (4.87).

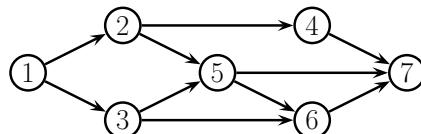
36

#### 37 4.3.2.8 Gaussian MRFs

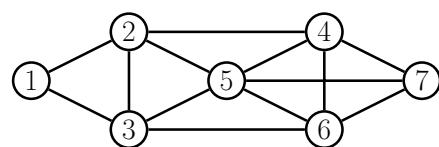
38

39 In Section 4.2.2.3, we showed how to represent a multivariate Gaussian using a PGM-D. In this  
40 section, we show how to represent a multivariate Gaussian using an PGM-U. (For further details, see  
41 e.g., [RH05].)

42 A **Gaussian graphical model** (or **GGM**), also called a **Gaussian MRF**, is a pairwise MRF of  
43



(a)



(b)

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

the following form:

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

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

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

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

The  $\psi_{ij}$  are **edge potentials** (pairwise terms), each the  $\psi_i$  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  $\ell_1$  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  $\{X_1, X_2\} \perp \{X_6, X_7\} | \{X_3, X_4, X_5\}$ .

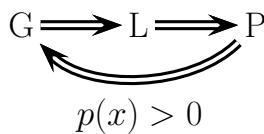


Figure 4.21: Relationship between Markov properties of PGM-U's.

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).<sup>5</sup> 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

<sup>5</sup> This assumes  $p(\mathbf{x}) > 0$  for all  $\mathbf{x}$ , i.e., that  $p$  is a positive density. The restriction to positive densities arises because deterministic constraints can result in independencies present in the distribution that are not explicitly represented in the graph. See e.g., [KF09a, p120] for some examples. Distributions with non-graphical CI properties are said to be **unfaithful** to the graph, so  $I(p) \neq I(G)$ .

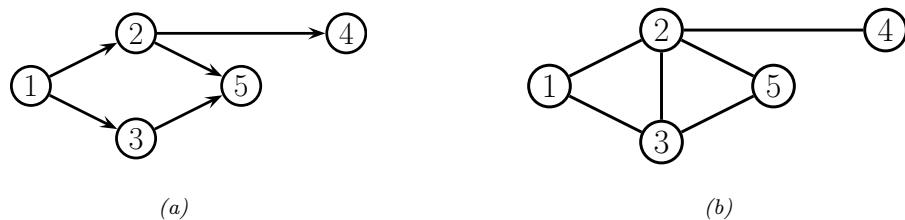


Figure 4.22: (a) The ancestral graph induced by the DAG in Figure 4.20(a) wrt  $U = \{X_2, X_4, X_5\}$ . (b) The moralized version of (a).

PGM-U by dropping the orientation of the edges, but this is clearly incorrect, since a v-structure  $A \rightarrow B \leftarrow C$  has quite different CI properties than the corresponding undirected chain  $A - B - C$  (e.g., the latter graph incorrectly states that  $A \perp C | B$ ). To avoid such incorrect CI statements, we can add edges between the “unmarried” parents  $A$  and  $C$ , and then drop the arrows from the edges, forming (in this case) a fully connected undirected graph. This process is called **moralization**. Figure 4.20 gives a larger example of moralization: we interconnect 2 and 3, since they have a common child 5, and we interconnect 4, 5 and 6, since they have a common child 7.

Unfortunately, moralization loses some CI information, and therefore we cannot use the moralized PGM-U to determine CI properties of the PGM-D. For example, in Figure 4.20(a), using d-separation, we see that  $X_4 \perp X_5 | X_2$ . Adding a moralization arc  $X_4 - X_5$  would lose this fact (see Figure 4.20(b)). However, notice that the 4-5 moralization edge, due to the common child 7, is not needed if we do not observe 7 or any of its descendants. This suggests the following approach to determining if  $A \perp B | C$ . First we form the **ancestral graph** of DAG  $G$  with respect to  $U = A \cup B \cup C$ . This means we remove all nodes from  $G$  that are not in  $U$  or are not ancestors of  $U$ . We then moralize this ancestral graph, and apply the simple graph separation rules for PGM-U’s. For example, in Figure 4.22(a), we show the ancestral graph for Figure 4.20(a) using  $U = \{X_2, X_4, X_5\}$ . In Figure 4.22(b), we show the moralized version of this graph. It is clear that we now correctly conclude that  $X_4 \perp X_5 | X_2$ .

### 4.3.4 Generation (sampling)

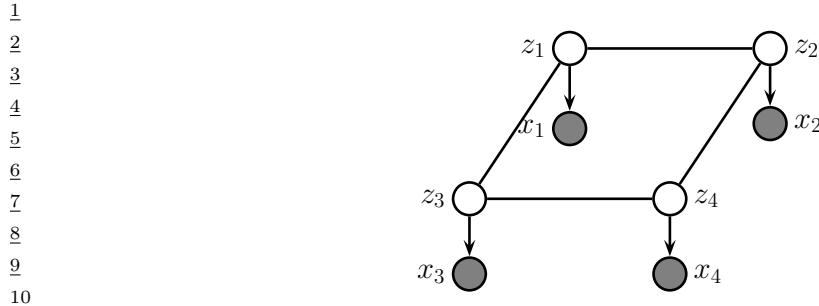
Unlike with PGM-D’s, it can be quite slow to sample from an PGM-U, even from the unconditional prior, because there is no ordering of the variables. Furthermore, we cannot easily compute the probability of any configuration unless we know the value of  $Z$ . Consequently it is common to use MCMC methods for generating from an PGM-U (see Chapter 12).

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

### 4.3.5 Inference

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

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



11 *Figure 4.23: A grid-structured MRF with hidden nodes  $z_i$  and local evidence nodes  $x_i$ . The prior  $p(\mathbf{z})$  is an*  
12 *undirected Ising model, and the likelihood  $p(\mathbf{x}|\mathbf{z}) = \prod_i p(x_i|z_i)$  is a directed fully factored model.*

13



22 *Figure 4.24: Example of image denoising using mean field variational inference. We use an Ising prior with*  
23  $W_{ij} = 1$  *and a Gaussian noise model with  $\sigma = 2$ .* (a) Noisy image. (b) Result of inference. Generated by

24 `ising_image_denoise_demo.py`.

25

26 pixels,  $x_i$ . We assume the joint model has the form

27

28

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

29

30 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  
31 PGM-U as a prior, and has directed edges for the likelihood, as shown in Figure 4.23; such a hybrid  
32 undirected-directed model is called a **chain graph** (even though it is not chain-structured).

33 The inference task is to compute the posterior marginals  $p(z_i|\mathbf{x})$ , or the posterior MAP estimate,  
34  $\text{argmax}_{\mathbf{z}} p(\mathbf{z}|\mathbf{x})$ . The exact computation is intractable for large grids (for reasons explained in  
35 Section 9.4.3), so we must use approximate methods. There are many algorithms that we can use,  
36 including mean field variational inference (Section 10.2.2), Gibbs sampling (Section 12.3.3), loopy  
37 belief propagation (Section 9.3), etc. In Figure 4.24, we show the results of variational inference.

38

### 4.3.6 Learning

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

42

is even harder, because of the additional normalizing constant  $p(\mathcal{D})$  — this case has been called **doubly intractable** [MGM06]. Consequently we will focus on point estimation methods such as MLE and MAP. (For one approach to Bayesian parameter inference in an MRF, based on persistent variational inference, see [IM17].)

#### 4.3.6.1 Learning from complete data

We will start by assuming there are no hidden variables or missing data during training (this is known as the **complete data** setting). For simplicity of presentation, we restrict our discussion to the case of MRFs with log-linear potential functions. (See Section 25.2 for the general nonlinear case, where we discuss MLE for energy based models.)

In particular, we assume the distribution has the following form:

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

where  $c$  indexes the cliques. The log-likelihood becomes

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

Its gradient is given by

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

We know from Section 2.5.3 that the derivative of the log partition function wrt  $\boldsymbol{\theta}_c$  is the expectation of the  $c$ 'th feature vector under the model, i.e.,

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

Hence the gradient of the log likelihood is

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

When the expected value of the features according to the data is equal to the expected value of the features according to the model, the gradient will be zero. This is called **moment matching**, and corresponds to this condition that must hold at the MLE:

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

Evaluating the first term is called the **clamped phase** or **positive phase**, since  $\mathbf{x}$  is set to the observed values  $\mathbf{x}_n$ ; evaluating the second term is called the **unclamped phase** or **negative phase**, since  $\mathbf{x}$  is free to vary, and is generated by the model.

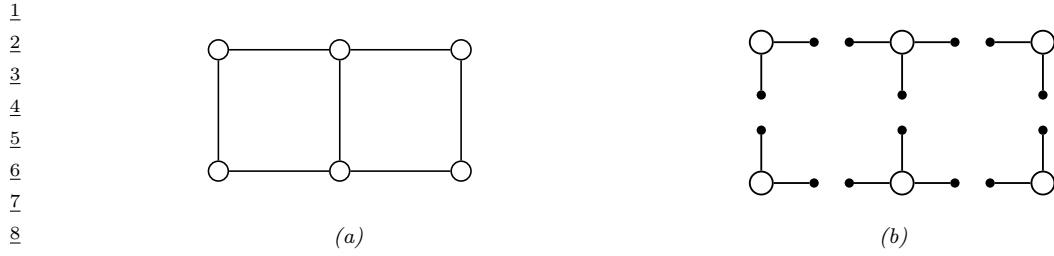


Figure 4.25: (a) A small 2d lattice. (b) The representation used by pseudo likelihood. Solid nodes are observed neighbors. Adapted from Figure 2.2 of [Car03].

In the case of MRFs with tabular potentials (i.e., one feature per entry in the clique table), we can use an algorithm called **iterative proportional fitting** or **IPF** [Fie70; BFH75; JP95] to solve these equations in an iterative fashion.<sup>6</sup> But in general, we must use gradient methods to perform parameter estimation.

#### 4.3.6.2 Computational issues

The biggest computational bottleneck in fitting MRFs and CRFs using MLE is the cost of computing the derivative of the log partition function,  $\log Z(\boldsymbol{\theta})$ , which is needed to compute the derivative of the log likelihood, as we saw in Section 4.3.6.1. To see why this is slow to compute, note that

$$\nabla_{\boldsymbol{\theta}} \log Z(\boldsymbol{\theta}) = \frac{\nabla_{\boldsymbol{\theta}} Z(\boldsymbol{\theta})}{Z(\boldsymbol{\theta})} = \frac{1}{Z(\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} \int \tilde{p}(\mathbf{x}; \boldsymbol{\theta}) d\mathbf{x} = \frac{1}{Z(\boldsymbol{\theta})} \int \nabla_{\boldsymbol{\theta}} \tilde{p}(\mathbf{x}; \boldsymbol{\theta}) d\mathbf{x} \quad (4.104)$$

$$= \frac{1}{Z(\boldsymbol{\theta})} \int \tilde{p}(\mathbf{x}; \boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x}; \boldsymbol{\theta}) d\mathbf{x} = \int \frac{\tilde{p}(\mathbf{x}; \boldsymbol{\theta})}{Z(\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x}; \boldsymbol{\theta}) d\mathbf{x} \quad (4.105)$$

$$= \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x}; \boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x}; \boldsymbol{\theta})] \quad (4.106)$$

where in Equation (4.105) we used the fact that  $\nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{\tilde{p}(\mathbf{x}; \boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} \tilde{p}(\mathbf{x}; \boldsymbol{\theta})$  (this is known as the **log-derivative trick**). Thus we see that we need to draw samples from the model at each step of SGD training, just to estimate the gradient

In Section 25.2.1, we discuss various efficient sampling methods. However, it is also possible to use alternative estimators which do not use the principle of maximum likelihood. For example, in Section 25.2.2 we discuss the technique of contrastive divergence. And in Section 4.3.6.3, we discuss the technique of pseudo likelihood. (See also [Sto17] for a review of many methods for parameter estimation in MRFs.)

1

### 4.3.6.3 Maximum pseudo-likelihood estimation

2

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

5

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

6

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

9

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

10

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

12

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

17

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

21

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

26

### 4.3.6.4 Learning from incomplete data

27

In this section, we consider parameter estimation for MRFs (and CRFs) with hidden variables. Such  
28 **incomplete data** can arise for several reasons. For example, we may want to learn a model of  
29 the form  $p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$  which lets us infer a “clean” image  $\mathbf{z}$  from a noisy or corrupted version  $\mathbf{x}$ . If  
30 we only observe  $\mathbf{x}$ , the model is called a **hidden Gibbs random field**. See Section 10.2.2 for an  
31 example. As another example, we may have a CRF in which the hidden variables are used to encode  
32

33

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

36

7. Geoff Hinton has an analogy for this problem. Suppose we want to learn to denoise images of symmetric shapes,  
37 such as Greek vases. Each hidden pixel  $x_i$  depends on its spatial neighbors, as well as the noisy observation  $y_i$ . Since its  
38 symmetric counterpart  $x_j$  will perfectly predict  $x_i$ , the model will ignore  $y_i$  and just rely on  $x_j$ , even though  $x_j$  will  
39 not be available at test time.

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

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

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

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

<sup>12</sup> where  $\tilde{p}(\mathbf{x}, \mathbf{z} | \boldsymbol{\theta})$  is the unnormalized distribution. We have dropped the sum over cliques  $c$  for brevity.  
<sup>13</sup> The log likelihood is now given by

The log likelihood is now given by

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

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

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

Note that

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

<sup>29</sup> where  $Z(\theta, \mathbf{x}_n)$  is the same as the partition function for the whole model, except that  $\mathbf{x}$  is fixed at  
<sup>30</sup>  $\mathbf{x}_n$ . Thus the log likelihood is a difference of two partition functions, one where  $\mathbf{x}$  is clamped to  $\mathbf{x}_n$   
<sup>31</sup> and  $\mathbf{z}$  is unclamped, and one where both  $\mathbf{x}$  and  $\mathbf{z}$  are unclamped. The gradient of these log partition  
<sup>32</sup> functions corresponds to the expected features, where (in the clamped case) we condition on  $\mathbf{x} = \mathbf{x}_n$ .  
<sup>33</sup> Hence

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

#### 4.4 Comparing directed and undirected PGMs

<sup>40</sup> In this section, we compare PGM-D's and PGM-U's in terms of their modeling power, we discuss  
<sup>41</sup> how to convert from one to the other, and we present a unified representation.

#### 4.4.1 CI properties

<sup>45</sup> Which model has more “expressive power”, a PGM-D or a PGM-U? To formalize the question, recall from Section 4.2.3 that  $G$  is an I-map of a distribution  $p$  if  $I(G) \subseteq I(p)$ , meaning that all

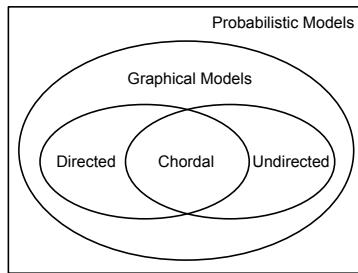


Figure 4.26: PGM-D's and PGM-U's can perfectly represent different sets of distributions. Some distributions can be perfectly represented by either PGM-D's or PGM-U's; the corresponding graph must be chordal.

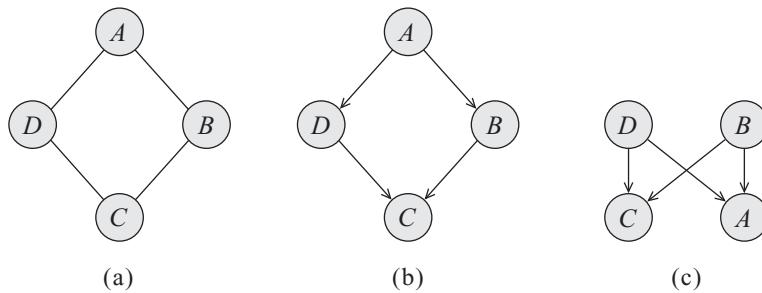


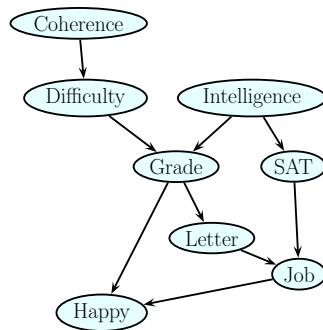
Figure 4.27: A PGM-U and two failed attempts to represent it as a PGM-U. From Figure 3.10 of [KF09a]. Used with kind permission of Daphne Koller.

the CI statements encoded by the graph  $G$  are true of the distribution  $p$ . Now define  $G$  to be **perfect map** of  $p$  if  $I(G) = I(p)$ , in other words, the graph can represent all (and only) the CI properties of the distribution. It turns out that PGM-D's and PGM-U's are perfect maps for different sets of distributions (see Figure 4.26). In this sense, neither is more powerful than the other as a representation language.

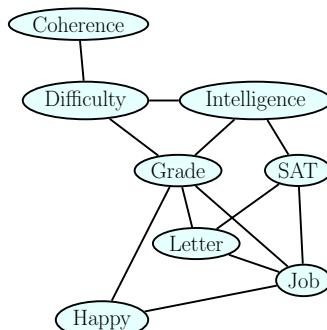
As an example of some CI relationships that can be perfectly modeled by a PGM-D but not a PGM-U, consider a v-structure  $A \rightarrow C \leftarrow B$ . This asserts that  $A \perp B$ , and  $A \not\perp B|C$ . If we drop the arrows, we get  $A - C - B$ , which asserts  $A \perp B|C$  and  $A \not\perp B$ , 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  
14



(a)



(b)

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

18  
19  
20  
21 CI statements encoded by this PGM-U.

22 Some distributions can be perfectly modeled by either a PGM-D or a PGM-U; the resulting graphs  
23 are called **decomposable** or **chordal**. Roughly speaking, this means the following: if we collapse  
24 together all the variables in each maximal clique, to make “mega-variables”, the resulting graph will  
25 be a tree. Of course, if the graph is already a tree (which includes chains as a special case), it will  
26 already be chordal.

27

#### 28 4.4.2 Converting between a directed and undirected model 29

30 Although PGM-D’s and PGM-U’s are not in general equivalent, if we are willing to allow the graph  
31 to encode fewer CI properties than may strictly hold, then we can safely convert one to the other, as  
32 we explain below.

33

34

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

36 We can easily convert a PGM-D to a PGM-U as follows. First, any “unmarried” parents that share a  
37 child must get “married”, by adding an edge between them; this process is known as **moralization**.  
38 Then we can drop the arrows, resulting in an undirected graph. The reason we need to do this is to  
39 ensure that the CI properties of the UGM match those of the DGM, as explained in Section 4.3.3.2.  
40 It also ensures there is a clique that can “store” the CPDs of each family.

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

$$44 \quad P(C, D, I, G, S, L, J, H) \quad (4.116)$$

$$45 \quad = P(C)P(D|C)P(I)P(G|I, D)P(S|I)P(L|G)P(J|L, S)P(H|G, J) \quad (4.117)$$

46

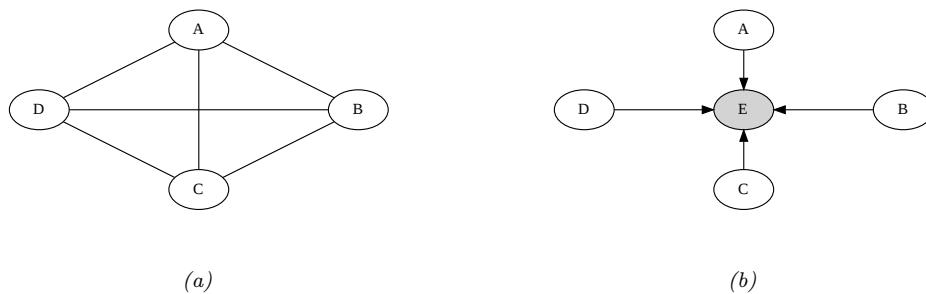


Figure 4.29: (a) An undirected graphical model. (b) A directed equivalent, obtained by adding a dummy observed child node.

Next, we define a potential or factor for every CPD, yielding

$$p(C, D, I, G, S, L, J, H) = \psi_C(C)\psi_D(D, C)\psi_I(I)\psi_G(G, I, D) \quad (4.118)$$

$$\psi_S(S, I)\psi_L(L, G)\psi_J(J, L, S)\psi_H(H, G, J) \quad (4.119)$$

All the potentials are **locally normalized**, since they are CPDs, there is no need for a global normalization constant, so  $Z = 1$ . The corresponding undirected graph is shown in Figure 4.28(b). We see the that we have added D-I, G-J, and L-S moralization edges.<sup>8</sup>

#### 4.4.2.2 Converting a PGM-U to a PGM-D

To convert a PGM-U to a PGM-D, we proceed as follows. For each potential function  $\psi_c(\mathbf{x}_c; \boldsymbol{\theta}_c)$ , we create a “dummy node”, call it  $Y_c$ , which is “clamped” to a special observed state, call it  $y_c^*$ . We then define  $p(Y_c = y_c^* | \mathbf{x}_c) = \psi_c(\mathbf{x}_c; \boldsymbol{\theta}_c)$ . This “local evidence” CPD encodes the same factor as in the DGM. The overall joint has the form  $p_{\text{undir}}(\mathbf{x}) \propto p_{\text{dir}}(\mathbf{x}, \mathbf{y}^*)$ .

As an example, consider the PGM-U in Figure 4.29(a), which defines the joint  $p(A, B, C, D) = \psi(A, B, C, D)/Z$ . We can represent this as a PGM-D by adding a dummy  $E$  node, which is a child of all the other nodes. We set  $E = 1$  and define the CPD  $p(E = 1 | A, B, C, D) \propto \psi(A, B, C, D)$ . By conditioning on this observed child, all the parents become dependent, as in the UGM.

#### 4.4.3 Combining directed and undirected graphs

We can also define graphical models that contain directed and undirected edges. We discuss a few examples below.

<sup>8</sup> We will see this example again in Section 9.4, where we use it to illustrate the variable elimination inference algorithm.

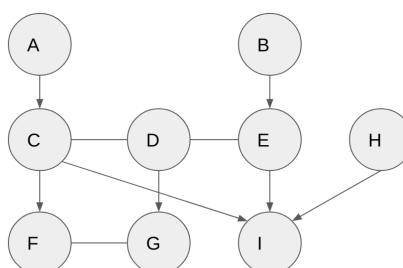


Figure 4.30: A partially directed acyclic graph (PDAG). The chain components are  $\{A\}$ ,  $\{B\}$ ,  $\{C, D, E\}$ ,  $\{F, G\}$ ,  $\{H\}$  and  $\{I\}$ . Adapted from Figure 4.15 of [KF09a].

#### 4.4.3.1 Chain graphs

A **chain graph** is a PGM which may have both directed and undirected edges, but without any directed cycles. A simple example is shown in Figure 10.2, which defines the following joint model:

$$p(\mathbf{x}_{1:D}, \mathbf{y}_{1:D}) = p(\mathbf{x}_{1:D})p(\mathbf{y}_{1:D}|\mathbf{x}_{1:D}) = \left[ \frac{1}{Z} \prod_{(ij)} \psi_{ij}(x_i, x_j) \right] \left[ \prod_{i=1}^D p(y_i|x_i) \right] \quad (4.120)$$

In this example, the prior  $p(\mathbf{x})$  is specified by a PGM-U, and the likelihood  $p(\mathbf{y}|\mathbf{x})$  is specified as a fully factorized PGM-D.

More generally, a chain graph can be defined in terms of a **partially directed acyclic graph (PDAG)**. This is a graph which can be decomposed into a directed graph of **chain components**, where the nodes within each chain component are connected with each other only with undirected edges. See Figure 4.30 for an example.

We can use a PDAG to define a joint distribution using  $\prod_i p(C_i|\text{pa}_{C_i})$ , where each  $C_i$  is a chain component, and each CPD is a conditional random field. For example, referring to Figure 4.30, we have

$$p(A, B, \dots, I) = p(A)p(B)p(C, D, E|A, B)p(F, G|C, D)p(H)p(I|C, E, H) \quad (4.121)$$

$$p(C, D, E|A, B) = \frac{1}{Z(A, B)} \phi(A, C)\phi(B, E)\phi(C, D)\phi(D, E) \quad (4.122)$$

$$p(F, G|C, D) = \frac{1}{Z(C, D)} \phi(F, C)\phi(G, D)\phi(F, G) \quad (4.123)$$

For more details, see e.g., [KF09a, Sec 4.6.2].

#### 4.4.3.2 Acyclic directed mixed graphs

One can show [Pea09b, p51] that every latent variable PGM-D can be rewritten in a way such that every latent variable is a root node with exactly two observed children. This is called the **projection** of the latent variable PGM, and is observationally indistinguishable from the original model.

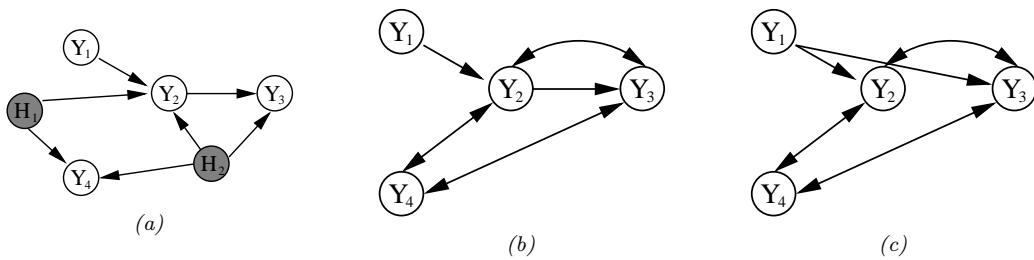


Figure 4.31: (a) A DAG with two hidden variables (shaded). (b) The corresponding ADMG. The bidirected edges reflect correlation due to the hidden variable. (c) A Markov equivalent ADMG. From Figure 3 of [SG09]. Used with kind permission of Ricardo Silva.

Each such latent variable root node induces a dependence between its two children. We can represent this with a directed arc. The resulting graph is called an **acyclic directed mixed graph** or **ADMG**. See Figure 4.31 for an example. (A **mixed graph** is one with undirected, unidirected, and bidirected edges.)

One can determine CI properties of ADMGs using a technique called **m-separation** [Ric03]. This is equivalent to d-separation in a graph where every bidirected edge  $Y_i \leftrightarrow Y_j$  is replaced by  $Y_i \leftarrow X_{ij} \rightarrow Y_j$ , where  $X_{ij}$  is a hidden variable for that edge.

The most common example of ADMGs is when everything is linear-Gaussian. This is known as a structural equation model and is discussed in Section 4.6.2.

#### 4.4.4 Comparing directed and undirected Gaussian PGMs

In this section, we compare directed and undirected Gaussian graphical models. In Section 4.2.2.3, we saw that directed GGMs correspond to sparse regression matrices, and hence sparse Cholesky factorizations of covariance matrices. In Section 4.3.2.8, we saw that undirected GGMs correspond to sparse precision matrices.

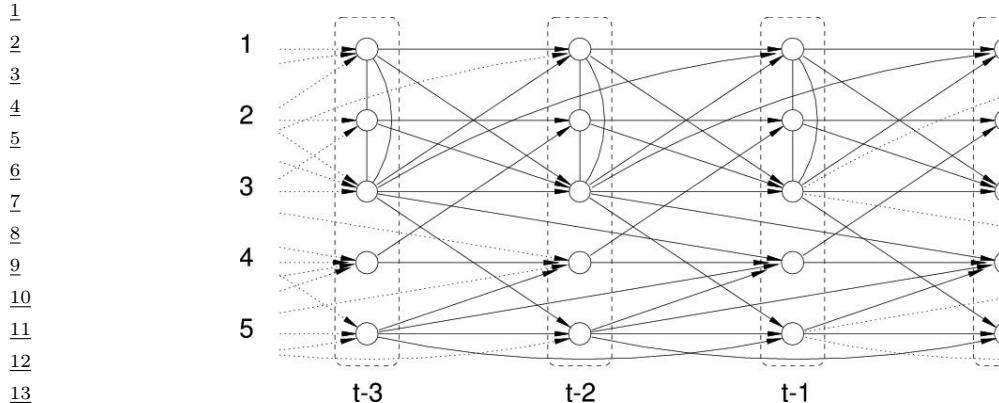
The advantage of the DAG formulation is that we can make the regression weights  $\mathbf{W}$ , and hence  $\Sigma$ , be conditional on covariate information [Pou04], without worrying about positive definite constraints. The disadvantage of the DAG formulation is its dependence on the order, although 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  $\Sigma^{-1}$  is sparse, then the correlation amongst the components within a time slice is most naturally modeled using a



*Figure 4.32: A VAR(2) process represented as a dynamic chain graph. From [DE00]. Used with kind permission of Rainer Dahlhaus.*

PGM-U. For example, suppose we have

$$\mathbf{A}_1 = \begin{pmatrix} \frac{3}{5} & 0 & \frac{1}{5} & 0 & 0 \\ 0 & \frac{3}{5} & 0 & -\frac{1}{5} & 0 \\ \frac{2}{5} & \frac{1}{5} & \frac{3}{5} & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{2} & \frac{1}{5} \\ 0 & 0 & \frac{1}{5} & 0 & \frac{1}{5} \end{pmatrix}, \quad \mathbf{A}_2 = \begin{pmatrix} 0 & 0 & -\frac{1}{5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{5} & 0 & \frac{1}{3} \\ 0 & 0 & 0 & 0 & -\frac{1}{5} \end{pmatrix} \quad (4.125)$$

and

$$\Sigma = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & 0 & 0 \\ \frac{1}{2} & 1 & -\frac{1}{3} & 0 & 0 \\ \frac{1}{3} & -\frac{1}{3} & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad \Sigma^{-1} = \begin{pmatrix} 2.13 & -1.47 & -1.2 & 0 & 0 \\ -1.47 & 2.13 & 1.2 & 0 & 0 \\ -1.2 & 1.2 & 1.8 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.126)$$

The resulting graphical model is illustrated in Figure 4.32. Zeros in the transition matrices  $\mathbf{A}_1$  and  $\mathbf{A}_2$  correspond to absent directed arcs from  $\mathbf{x}_{t-1}$  and  $\mathbf{x}_{t-2}$  into  $\mathbf{x}_t$ . Zeros in the precision matrix  $\Sigma^{-1}$  correspond to absent undirected arcs between nodes in  $\mathbf{x}_t$ .

37

#### 4.4.4.1 Covariance graphs

39

Sometimes we have a sparse covariance matrix rather than a sparse precision matrix. This can be represented using a **bi-directed graph**, where each edge has arrows in both directions, as in Figure 4.33(a). Here nodes that are not connected are unconditionally independent. For example in Figure 4.33(a) we see that  $Y_1 \perp Y_3$ . In the Gaussian case, this means  $\Sigma_{1,3} = \Sigma_{3,1} = 0$ . (A graph representing a sparse covariance matrix is called a **covariance graph**, see e.g., [Pen13]). By contrast, if this were an undirected model, we would have that  $Y_1 \perp Y_3 | Y_2$ , and  $\Lambda_{1,3} = \Lambda_{3,1} = 0$ , where  $\Lambda = \Sigma^{-1}$ .

47



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).

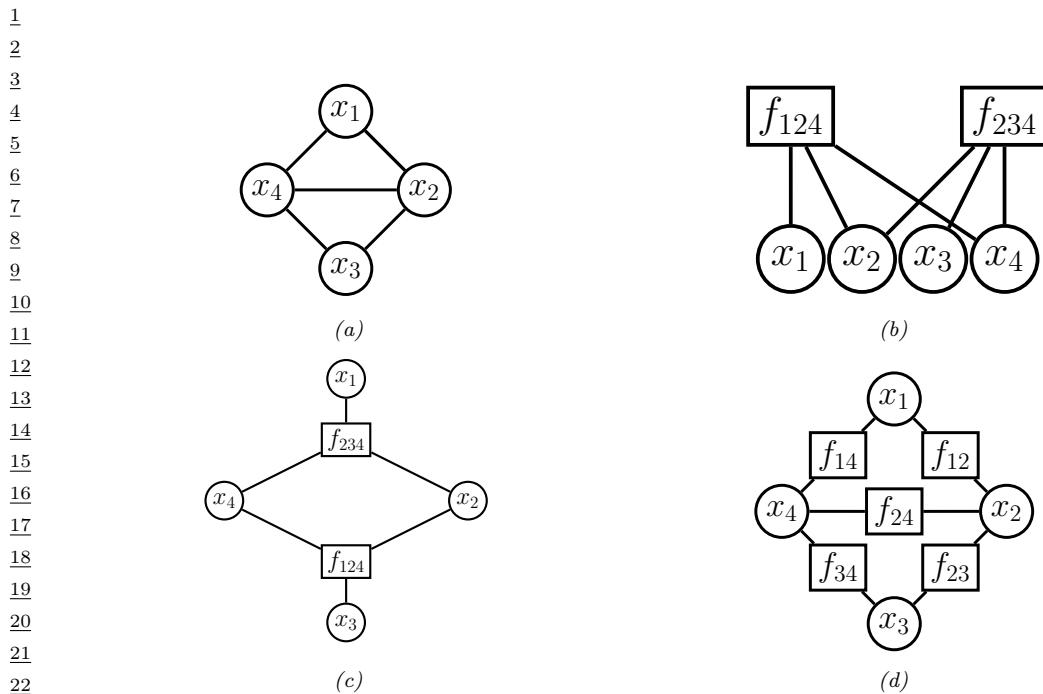
One advantage of factor graphs over PGM-U diagrams is that they are more fine-grained. For example, suppose we associate one potential per edge, rather than per clique. In this case, we get the factor graph in Figure 4.34(d), which represents the function

$$f(x_1, x_2, x_3, x_4) = f_{14}(x_1, x_4)f_{12}(x_1, x_2)f_{34}(x_3, x_4)f_{23}(x_2, x_3)f_{24}(x_2, x_4) \quad (4.128)$$

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.



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

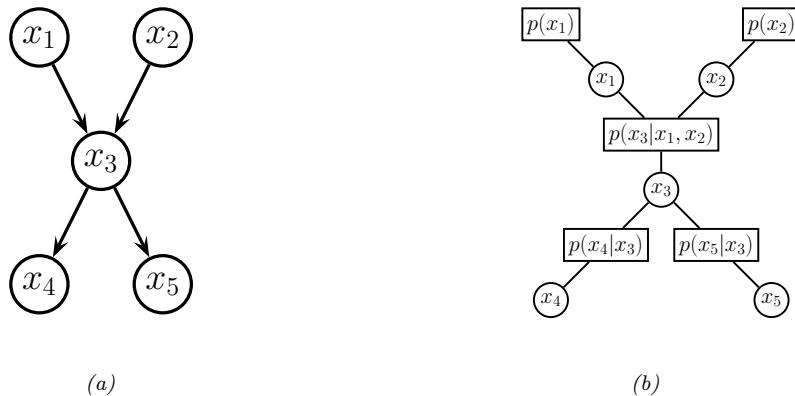
39

40

41

42

43



44 Figure 4.35: (a) A simple PGM-D. (b) Its corresponding factor graph.

45

46

47

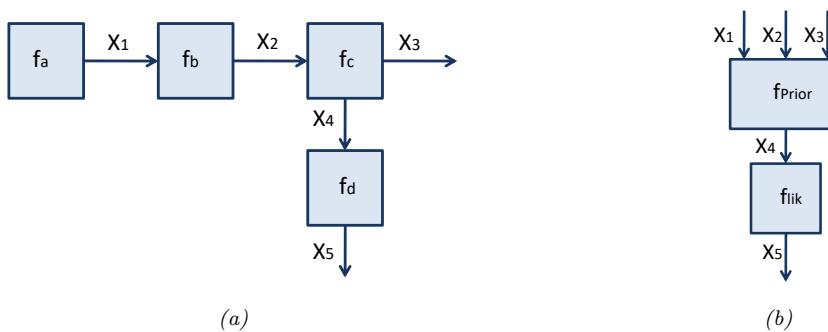


Figure 4.36: A Forney factor graph. (a) Directed version. (b) Hierarchical version.

#### 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

$$f(x_1, \dots, x_5) = f_{prior}(x_1, x_2, x_3, x_4)f_{lik}(x_4, x_5) \quad (4.131)$$

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

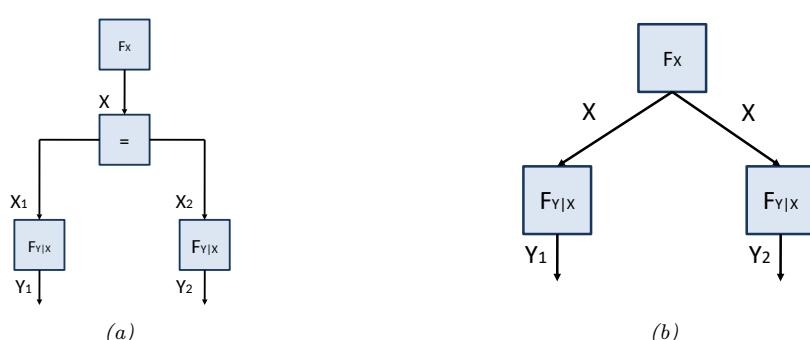


Figure 4.37: An FFG with an equality constraint node.

17 factor acts like a “wire splitter”. Thus the function represented in Figure 4.37(a) is equivalent to the  
18 following:

$$\frac{\partial}{\partial x} f(x, y_1, y_2) = f_x(x) f_{y|x}(y_1, x) f_{y|x}(y_2, x) \quad (4.133)$$

22 This simplified form is represented in Figure 4.37(b), where we reuse the  $x$  variable across multiple  
 23 edges. We have chosen the edge orientations to reflect our interpretation of the factors  $f_{y|x}(y, x)$  as  
 24 likelihood terms,  $p(y|x)$ . We have also chosen to reuse the same  $f_{y|x}$  factor for both  $y$  variables; this  
 25 is an example of **parameter tying**.

## 27 28 4.5 Extensions of Bayes nets

In this section, we discuss some extensions to directed graphical models.

### [32](#) 4.5.1 Probabilistic circuits

<sup>34</sup> 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.

37 Here we briefly describe SPNs. An SPN is a probabilistic model, based on a directed tree-structured  
38 graph, in which terminal nodes represent univariate probability distributions and non-terminal nodes  
39 represent convex combinations (weighted sums) and products of probability functions. SPNs are  
40 similar to deep mixture models, in which we combine together dimensions. SPNs leverage context-  
41 specific independence to reduce the complexity of exact inference to time that is proportional to the  
42 number of links in the graph, as opposed to the treewidth of the graph (see Section 9.4.2 for details  
43 on treewidth).

<sup>44</sup> SPNs are particularly useful for tasks such as missing data imputation of tabular data (see e.g.,  
<sup>45</sup> [Cla20; Ver+19]). A recent extension of SPNs, known as **einsum networks**, is proposed in [Peh+20]  
<sup>46</sup> (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.<sup>9</sup> 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.<sup>10</sup>)

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):

$$H(c) \sim \text{Cat}(0.99, 0.01) \quad (4.138)$$

$$K(c) \sim \text{Cat}(0.1, 0.1, 0.2, 0.3, 0.3) \quad (4.139)$$

$$Q(b) \sim \text{Cat}(0.05, 0.2, 0.4, 0.2, 0.15) \quad (4.140)$$

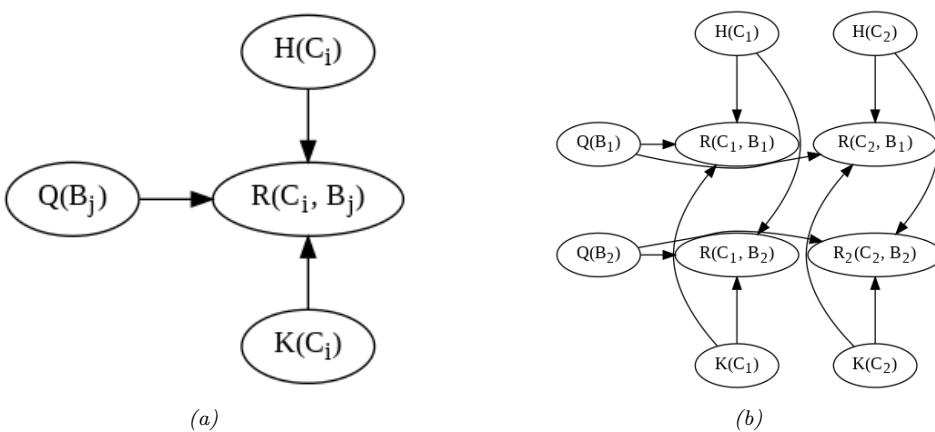
For the recommendation nodes, we need to define a conditional distribution of the form

$$R(c, b) \sim \text{RecCPD}(H(c), K(c), Q(b)) \quad (4.141)$$

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

<sup>9</sup>. See e.g., this blog post from Rob Zinkov: <https://www.zinkov.com/posts/2013-07-28-stop-using-plates>.

<sup>10</sup>. 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.



16 Figure 4.38: RPM for the book review domain. (a) Template for a generic customer  $C_i$  and book  $B_j$  pair.  $R$  17 is rating,  $Q$  is quality,  $H$  is honesty, and  $K$  is kindness. (b) Unrolled model for 2 books and 2 customers.

<sup>19</sup> about what kind of ratings a book receives based on the quality of the book, but also properties of  
<sup>20</sup> the reviewer, such as their honest and kindness. (More sophisticated models of human raters in the  
<sup>21</sup> context of crowd-sourced data collection can be found in e.g., [LRC19].)  
<sup>22</sup>

We can convert the above formulae into a graphical model “**template**”, as shown in Figure 4.38a. Given a set of objects, we can “**unroll**” the template to create a “**ground network**”, as shown in Figure 4.38b. There are  $C \times B + 2C + B$  random variables, with a corresponding joint state space (set of **possible worlds**) of size  $2C5^{C+B+BC}$ , which can get quite large. However, if we are only interested in answering specific queries, we can dynamically unroll small pieces of the network that are relevant to that query [GC90; Bre92].

Let us assume that only a subset of the  $R(c, b)$  entries are observed, and we would like to predict the missing entries of this matrix. This is essentially a simplified **recommender system**. (Unfortunately it ignores key aspects of the problem, such as the content/topic of the books, and the interests/preferences of the customers.) We can use standard probabilistic inference methods for graphical models (which we discuss in Chapter 9) to solve this problem.

Things get more interesting when we don't know which objects are being referred to. For example, customer  $C_1$  might write a review of a book called "Probabilistic Machine Learning", but do they mean edition 1 ( $B_1$ ) or edition 2 ( $B_2$ )? To handle this kind of **relational uncertainty**, we can add all possible referents as parents to each relation. This is illustrated in Figure 4.39, where now  $Q(B_1)$  and  $Q(B_2)$  are both parents of  $R(C_1, B_1)$ . This is necessary because their review score might either depend on  $Q(B_1)$  or  $Q(B_2)$ , depending on which edition they are writing about. To disambiguate this, we create a new variable,  $L(C_i)$ , which specifies which version number of each book customer  $i$  is referring to. The new CPD for the recommendation node,  $p(R(c, b)|H(c), K(c), Q(1 : B), L(c))$ , has the form

$$\frac{43}{43} \quad B(c, b) \sim \text{RecCPT}(H(c), K(c), O(b')) \text{ where } b' = L(c) \quad (4.142)$$

~~45~~ This CPD acts like a **multiplexer**, where the  $L(c)$  node specifies which of the parents  $Q(1 : B)$  to  
~~46~~ actually use.

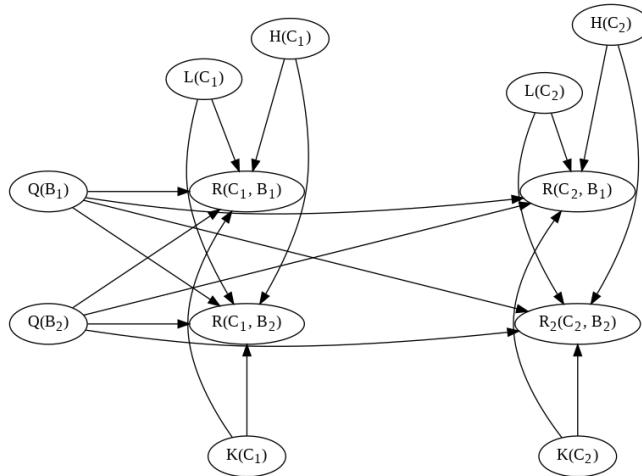


Figure 4.39: An extension of the book review RPM to handle identity uncertainty about which book a given customer is actually reviewing. The  $R(c, b)$  node now depends on all books, since we don't know which one is being referred to. We can select one of these parents based on the mapping specified by the user's library,  $L(c)$ .

Although the above problem may seem contrived, **identity uncertainty** is a widespread problem in many areas, such as citation analysis, credit card histories, and object tracking (see Section 4.5.3). In particular, the problem of **entity resolution** or **record linkage** — which refers to the task of mapping particular strings (such as names) to particular objects (such as people) — is a whole field of research (see e.g., [https://en.wikipedia.org/wiki/Record\\_linkage](https://en.wikipedia.org/wiki/Record_linkage) for an overview and [SHF15] for a Bayesian approach).

### 4.5.3 Open-universe probability models

In Section 4.5.2, we discussed relational probability models, as well as the topic of identity uncertainty. However, we also implicitly made a **closed world assumption**, namely that the set of all objects is fixed and specified ahead of time. In many real world problems, this is an unrealistic assumption.

For example, consider the problem of **multi-target tracking**, where we want to keep track of objects (such as planes or missiles) flying in the sky. Suppose at each time step we get two “blips” on our radar screen, representing the presence of an object at a given location. These measurements are not tagged with the source of the object that generated them, so the data looks like Figure 4.40(a). In Figure 4.40(b-c) we show two different hypotheses about the underlying object trajectories that could have generated this data. However, how can we know there are two objects? Maybe there are more, but some are just not detected. Maybe there are fewer, and some observations are false alarms due to background clutter. One such more complex hypothesis is shown in Figure 4.40(d).

We study specific techniques for **multiple hypothesis tracking** in Section 31.3.4, but the problem is much more general than the above example may suggest. For example, consider the problem of enforcing the UN Comprehensive Nuclear Test Ban Treaty (CTBT). This requires monitoring seismic events, and determining if they were caused by nature or man-made explosions. Thus the number

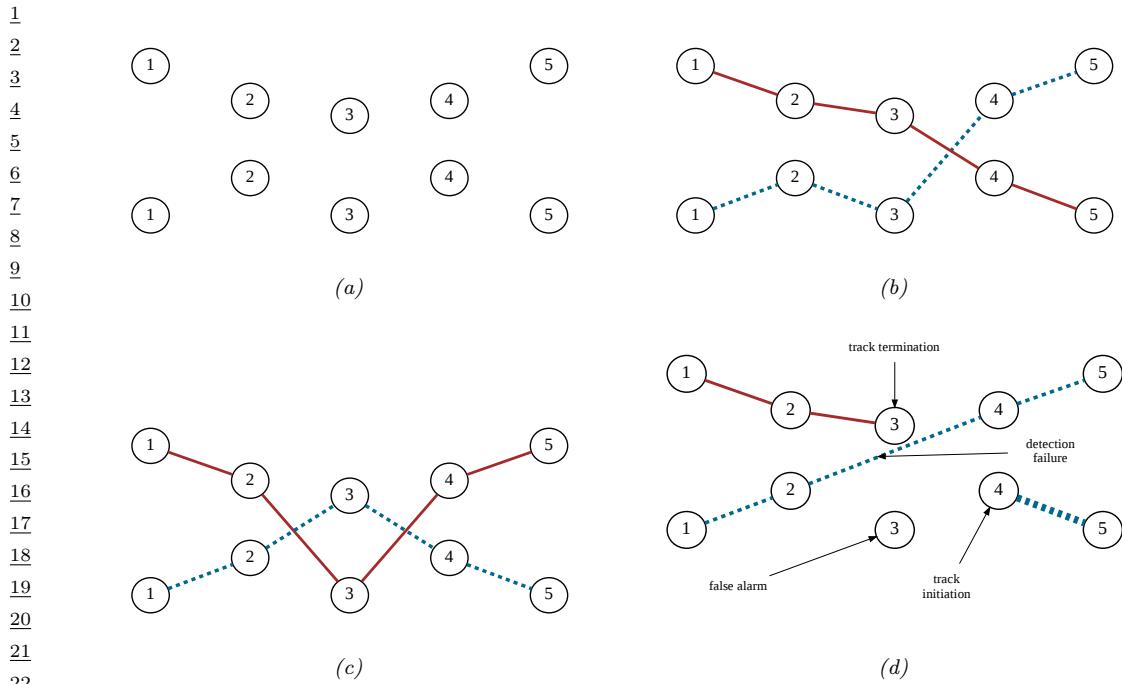


Figure 4.40: Illustration of multi-target tracking in 2d over 5 time steps. (a) We observe 2 measurements per time step. (b-c) Possible hypotheses about the underlying object tracks. (d) A more complex hypothesis in which the red track stops at step 3, the dashed red track starts at step 4 the dotted blue track has a detection failure at step 3, and one of the measurements at step 3 is a false alarm. Adapted from Figure 15.8 of [RN19].

of objects of each type, as well as their source, is uncertain.

As another (more peaceful) example, suppose we want to perform **citation matching**, in which we want to know whether to cite an arxiv version of a paper or the version on some conference website. Are these the same object? It is often hard to tell, since the titles and author might be the same, yet the content may have been updated. It is often necessary to use subtle cues, such as the date stored in the meta-data, to infer if the two “textual measurements” refer to the same underlying object (paper) or not.

In problems such as these, the number of objects of each type, as well as their relationships, is uncertain. This requires the use of **open universe probability models** or **OUPM**, which can generate new objects as well as their properties [Rus15; MR10]. The first formal language for OUPMs was **BLOG** [Mil+05], which stands for “Bayesian LOGic”. This used a general purpose, but slow, MCMC inference scheme to sample over possible worlds of variable size and shape. [Las08; LLC20] describes another open-universe modeling language called **multi-entity Bayesian networks**. More refined versions of this framework were applied to the test ban problem in [ARS13], and to the citation matching problem in [Pas+02].

Very recently, Facebook has released the **Bean Machine** library, available at <https://beanmachine.org/>, which supports more efficient inference in OUPMs. Details can be found in [Teh+20], as well

as their blog post.<sup>11</sup>

#### 4.5.4 Programs as probability models

OUPMs, discussed in Section 4.5.3, let us define probability models over complex dynamic state spaces of unbounded and variable size. The set of possible worlds correspond to objects and their attributes and relationships. Another approach is to use a **probabilistic programming language** or **PPL**, in which we define the set of possible words as the set of **execution traces** generated by the program when it is endowed with a random choice mechanism. (This is a **procedural approach** to the problem, whereas OUPMs are a **declarative approach**.)

The difference between a probabilistic programming language and a standard one was described in [Gor+14] as follows: “Probabilistic programs are usual functional or imperative programs with two added constructs: (1) the ability to draw values at random from distributions, and (2) the ability to condition values of variables in a program via observation”. The former is a way to define  $p(\mathbf{z}, \mathbf{y})$ , and the latter is the same as standard Bayesian conditioning  $p(\mathbf{z}|\mathbf{y})$ .

Some recent examples of PPLs include **Gen** [CT+19], **Pyro** [Bin+19] and **Turing** [GXG18]. Inference in such models is often based on SMC, which we discuss in Chapter 13. For more details on PPLs, see e.g. [Mee+18].

## 4.6 Structural causal models

While probabilities encode our beliefs about a static world, causality tells us whether and how probabilities change when the world changes, be it by intervention or by act of imagination. — Judea Pearl, [PM18b].

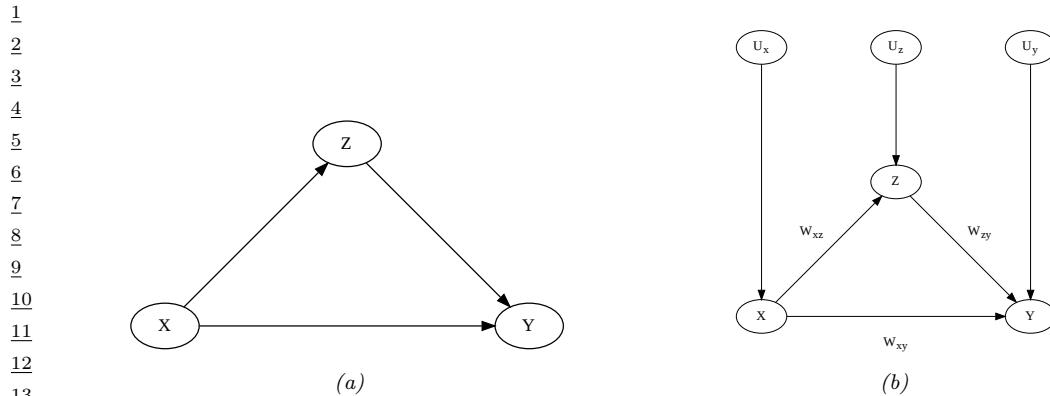
In this section, we discuss how we can use directed graphical model notation to represent **causal models**. We discuss causality in greater detail in Chapter 38, but we introduce some basic ideas and notation here, since it is foundational material that we will need in other parts of the book.

The core idea behind causal models is to create a mechanistic model of the world in which we can reason about the effects of local changes. The canonical example is an electronic circuit: we can predict the effects of any action, such as “knocking out” a particular transistor, or changing the resistance level of a wire, by modifying the circuit locally, and then “re-running” it from the same initial conditions.

We can generalize this idea to create a **structural causal models** or **SCM** [PGJ16], also called **functional causal model** [Sch19]. An SCM is a triple  $\mathcal{M} = (\mathcal{U}, \mathcal{V}, \mathcal{F})$ , where  $\mathcal{U} = \{U_i : i = 1 : N\}$  is a set of unexplained or **exogenous** “noise” variables, which are passed as input to the model,  $\mathcal{V} = \{V_i : i = 1 : N\}$  is a set of **endogeneous** variables that are part of the model itself, and  $\mathcal{F} = \{f_i : i = 1 : N\}$  is a set of deterministic functions of the form  $V_i = f_i(V_{\text{pa}_i}, U_i)$ , where  $\text{pa}_i$  are the parents of variable  $i$ , and  $U_i \in \mathcal{U}$  are the external inputs. We assume the equations can be structured in a **recursive** way, so the dependency graph of nodes given their parents is a DAG. Finally, we assume our model is **causally sufficient**, which means that  $\mathcal{V}$  and  $\mathcal{U}$  are all of the causally relevant factors (although they may not all be observed). This is called the “**causal Markov assumption**”.

Of course, a model typically cannot represent all the variables that might influence observations or decisions. After all, models are *abstractions* of reality. The variables that we choose not to model

<sup>11</sup> See <https://tinyurl.com/2svy5tmh>.



<sup>14</sup> Figure 4.41: (a) PGM for modeling relationship between salary, education and work experience. (b) Corresponding SCM.

<sup>18</sup> explicitly in a functional way can be lumped into the unmodeled exogenous terms. To represent  
<sup>19</sup> our ignorance about these terms, we can use a distribution  $p(U)$  over their values. By “pushing”  
<sup>20</sup> this external noise through the deterministic part of the model, we induce a distribution over the  
<sup>21</sup> endogeneous variables,  $p(\mathcal{V})$ , as in a probabilistic graphical model. However, SCMs make stronger  
<sup>22</sup> assumptions than PGMs.

23 We usually assume  $p(U)$  is factorized (i.e., the  $U_i$  are independent). If they were not, it would  
24 break the assumption that outcomes can be determined locally using deterministic functions. If  
25 there are believed to be dependencies between some of the  $U_i$ , we can add extra hidden parents to  
26 represent this; this is often depicted as a bidirected or undirected edge connecting the  $U_i$ .

### 4.6.1 Example: causal impact of education on wealth

30 We now give a simple example of an SCM, based on [PM18b, p276]. Suppose we are interested in  
31 the causal effect of education on wealth. Let  $X$  represent the level of education of a person (on  
32 some numeric scale, say 0 = high school, 1 = college, 2 = graduate school), and  $Y$  represent their  
33 wealth (at some moment in time). In some cases we might expect that increasing  $X$  would increase  $Y$   
34 (although it of course depends on the nature of the degree, the nature of the job, etc). Thus we add  
35 an edge from  $X$  to  $Y$ . However, getting more education can cost a lot of money (in certain countries),  
36 which is a potentially confounding factor on wealth. Let  $Z$  be the debt incurred by a person based  
37 on their education. We add an edge from  $X$  to  $Z$  to reflect the fact that larger  $X$  means larger  $Z$  (in  
38 general), and we add an edge from  $Z$  to  $Y$  to reflect that larger  $Z$  means lower  $Y$  (in general).

<sup>39</sup> We can represent our structural assumptions graphically as shown in Section 4.6.1(a). The  
<sup>40</sup> corresponding SCM has the form:

$$\frac{41}{41} \quad \sum_{\sigma} f_{\sigma}(U_{\sigma}) \quad (4.142)$$

$$f_{\alpha}(X, U) = f_{\alpha}(X, \{x\}) \quad (4.144)$$

$$\underline{\underline{Y}} = f_{\Sigma}(X, Z, H) \quad (4.145)$$

46 for some set of functions  $f_x, f_y, f_z$ , and some prior distribution  $p(U_x, U_y, U_z)$ . We can also explicitly  
47

represent the exogeneous noise terms as shown in Section 4.6.1(b); this makes clear our assumption that the noise terms are a-priori independent. (We return to this point later.)

### 4.6.2 Structural equation models

A **structural equation model** [Bol89; BP13], also known as a **path diagram**, is a special case of a structural causal model in which all the functional relationships are linear, and the prior on the noise terms is Gaussian. SEMs are widely used in economics and social science, due to the fact that they have a causal interpretation, yet they are computationally tractable.

For example, let us make an SEM version of our education example. We have

$$X = U_x \quad (4.146)$$

$$Z = c_z + w_{xz}X + U_z \quad (4.147)$$

$$Y = c_y + w_{xy}X + w_{zy}Z + U_y \quad (4.148)$$

If we assume  $p(U_x) = \mathcal{N}(U_x|0, \sigma_x^2)$ ,  $p(U_z) = \mathcal{N}(U_z|0, \sigma_z^2)$ , and  $p(U_y) = \mathcal{N}(U_y|0, \sigma_y^2)$ , then the model can be converted to the following Gaussian DGM:

$$p(X) = \mathcal{N}(X|\mu_x, \sigma_x^2) \quad (4.149)$$

$$p(Z|X) = \mathcal{N}(Z|c_z + w_{xz}X, \sigma_z^2) \quad (4.150)$$

$$p(Y|X, Z) = \mathcal{N}(Y|c_y + w_{xy}X + w_{zy}Z, \sigma_y^2) \quad (4.151)$$

We can relax the linearity assumption, to allow arbitrarily flexible functions, and relax the Gaussian assumption, to allow any noise distribution. The resulting “nonparametric SEMs” are equivalent to structural causal models. (For a more detailed comparison between SEMs and SCMs, see [Pea12; BP13; Shi00b].)

### 4.6.3 Do operator and augmented DAGs

One of the main advantages of SCMs is that they let us predict the effect of **interventions**, which are actions that change one or more local mechanisms. A simple intervention is to force a variable to have a given value, e.g., we can force a gene to be “on” or “off”. This is called a **perfect intervention** and is written as  $\text{do}(X_i = x_i)$ , where we have introduced new notation for the “**do**” operator (as in the verb “to do”). This notation means we actively clamp variable  $X_i$  to value  $x_i$  (as opposed to just observing that it has this value). Since the value of  $X_i$  is now independent of its usual parents, we should “cut” the incoming edges to node  $X_i$  in the graph. This is called the “**graph surgery**” operation.

In Figure 4.42a we illustrate this for our education SCM, where we force  $Z$  to have a given value. For example, we may set  $Z = 0$ , by paying off everyone’s student debt. Note that  $p(X|\text{do}(Z = z)) \neq p(X|Z = z)$ , since the intervention changes the model. For example, if we see someone with a debt of 0, we may infer that they probably did not get higher education, i.e.,  $p(X \geq 1|Z = 0)$  is small; but if we pay off everyone’s college loans, then observing someone with no debt in this modified world should not change our beliefs about whether they got higher education, i.e.,  $p(X \geq 1|\text{do}(Z = 0)) = p(X \geq 1)$ .

In more realistic scenarios, we may not be able to set a variable to a specific value, but we may be able to change it from its current value in some way. For example, we may be able to reduce

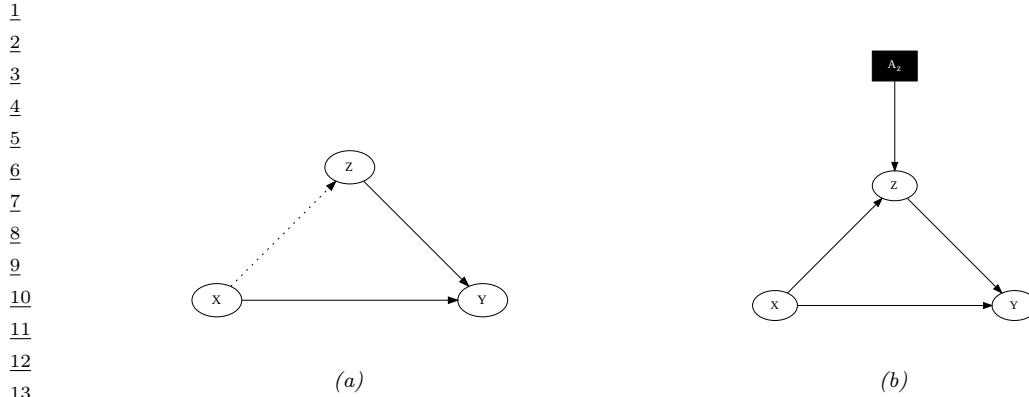


Figure 4.42: An SCM in which we intervene on  $Z$ . (a) Hard intervention, in which we clamp  $Z$  and thus cut its incoming edges (shown as dotted). (b) Soft intervention, in which we change  $Z$ 's mechanism. The square node is an “action” node, using the influence diagram notation from Section 36.2.

everyone’s debt by some fixed amount, say  $\Delta = -10,000$ . Thus we replace  $Z = f_Z(X, U_z)$  with  $Z = f'_Z(Z, U_z)$ , where  $f'_Z(Z, U_z) = f_Z(Z, U_z) + \Delta$ . This is called an **additive intervention**.

To model this kind of scenario, we can add create an **augmented DAG**, in which every variable is augmented with an additional parent node, representing whether or not the variable’s mechanism is changed in some way [Daw02; Daw15; CPD17]. These extra variables are represented by square nodes, and correspond to decision variables or actions, as in the influence diagram formalism (Section 36.2). The same formalism is used in MDPs for reinforcement learning (see Section 36.5).

We give an example of this in Figure 4.42b, where we add the  $A_z \in \{0, 1\}$  node to specify whether we use the debt reduction policy or not. The modified mechanism for  $Z$  becomes

$$Z = f'_Z(X, U_x, A_z) = \begin{cases} f_Z(X, U_x) & \text{if } A_z = 0 \\ f_Z(X, U_x) + \Delta & \text{if } A_z = 1 \end{cases} \quad (4.152)$$

With this new definition, conditioning on the effects of an action can be performed using standard probabilistic inference. That is,  $p(Q|do(A_z = a), E = e) = p(Q|A_z = a, E = e)$ , where  $Q$  is the query (e.g., the event  $X \geq 1$ ) and  $E$  are the (possibly empty) evidence variables. This is because the  $A_z$  node has no parents, so it has no incoming edges to cut when we clamp it.

Although the augmented DAG allows us to use standard notation (no explicit do operators) and inference machinery, the use of “surgical” interventions, which delete incoming edges to a node that is set to a value, results in a simpler graph, which can simplify many calculations, particularly in the non-parametric setting (see [Pea09b, p361] for a discussion). It is therefore a useful abstraction, even if it is less general than the augmented DAG approach.

#### 4.6.4 Estimating average treatment effect using path analysis

We define the **average treatment effect** or **ATE** of some treatment or action  $A$  on some outcome or response variable  $Y$  as follows:

$$\text{ATE}(A) = \mathbb{E}[Y|do(A = 1)] - \mathbb{E}[Y|do(A = 0)] \quad (4.153)$$

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  $\Delta$ , 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  $\Delta$  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:

$$\text{ATE} = \mathbb{E}[Y|A_x = 1] - \mathbb{E}[Y|A_x = 0] \quad (4.159)$$

$$= c_y + w_{xy}(\mu_x + \Delta) + w_{zy}(c_z + w_{xz}(\mu_x + \Delta)) - c_y + w_{xy}(\mu_x) + w_{zy}(c_z + w_{xz}(\mu_x)) \quad (4.160)$$

$$= w_{xy}\Delta + w_{zy}w_{xz}\Delta \quad (4.161)$$

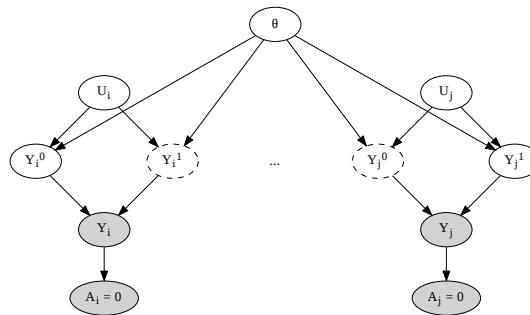
Thus we see that  $w_{xy}$  measures the magnitude of the **direct cause**  $X \rightarrow Y$ , whereas  $w_{xz}w_{zy}$  measures the magnitude of the **indirect cause**,  $X \rightarrow Z \rightarrow Y$ , as **mediated by**  $Z$ .

#### 4.6.5 Counterfactuals

So far we have been focused on predicting the **effects of causes**, 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 [Lattimore2019bayes; Daw00; Daw15; Roh21] for more details).

Now suppose we are interested in the **causes of effects**. For example, suppose I took the aspirin and my headache did go away. I might be interested in the **counterfactual question** “if I had not

1	Level	Activity	Questions	Examples
2	1:Association. $p(Y a)$	Seeing	How would seeing $A$ change my belief in $Y$ ?	Someone took aspirin, how likely is it their headache will be cured?
3	2:Intervention. $p(Y \text{do}(a))$	Doing	What if I do $A$ ?	If I take aspirin, will my headache be cured?
4	3:Counterfactuals. $p(Y^a \text{do}(a'), y')$	Imagining	Was it $A$ that caused $Y$ ?	Would my headache be cured had I not taken aspirin?

10 *Table 4.5: Pearl's causal hierarchy. Adapted from Table 1 of [Pea19].*25 *Figure 4.43: Illustration of the potential outcomes framework as a SCM. The nodes with dashed edges are  
26 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  
27 2, we select action  $A_2 = 1$  and observe  $Y_2 = Y_2^1 = y_2$ .*

30 taken the aspirin, would my headache have gone away anyway?”. This kind of reasoning is crucial for  
31 legal reasoning (see e.g., [DMM17]), as well as for tasks like explainability and fairness.

32 Counterfactual reasoning requires strictly more assumptions than reasoning about interventions,  
33 as we discuss below. Indeed, Judea Pearl has proposed what he calls the **causal hierarchy** [Pea09b;  
34 PGJ16; PM18b], which has three levels of analysis, each more powerful than the last, but each  
35 making stronger assumptions. See Table 4.5 for a summary.

36 In counterfactual reasoning, we want to answer questions of the type  $p(Y^{a'}|\text{do}(a), y)$ , which is read  
37 as: “what is the probability distribution over outcomes  $Y$  if I were to do  $a'$ , given that I have already  
38 done  $a$  and observed outcome  $y$ ”. (We can also condition on any other evidencee that was observed,  
39 such as covariates  $\mathbf{x}$ .) The quantity  $Y^{a'}$  is often called a **potential outcome** [Rub74], since it is  
40 the outcome that would occur in a hypothetical world in which you did  $a'$  instead of  $a$ . (Note that  
41  $p(Y^{a'} = y)$  is equivalent to  $p(Y = y|\text{do}(a'))$ , and is an interventional prediction, not a counterfactual  
42 one.)

43 The assumptions behind the potential outcomes framework can be clearly expressed using a  
44 structural causal model. We illustrate this in Figure 4.43 for a simple case where there are two  
45 possible actions. We see that we have a set of “**units**”, such as individual patients, indexed by  
46 subscripts. Each unit is associated with a hidden exogeneous random noise source,  $U_i$ , that captures  
47

1 everything that is unique about that unit. This noise gets deterministically mapped to two potential  
 2 outcomes,  $Y_i^0$  and  $Y_i^1$ , depending on which action is taken. For any given unit, we only get to observe  
 3 one of the outcomes, namely the one corresponding to the action that was actually chosen. In the  
 4 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  
 5 action  $A_2 = 1$ , so we get to see  $Y_2^1 = y_2$ . The fact that we cannot simultaneously see both outcomes  
 6 for the same unit is called the “**fundamental problem of causal inference**” [Hol86].  
 7

8 We will assume the noise sources are independent, which is known as the “stable unit treatment  
 9 value assumption” or **SUTVA**. (This would not be true if the treatment on person  $j$  could somehow  
 10 affect the outcome of person  $i$ , e.g., due to spreading disease or information between  $i$  and  $j$ .) We  
 11 also assume that the deterministic mechanisms that map noise to outcomes are the same across  
 12 all units (represented by the shared parameter vector  $\theta$  in Figure 4.43). We need to make one final  
 13 assumption, namely that the exogenous noise is not affected by our actions. (This is a formalization  
 14 of the assumption known as “all else being equal”, or (in legal terms) “**ceteris paribus**”.)

15 With the above assumptions, we can predict what the outcome *for an individual unit* would have  
 16 been in the alternative universe where we picked the other action. The procedure is as follows: first  
 17 we perform **abduction** using SCM  $G$ , to infer  $p(U_i|A_i = a, Y_i = y_i)$ , which is the posterior over  
 18 the latent factors for unit  $i$  given the observed evidence in the actual world; second we perform  
 19 **intervention**, in which we modify the causal mechanisms of  $G$  by replacing  $A_i = a$  with  $A_i = a'$  to  
 20 get  $G_{a'}$ ; third we perform **prediction**, in which we propagate the distribution of the latent factors,  
 21  $p(U_i|A_i = a, Y_i = y_i)$ , through the modified SCM  $G_{a'}$  to get  $p(Y_i^{a'}|A_i = a, Y_i = y_i)$ .

22 In Figure 4.43, we see that we have two copies of every possible outcome variable, to represent  
 23 the set of possible worlds. Of course, we only get to see one such world, based on the actions that  
 24 we actually took. More generally, a model in which we “clone” all the deterministic variables, with  
 25 the noise being held constant between the two branches of the graph for the same unit, is called a  
 26 **twin network** [Pea09b]. We will see a more practical example in Section 19.3.5, where we discuss  
 27 assessing the counterfactual causal impact of an intervention in a time series. (See also [RR11; RR13],  
 28 who propose a related formalism known as **single world intervention graph** or **SWIG**.)

29 We see from the above that the potential outcomes framework is mathematically equivalent to  
 30 structural causal models, but does not use graphical model notation. This has led to heated debate  
 31 between the founders of the two schools of thought.<sup>12</sup>. The SCM approach is more popular in  
 32 computer science (see e.g., [PJS17; Sch19; Sch+21b]), and the PO approach is more popular in  
 33 economics (see e.g. [AP09; Imb19]). Modern textbooks on causality usually use both formalisms (see  
 34 e.g., [HR20a; Nea20]).

35  
 36  
 37  
 38  
 39  
 40  
 41  
 42  
 43  
 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](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.



# 5 Information theory

Machine learning is fundamentally about **information processing**. But what do we mean by “information”? We discuss this in Section 5.1–Section 5.3. We then go on to briefly discuss two main applications of information theory. The first application is **data compression** or **source coding**, which is the problem of removing redundancy from data so it can be represented more compactly, either in a lossless way (e.g., ZIP files) or a lossy way (e.g., MP3 files). See Section 5.4 for details. The second application is **error correction** or **channel coding**, which means encoding data in such a way that it is robust to errors when sent over a noisy channel, such as a telephone line or a satellite link. See Section 5.5 for details.

It turns out that methods for data compression and error correction both rely on having an accurate probabilistic model of the data. For compression, a probabilistic model is needed so the sender can assign shorter **codewords** to data vectors which occur most often, and hence save space. For error correction, a probabilistic model is needed so the receiver can infer the most likely source message by combining the received noisy message with a prior over possible messages.

It is clear that probabilistic machine learning is useful for information theory. However, information theory is also useful for machine learning. Indeed, we have seen that Bayesian machine learning is about representing and reducing our uncertainty, and so is fundamentally about information. In Section 5.6.2, we explore this direction in more detail, where we discuss the information bottleneck.

For more information on information theory, see e.g., [Mac03; CT06].

## 5.1 KL divergence

*This section is written by Alex Alemi.*

To discuss information theory, we need some way to measure or quantify information itself. Let’s say we start with some distribution describing our degrees of belief about a random variable, call it  $q(x)$ . We then want to update our degrees of belief to some new distribution  $p(x)$ , perhaps because we’ve taken some new measurements or merely thought about the problem a bit longer. What we seek is a mathematical way to quantify the magnitude of this update, which we’ll denote  $I[p\|q]$ . What sort of criteria would be reasonable for such a measure? We discuss this issue below, and then define a quantity that satisfies these criteria.

1 2 **5.1.1 Desiderata**

3 For simplicity, imagine we are describing a distribution over  $N$  possible events. In this case, the  
4 probability distribution  $q(\mathbf{x})$  consists of  $N$  non-negative real numbers that add up to 1. To be even  
5 more concrete, imagine we are describing the random variable representing the suit of the next card  
6 we'll draw from a deck:  $S \in \{\clubsuit, \spadesuit, \heartsuit, \diamondsuit\}$ . Imagine we initially believe the distributions over suits to  
7 be uniform:  $q = [\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}]$ . If our friend told us they removed all of the red cards we could update  
8 to:  $q' = [\frac{1}{2}, \frac{1}{2}, 0, 0]$ . Alternatively, we might believe some diamonds changed into clubs and want to  
9 update to  $q'' = [\frac{3}{8}, \frac{2}{8}, \frac{2}{8}, \frac{1}{8}]$ . Is there a good way to quantify *how much* we've updated our beliefs?  
10 Which is a larger update:  $q \rightarrow q'$  or  $q \rightarrow q''$ ?

11 It seems desireable that any useful such measure would satisfy the following properties:  
12

- 13 1. *continuous* in its arguments: If we slightly perturb either our starting or ending distribution,  
14 it should similarly have a small effect on the magnitude of the update. For example:  $I[p \parallel \frac{1}{4} +$   
15  $\epsilon, \frac{1}{4}, \frac{1}{4}, \frac{1}{4} - \epsilon]$  should be close to  $I[p \parallel q]$  for small  $\epsilon$ , where  $q = [\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}]$ .
- 16 2. *non-negative*:  $I[p \parallel q] \geq 0$  for all  $p(\mathbf{x})$  and  $q(\mathbf{x})$ . The magnitude of our updates are non-negative.  
17
- 18 3. *permutation invariant*: The magnitude of the update should not depend on the order we choose for  
19 the elements of  $\mathbf{x}$ . For example, it shouldn't matter if I list my probabilities for the suits of cards  
20 in the order  $\clubsuit, \spadesuit, \heartsuit, \diamondsuit$  or  $\clubsuit, \diamondsuit, \heartsuit, \spadesuit$ , if I keep the order consistent across all of the distributions,  
21 I should get the same answer. For example:  $I[a, b, c, d \parallel e, f, g, h] = I[a, d, c, b \parallel e, h, g, f]$ .  
22
- 23 4. *monotonic* for uniform distributions: While its hard to say how large the updates in our beliefs  
24 are in general, there are some special cases for which we have a strong intuition. If our beliefs  
25 update from a uniform distribution on  $N$  elements to one that is uniform in  $N'$  elements, the  
26 information gain should be an increasing function of  $N$  and a decreasing function of  $N'$ . For  
27 instance changing from a uniform distribution on all four suits  $[\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}]$  (so  $N = 4$ ) to only one  
28 suit, such as all clubs,  $[1, 0, 0, 0]$  where  $N' = 1$ , is a larger update than if I only updated to the  
29 card being black,  $[\frac{1}{2}, \frac{1}{2}, 0, 0]$  where  $N' = 2$ .
- 30 5. satisfy a natural *chain rule*: So far we've been describing our beliefs in what will happen on the next  
31 card draw as a single random variable representing the suit of the next card ( $S \in \{\clubsuit, \spadesuit, \heartsuit, \diamondsuit\}$ ).  
32 We could equivalently describe the same physical process in two steps. First we consider the  
33 random variable representing the color of the card ( $C \in \{\blacksquare, \square\}$ ), which could be either black  
34 ( $\blacksquare = \{\clubsuit, \spadesuit\}$ ) or red ( $\square = \{\heartsuit, \diamondsuit\}$ ). Then, if we draw a red card we describe our belief that it is  $\heartsuit$   
35 versus  $\diamondsuit$ . If it was instead black we would assign beliefs to it being  $\clubsuit$  versus  $\spadesuit$ . We can convert  
36 any distribution over the four suits into this conditional factorization, for example:  
37

$$\text{p}(S) = \left[ \frac{3}{8}, \frac{2}{8}, \frac{2}{8}, \frac{1}{8} \right] \quad (5.1)$$

40 becomes

$$\text{p}(C) = \left[ \frac{5}{8}, \frac{3}{8} \right] \quad \text{p}(\{\clubsuit, \spadesuit\} | C = \blacksquare) = \left[ \frac{3}{5}, \frac{2}{5} \right] \quad \text{p}(\{\heartsuit, \diamondsuit\} | C = \square) = \left[ \frac{2}{3}, \frac{1}{3} \right]. \quad (5.2)$$

45 In the same way we could decompose our uniform distribution  $q$ . Obviously, for our measure of  
46 information to be of use the magnitude of the update needs to be the same regardless of how we  
47

choose to describe what is ultimately the same physical process. What we need is somehow to relate what would be four different invocations of our information function:

$$I_S \equiv I[p(S)\|q(S)] \quad (5.3)$$

$$I_C \equiv I[p(C)\|q(C)] \quad (5.4)$$

$$I_{\blacksquare} \equiv I[p(\{\clubsuit, \spadesuit\}|C = \blacksquare)\|q(\{\clubsuit, \spadesuit\}|C = \blacksquare)] \quad (5.5)$$

$$I_{\square} \equiv I[p(\{\heartsuit, \diamondsuit\}|C = \square)\|q(\{\heartsuit, \diamondsuit\}|C = \square)]. \quad (5.6)$$

Clearly  $I_S$  should be some function of  $\{I_C, I_{\blacksquare}, I_{\square}\}$ . Our last desiderata is that the way we measure the magnitude of our updates will have  $I_S$  be a linear combination of  $I_C, I_{\blacksquare}, I_{\square}$ . In particular, we will require that they combine as a weighted linear combinations, with weights set by the probability that we would find ourselves in that branch according to the distribution  $p$ :

$$I_S = I_C + p(C = \blacksquare)I_{\blacksquare} + p(C = \square)I_{\square} = I_C + \frac{5}{8}I_{\blacksquare} + \frac{3}{5}I_{\square} \quad (5.7)$$

Stating this requirement more generally: If we partition  $\mathbf{x}$  into two pieces  $[\mathbf{x}_L, \mathbf{x}_R]$ , so that we can write  $p(\mathbf{x}) = p(\mathbf{x}_L)p(\mathbf{x}_R|\mathbf{x}_L)$  and similarly for  $q$ , the magnitude of the update should be

$$I[p(\mathbf{x})\|q(\mathbf{x})] = I[p(\mathbf{x}_L)\|q(\mathbf{x}_L)] + \mathbb{E}_{p(\mathbf{x}_L)} [I[p(\mathbf{x}_R|\mathbf{x}_L)\|q(\mathbf{x}_R|\mathbf{x}_L)]] . \quad (5.8)$$

Notice that this requirement *breaks the symmetry between our two distributions*: The right hand side asks us to take the expected conditional information gain with respect to the marginal, but we need to decide which of two marginals to take the expectation with respect to.

### 5.1.2 The KL divergence uniquely satisfies the desiderata

We will now define a quantity that is the only measure (up to a multiplicative constant) that satisfies the above desiderata. The **Kullback-Leibler divergence** or **KL divergence**, also known as the **information gain** or **relative entropy**, is defined as follows:

$$D_{\text{KL}}(p\|q) \triangleq \sum_{k=1}^K p_k \log \frac{p_k}{q_k}. \quad (5.9)$$

This naturally extends to continuous distributions:

$$D_{\text{KL}}(p\|q) \triangleq \int dx p(x) \log \frac{p(x)}{q(x)}. \quad (5.10)$$

Next we will verify that this definition satisfies all of our desiderata. (The proof that it is the unique measure which captures these properties can be found in e.g., [Hob69; Rén61].)

#### 5.1.2.1 Continuity of KL

One of our desiderata was that our measure of information gain should be continuous. The KL divergence is manifestly continuous in its arguments except potentially when  $p_k$  or  $q_k$  is zero. In the first case, notice that the limit as  $p \rightarrow 0$  is well behaved:

$$\lim_{p \rightarrow 0} p \log \frac{p}{q} = 0. \quad (5.11)$$

1 Taking this as the definition of the value of the integrand when  $p = 0$  will make it continuous there.  
2 Notice that we do have a problem however if  $q = 0$  in some place that  $p \neq 0$ . Our information  
3 gain requires that our original distribution of beliefs  $q$  has some support everywhere the updated  
4 distribution does. Intuitively it would require an infinite amount of information for us to update our  
5 beliefs in some outcome to change from being exactly 0 to some positive value.  
6

### 7 5.1.2.2 Non-negativity of KL divergence

8 In this section, we prove that the KL divergence as defined is always non-negative. We will make use  
9 of **Jensen's inequality**, which states that for any convex function  $f$ , we have that  
10

$$\underline{12} \quad f\left(\sum_{i=1}^n \lambda_i \mathbf{x}_i\right) \leq \sum_{i=1}^n \lambda_i f(\mathbf{x}_i) \quad (5.12)$$

13 where  $\lambda_i \geq 0$  and  $\sum_{i=1}^n \lambda_i = 1$ . This can be proved by induction, where the base case with  $n = 2$   
14 follows by definition of convexity.

15  
16 **Theorem 5.1.1.** (*Information inequality*)  $D_{\text{KL}}(p\|q) \geq 0$  with equality iff  $p = q$ .

17 *Proof.* We now prove the theorem, following [CT06, p28]. As we noted in the previous section, the  
18 KL divergence requires special consideration when  $p(x)$  or  $q(x) = 0$ , the same is true here. Let  
19  $A = \{x : p(x) > 0\}$  be the support of  $p(x)$ . Using the convexity of the log function and Jensen's  
20 inequality, we have that  
21

$$\underline{22} \quad -D_{\text{KL}}(p\|q) = -\sum_{x \in A} p(x) \log \frac{p(x)}{q(x)} = \sum_{x \in A} p(x) \log \frac{q(x)}{p(x)} \quad (5.13)$$

$$\underline{23} \quad \leq \log \sum_{x \in A} p(x) \frac{q(x)}{p(x)} = \log \sum_{x \in A} q(x) \quad (5.14)$$

$$\underline{24} \quad \leq \log \sum_{x \in \mathcal{X}} q(x) = \log 1 = 0 \quad (5.15)$$

25 Since  $\log(x)$  is a strictly concave function ( $-\log(x)$  is convex), we have equality in Equation (5.14) iff  
26  $p(x) = cq(x)$  for some  $c$  that tracks the fraction of the whole space  $\mathcal{X}$  contained in  $A$ . We have equality  
27 in Equation (5.15) iff  $\sum_{x \in A} q(x) = \sum_{x \in \mathcal{X}} q(x) = 1$ , which implies  $c = 1$ . Hence  $D_{\text{KL}}(p\|q) = 0$  iff  
28  $p(x) = q(x)$  for all  $x$ .  $\square$   
29

30 The non-negativity of KL divergence often feels as though its one of the most useful results in  
31 Information Theory. It is a good result to keep in your back pocket. Anytime you can rearrange an  
32 expression in terms of KL divergence terms, since those are guaranteed to be non-negative, dropping  
33 them immediately generates a bound.  
34

### 35 5.1.2.3 KL divergence is invariant to reparameterizations

36 We wanted our measure of information to be invariant to permutations of the labels. The discrete  
37 form is manifestly permutation invariant as summations are. The KL divergence actually satisfies a  
38

1 much stronger property of reparameterization invariance. Namely, we can transform our random  
2 variable through an arbitrary invertible map and it won't change the value of the KL divergence.  
3

4 If we transform our random variable from  $x$  to some  $y = f(x)$  we know that  $p(x) dx = p(y) dy$  and  
5  $q(x) dx = q(y) dy$ . Hence the KL divergence remains the same for both random variables:

$$\underline{6} \quad D_{\text{KL}}(p(x)\|q(x)) = \int dx p(x) \log \frac{p(x)}{q(x)} = \int dy p(y) \log \left( \frac{p(y)}{q(y)} \left| \frac{dy}{dx} \right| \right) = D_{\text{KL}}(p(y)\|q(y)). \quad (5.16)$$

10 Because of this reparameterization invariance we can rest assured that when we measure the KL  
11 divergence between two distributions we are measuring something about the distributions and not the  
12 way we choose to represent the space in which they are defined. We are therefore free to transform  
13 our data into a convenient basis of our choosing, such as a Fourier bases for images, without affecting  
14 the result.

#### 16 5.1.2.4 Monotonicity for uniform distributions

17 Consider updating a probability distribution from a uniform distribution on  $N$  elements to a uniform  
18 distribution on  $N'$  elements. The KL divergence is:

$$\underline{20} \quad D_{\text{KL}}(p\|q) = \sum_k \frac{1}{N'} \log \frac{\frac{1}{N'}}{\frac{1}{N}} = \log \frac{N}{N'}, \quad (5.17)$$

23 or the log of the ratio of the elements before and after the update. This satisfies our monotonicity  
24 requirement.

25 We can interpret this result as follows: Consider finding an element of a sorted array by means of  
26 bisection. A well designed yes/no question can cut the search space in half. Measured in bits, the  
27 KL divergence tells us how many well designed yes/no questions are required on average to move  
28 from  $q$  to  $p$ .

#### 30 5.1.2.5 Chain rule for KL divergence

31 Here we show that the KL divergence satisfies a natural chain rule:

$$\underline{33} \quad D_{\text{KL}}(p(x,y)\|q(x,y)) = \int dx dy p(x,y) \log \frac{p(x,y)}{q(x,y)} \quad (5.18)$$

$$\underline{35} \quad = \int dx dy p(x,y) \left[ \log \frac{p(x)}{q(x)} + \log \frac{p(y|x)}{q(y|x)} \right] \quad (5.19)$$

$$\underline{37} \quad = D_{\text{KL}}(p(x)\|q(x)) + \mathbb{E}_{p(x)} [D_{\text{KL}}(p(y|x)\|q(y|x))]. \quad (5.20)$$

39 We can rest assured that we can decompose our distributions into their conditionals and the KL  
40 divergences will just add.

41 As a notational convenience, the **conditional KL divergence** is defined to be the expected value  
42 of the KL divergence between two conditional distributions:

$$\underline{44} \quad D_{\text{KL}}(p(y|x)\|q(y|x)) \triangleq \int dx p(x) \int dy p(y|x) \log \frac{p(y|x)}{q(y|x)}. \quad (5.21)$$

46 This allows us to drop many expectation symbols.

1    2 **5.1.3 Thinking about KL**

3 In this section, we discuss some qualitative properties of the KL divergence.

5    6 **5.1.3.1 Units of KL**

7 Above we said that the desiderata we listed determined the KL divergence up to a multiplicative  
8 constant. Because the KL divergence is logarithmic, and logarithms in different bases are the same  
9 up to a multiplicative constant, our choice of the base of the logarithm when we compute the KL  
10 divergence is a choice akin to choosing which units to measure the information in.

11 If the KL divergence is measured with the base-2 logarithm, it is said to have units of **bits**, short  
12 for “binary digits”. If measured using the natural logarithm as we normally do for mathematical  
13 convenience, it is said to be measured in **nats** for “natural units”.

14 To convert between the systems, we use  $\log_2 y = \frac{\log y}{\log 2}$ . Henec

$$\text{16} \quad 1 \text{ bit} = \log 2 \text{ nats} \sim 0.693 \text{ nats} \tag{5.22}$$

$$\text{17} \quad 1 \text{ nat} = \frac{1}{\log 2} \text{ bits} \sim 1.44 \text{ bits.} \tag{5.23}$$

20 **5.1.3.2 Asymmetry of the KL divergence**

22 The KL divergence is *not* symmetric in its two arguments. While many find this asymmetry confusing  
23 at first, we can see that the asymmetry stems from our requirement that we have a natural chain  
24 rule. When we decompose the distribution into its conditional, we need to take an expectation with  
25 respect to the variables being conditioned on. In the KL divergence we take this expectation with  
26 respect to the first argument  $p(x)$ . This breaks the symmetry between the two distributions.

27 At a more intuitive level, we can see that the information required to move from  $q$  to  $p$  is in general  
28 different than the information required to move from  $p$  to  $q$ . For example, consider the KL divergence  
29 between two Bernoulli distributions, the first with the probability of success given by 0.443 and the  
30 second with 0.975:

$$\text{31} \quad \text{KL} = 0.975 \log \frac{0.975}{0.443} + 0.025 \log \frac{0.025}{0.557} = 0.692 \text{ nats} \sim 1.0 \text{ bits.} \tag{5.24}$$

34 So it takes 1 bit of information to update from a [0.443, 0.557] distribution to a [0.975, 0.025] Bernoulli  
35 distribution. What about the reverse?

$$\text{36} \quad \text{KL} = 0.443 \log \frac{0.443}{0.975} + 0.557 \log \frac{0.557}{0.025} = 1.38 \text{ nats} \sim 2.0 \text{ bits,} \tag{5.25}$$

39 so it takes two bits, or twice as much information to move the other way. Thus we see that starting  
40 with a distribution that is nearly even and moving to one that is nearly certain takes about 1 bit of  
41 information, or one well designed yes/no question. To instead move us from near certainty in an  
42 outcome to something that is akin to the flip of a coin requires more persuasion.

43 The asymmetry of KL means that finding a  $p$  that is close to  $q$  by minimizing  $D_{\text{KL}}(p||q)$  gives  
44 different behavior than minimizing  $D_{\text{KL}}(q||p)$ . For example, consider the bimodal distribution  $q$   
45 shown in blue in Figure 5.1, which we approximate with a unimodal Gaussian. To prevent  $D_{\text{KL}}(q||p)$   
46 from becoming infinite, we must have  $p > 0$  whenever  $q > 0$  (i.e.,  $p$  must have support everywhere

47

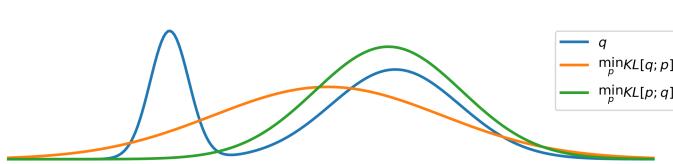


Figure 5.1: Demonstration of the mode-covering or mode-seeking behavior of KL divergence. The original distribution  $q$  is bimodal. When we minimize  $D_{\text{KL}}(q||p)$ , then  $p$  covers the modes of  $q$  (orange). When we minimize  $D_{\text{KL}}(p||q)$ , then  $p$  ignores some of the modes of  $q$  (green).

$q$  does), so  $p$  tends to *cover* both modes as it must be nonvanishing everywhere  $q$  is; this is called **mode-covering** or **zero-avoiding** behavior (orange curve). By contrast, to prevent  $D_{\text{KL}}(p||q)$  from becoming infinite, we must have  $p = 0$  whenever  $q = 0$ , which creates **mode-seeking** or **zero-forcing** behavior (green curve).

### 5.1.3.3 KL as expected weight of evidence

Imagine you have two different hypotheses you wish to select between, which we'll label  $P$  and  $Q$ . You collect some data  $D$ . Bayes' rule tells us how to update our beliefs in the hypotheses being correct:

$$\Pr(P|D) = \frac{\Pr(D|P)}{\Pr(D)} \Pr(P). \quad (5.26)$$

Normally this requires being able to evaluate the marginal likelihood  $\Pr(D)$ , which is difficult. If we instead consider the ratio of the probabilities for the two hypotheses:

$$\frac{\Pr(P|D)}{\Pr(Q|D)} = \frac{\Pr(D|P)}{\Pr(D|Q)} \frac{\Pr(P)}{\Pr(Q)}, \quad (5.27)$$

the marginal likelihood drops out. Taking the logarithm of both sides, and identifying the probability of the data under the model as the likelihood we find:

$$\log \frac{\Pr(P|D)}{\Pr(Q|D)} = \log \frac{p(D)}{q(D)} + \log \frac{\Pr(P)}{\Pr(Q)}. \quad (5.28)$$

The posterior log probability ratio for one hypothesis over the other is just our prior log probability ratio plus a term that IJ Good called the **weight of evidence** [Goo85]  $D$  for hypothesis  $P$  over  $Q$ :

$$w[P/Q; D] \triangleq \log \frac{p(D)}{q(D)}. \quad (5.29)$$

With this interpretation, the KL divergence is the expected weight of evidence for  $P$  over  $Q$  given by each observation, provided  $P$  were correct. Thus we see that data will (on average) add rather than subtract evidence towards the correct hypothesis, since KL divergence is always non-negative in expectation (see Section 5.1.2.2).

1 **5.1.4 Properties of KL**

3 Below are some other useful properties of the KL divergence.

5 **5.1.4.1 Compression Lemma**

7 An important general purpose result for the KL divergence is the Compression Lemma:

9 **Theorem 5.1.2.** *For any distributions  $P$  and  $Q$  with a well defined KL divergence, and for any*  
10 *scalar function  $\phi$  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 
$$g(x) = \frac{q(x)}{\mathcal{Z}} e^{\phi(x)}. \quad (5.31)$$

21 where the *partition function* is given by:

23 
$$\mathcal{Z} = \int dx q(x) e^{\phi(x)}. \quad (5.32)$$

26 Taking the KL divergence between  $p(x)$  and  $g(x)$  and rearranging gives the bound:

28 
$$D_{\text{KL}} (P \| G) = D_{\text{KL}} (P \| Q) - \mathbb{E}_P [\phi(x)] + \log(\mathcal{Z}) \geq 0. \quad (5.33)$$

30

31  $\square$

32

33 One way to view the compression lemma is that it provides what is termed the Donsker-Varadhan  
34 variational representation of the KL divergence:

36 
$$D_{\text{KL}} (P \| Q) = \sup_{\phi} \mathbb{E}_P [\phi(x)] - \log \mathbb{E}_Q [e^{\phi(x)}]. \quad (5.34)$$

38

39 In the space of all possible functions  $\phi$  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

$$\underline{4} \quad D_{\text{KL}}(p(x, y) \| q(x, y)) = \int dx \int dy p(x, y) \log \frac{p(x, y)}{q(x, y)} \quad (5.44)$$

5

6

$$\underline{7} \quad = - \int dy p(y) \int dx p(x|y) \log \left( \frac{q(y)}{p(y)} \frac{q(x|y)}{p(x|y)} \right) \quad (5.45)$$

8

9

$$\underline{10} \quad \geq - \int dy p(y) \log \left( \frac{q(y)}{p(y)} \int dx q(x|y) \right) \quad (5.46)$$

11

12

$$\underline{13} \quad = \int dy p(y) \log \frac{p(y)}{q(y)} = D_{\text{KL}}(p(y) \| q(y)) \quad (5.47)$$

14

15

$$(5.48)$$

16

□

17 One intuitive interpretation of this result is that if you only partially observe random variables, it  
18 is harder to distinguish between two candidate distributions than if you observed all of them.

### 19 20 5.1.5 KL divergence and MLE

21 Suppose we want to find the distribution  $q$  that is as close as possible to  $p$ , as measured by KL  
22 divergence:

23

$$\underline{24} \quad q^* = \arg \min_q D_{\text{KL}}(p \| q) = \arg \min_q \int p(x) \log p(x) dx - \int p(x) \log q(x) dx \quad (5.49)$$

25

26 Now suppose  $p$  is the empirical distribution, which puts a probability atom on the observed training  
27 data and zero mass everywhere else:

28

$$\underline{29} \quad p_{\mathcal{D}}(x) = \frac{1}{N} \sum_{n=1}^N \delta(x - x_n) \quad (5.50)$$

30

31 Using the sifting property of delta functions we get

32

$$\underline{33} \quad D_{\text{KL}}(p_{\mathcal{D}} \| q) = - \int p_{\mathcal{D}}(x) \log q(x) dx + C \quad (5.51)$$

34

35

$$\underline{36} \quad = - \int \left[ \frac{1}{N} \sum_n \delta(x - x_n) \right] \log q(x) dx + C \quad (5.52)$$

37

38

$$\underline{39} \quad = - \frac{1}{N} \sum_n \log q(x_n) + C \quad (5.53)$$

40

41 where  $C = \int p_{\mathcal{D}}(x) \log p_{\mathcal{D}}(x)$  is a constant independent of  $q$ .

42 We can rewrite the above as follows

43

$$\underline{44} \quad D_{\text{KL}}(p_{\mathcal{D}} \| q) = \mathbb{H}(p_{\mathcal{D}}, q) - \mathbb{H}(p_{\mathcal{D}}) \quad (5.54)$$

45

46

1  
2 where

3  
4  $\mathbb{H}(p, q) \triangleq -\sum_k p_k \log q_k$  (5.55)

5

6 is known as the **cross entropy**. The quantity  $\mathbb{H}(p_D, q)$  is the average negative log likelihood of  $q$   
7 evaluated on the training set. Thus we see that minimizing KL divergence to the empirical distribution  
8 is equivalent to maximizing likelihood.

9 This perspective points out the flaw with likelihood-based training, namely that it puts too  
10 much weight on the training set. In most applications, we do not really believe that the empirical  
11 distribution is a good representation of the true distribution, since it just puts “spikes” on a finite  
12 set of points, and zero density everywhere else. Even if the dataset is large (say 1M images), the  
13 universe from which the data is sampled is usually even larger (e.g., the set of “all natural images”)  
14 is much larger than 1M). Thus we need to somehow smooth the empirical distribution by sharing  
15 probability mass between “similar” inputs.  
16

### 17 18 5.1.6 KL divergence and Bayesian Inference

19 Bayesian inference itself can be motivated as the solution to a particular minimization problem of  
20 KL.

21 Consider a prior set of beliefs described by a joint distribution  $q(\theta, D) = q(\theta)q(D|\theta)$ , involving  
22 some *prior*  $q(\theta)$  and some *likelihood*  $q(D|\theta)$ . If we happen to observe some particular dataset  $D_0$ ,  
23 how should we update our beliefs? We could search for the joint distribution that is as close as  
24 possible to our prior beliefs but that respects the constraint that we now know the value of the data:  
25

26  $p(\theta, D) = \min D_{\text{KL}}(p(\theta, D) \| q(\theta, D))$  such that  $p(D) = \delta(D - D_0)$ . (5.56)

27

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  $D_{\text{KL}}(p(\theta, D) \| q(\theta, D)) = D_{\text{KL}}(p(D) \| q(D)) + D_{\text{KL}}(p(\theta|D) \| q(\theta|D)),$  (5.57)

32

33 makes clear that the solution is given by the joint distribution:

35  $p(\theta, D) = p(D)p(\theta|D) = \delta(D - D_0)q(\theta|D).$  (5.58)

36

37 Our updated beliefs have a marginal over the  $\theta$

39  $p(\theta) = \int dD p(\theta, D) = \int dD \delta(D - D_0)q(\theta|D) = q(\theta|D = D_0),$  (5.59)

40

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  $q(\theta, D) = q(D)q(\theta|D) = q(\theta)q(D|\theta) \implies q(\theta|D) = \frac{q(D|\theta)}{q(D)}q(\theta).$  (5.60)

46

47

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

8 If there was some additional measurement error that was well understood, we ought to instead of  
9 pegging out updated beliefs to be a delta function on the observed data, simply peg it to be the well  
10 understood distribution  $p(D)$ . For example, we might not know the precise value the data takes, but  
11 believe after measuring things that it is a Gaussian distribution with a certain mean and standard  
12 deviation.

13 Because of the chain rule of KL, this has no effect on our updated conditional distribution over  
14 parameters, which remains the Bayesian posterior:  $p(\theta|D) = q(\theta|D)$ . However, this does change our  
15 marginal beliefs about the parameters, which are now:

$$\underline{16} \quad p(\theta) = \int dD p(D)q(\theta|D). \quad (5.61)$$

17 This generalization of Bayes' rule is sometimes called **Jeffrey's conditionalization rule** [Cat08].  
18

### 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  $\mathcal{T}(\mathbf{x})$ :

$$\underline{25} \quad p(\mathbf{x}) = h(\mathbf{x}) \exp[\boldsymbol{\eta}^\top \mathcal{T}(\mathbf{x}) - A(\boldsymbol{\eta})] \quad (5.62)$$

27 where

$$\underline{29} \quad A(\boldsymbol{\eta}) = \log \int h(\mathbf{x}) \exp(\boldsymbol{\eta}^\top \mathcal{T}(\mathbf{x})) d\mathbf{x} \quad (5.63)$$

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:

$$\underline{34} \quad D_{\text{KL}}(p(\mathbf{x}|\boldsymbol{\eta}_1) \| p(\mathbf{x}|\boldsymbol{\eta}_2)) = \mathbb{E}_{\boldsymbol{\eta}_1} [(\boldsymbol{\eta}_1 - \boldsymbol{\eta}_2)^\top \mathcal{T}(\mathbf{x}) - A(\boldsymbol{\eta}_1) + A(\boldsymbol{\eta}_2)] \quad (5.64)$$

$$\underline{35} \quad = (\boldsymbol{\eta}_1 - \boldsymbol{\eta}_2)^\top \boldsymbol{\mu}_1 - A(\boldsymbol{\eta}_1) + A(\boldsymbol{\eta}_2) \quad (5.65)$$

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

$$\begin{aligned} \underline{43} \quad & D_{\text{KL}}(\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) \| \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)) \\ \underline{44} \quad &= \frac{1}{2} \left[ \text{tr}(\boldsymbol{\Sigma}_2^{-1} \boldsymbol{\Sigma}_1) + (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^\top \boldsymbol{\Sigma}_2^{-1} (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) - D + \log \left( \frac{\det(\boldsymbol{\Sigma}_2)}{\det(\boldsymbol{\Sigma}_1)} \right) \right] \end{aligned} \quad (5.66)$$

1 In the scalar case, this becomes  
2

$$\underline{3} \quad D_{\text{KL}}(\mathcal{N}(x|\mu_1, \sigma_1) \parallel \mathcal{N}(x|\mu_2, \sigma_2)) = \log \frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2} \quad (5.67)$$

### 6 5.1.7.2 Connection with Bregman divergence 7

8 Recall that the log partition function  $A(\boldsymbol{\eta})$  is a convex function. We can therefore use it to define  
9 the Bregman divergence (Section 6.5.1) between the two distributions,  $p$  and  $q$ , as follows:

$$\underline{10} \quad B_f(\boldsymbol{\eta}_q \parallel \boldsymbol{\eta}_p) = A(\boldsymbol{\eta}_q) - A(\boldsymbol{\eta}_p) - (\boldsymbol{\eta}_q - \boldsymbol{\eta}_p)^\top \nabla_{\boldsymbol{\eta}_p} A(\boldsymbol{\eta}_p) \quad (5.68)$$

$$\underline{12} \quad = A(\boldsymbol{\eta}_q) - A(\boldsymbol{\eta}_p) - (\boldsymbol{\eta}_q - \boldsymbol{\eta}_p)^\top \mathbb{E}_p [\mathcal{T}(\mathbf{x})] \quad (5.69)$$

$$\underline{13} \quad = D_{\text{KL}}(p \parallel q) \quad (5.70)$$

14 where we exploited the fact that the gradient of the log partition function computes the expected  
15 sufficient statistics as shown in Section 2.5.3.

16 In fact, the KL divergence is the only divergence that is both a Bregman divergence and an  
17  $f$ -divergence (See Section 2.9.1) [Ama09].

## 20 5.2 Entropy 21

22 In this section, we discuss the **entropy** of a distribution  $p$ , which is just a shifted and scaled version  
23 of the KL divergence between the probability distribution and the uniform distribution, as we will  
24 see.

### 26 5.2.1 Definition

27 The entropy of a discrete random variable  $X$  with distribution  $p$  over  $K$  states is defined by  
28

$$\underline{29} \quad \mathbb{H}(X) \triangleq - \sum_{k=1}^K p(X=k) \log_2 p(X=k) = -\mathbb{E}_X [\log p(X)] \quad (5.71)$$

32 This is equivalent to a constant minus the KL divergence from the uniform distribution:

$$\underline{33} \quad \mathbb{H}(X) = \log K - D_{\text{KL}}(p(X) \parallel u(X)) \quad (5.72)$$

$$\underline{35} \quad D_{\text{KL}}(p(X) \parallel u(X)) = \sum_{k=1}^K p(X=k) \log \frac{p(X=k)}{\frac{1}{K}} \quad (5.73)$$

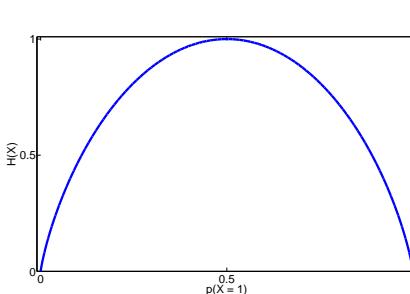
$$\underline{38} \quad = \log K + \sum_{k=1}^K p(X=k) \log p(X=k) \quad (5.74)$$

41 If  $p$  is uniform, the KL is zero, and we see that the entropy achieves its maximal value of  $\log K$ .

42 For the special case of binary random variables,  $X \in \{0, 1\}$ , we can write  $p(X=1) = \theta$  and  
43  $p(X=0) = 1 - \theta$ . Hence the entropy becomes

$$\underline{44} \quad \mathbb{H}(X) = -[p(X=1) \log_2 p(X=1) + p(X=0) \log_2 p(X=0)] \quad (5.75)$$

$$\underline{46} \quad = -[\theta \log_2 \theta + (1 - \theta) \log_2 (1 - \theta)] \quad (5.76)$$



11 Figure 5.2: Entropy of a Bernoulli random variable as a function of  $\theta$ . The maximum entropy is  $\log_2 2 = 1$ .  
12 Generated by [bernoulli\\_entropy\\_fig.py](#).

13  
14  
15 This is called the **binary entropy function**, and is also written  $\mathbb{H}(\theta)$ . We plot this in Figure 5.2.  
16 We see that the maximum value of 1 bit occurs when the distribution is uniform,  $\theta = 0.5$ . A fair coin  
17 requires a single yes/no question to determine its state.

18  
19 **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

$$\underline{23} \quad h(X) \triangleq - \int_{\mathcal{X}} dx p(x) \log p(x) \quad (5.77)$$

24 assuming this integral exists.

25 For example, one can show that the entropy of a  $d$ -dimensional Gaussian is

$$\underline{26} \quad h(\mathcal{N}(\mu, \Sigma)) = \frac{1}{2} \ln |2\pi e \Sigma| = \frac{1}{2} \ln [(2\pi e)^d |\Sigma|] = \frac{d}{2} + \frac{d}{2} \ln(2\pi) + \frac{1}{2} \ln |\Sigma| \quad (5.78)$$

27 In the 1d case, this becomes

$$\underline{28} \quad h(\mathcal{N}(\mu, \sigma^2)) = \frac{1}{2} \ln [2\pi e \sigma^2] \quad (5.79)$$

29 Note that, unlike the discrete case, *differential entropy can be negative*. This is because pdf's can  
30 be bigger than 1. For example, suppose  $X \sim U(0, a)$ . Then

$$\underline{31} \quad h(X) = - \int_0^a dx \frac{1}{a} \log \frac{1}{a} = \log a \quad (5.80)$$

32 If we set  $a = 1/8$ , we have  $h(X) = \log_2(1/8) = -3$ .

33 One way to understand differential entropy is to realize that all real-valued quantities can only be  
34 represented to finite precision. It can be shown [CT91, p228] that the entropy of an  $n$ -bit quantization  
35 of a continuous random variable  $X$  is approximately  $h(X) + n$ . For example, suppose  $X \sim U(0, \frac{1}{8})$ .  
36 Then in a binary representation of  $X$ , the first 3 bits to the right of the binary point must be 0 (since  
37 the number is  $\leq 1/8$ ). So to describe  $X$  to  $n$  bits of accuracy only requires  $n - 3$  bits, which agrees  
38 with  $h(X) = -3$  calculated above.

39  
40  
41  
42  
43  
44  
45  
46  
47

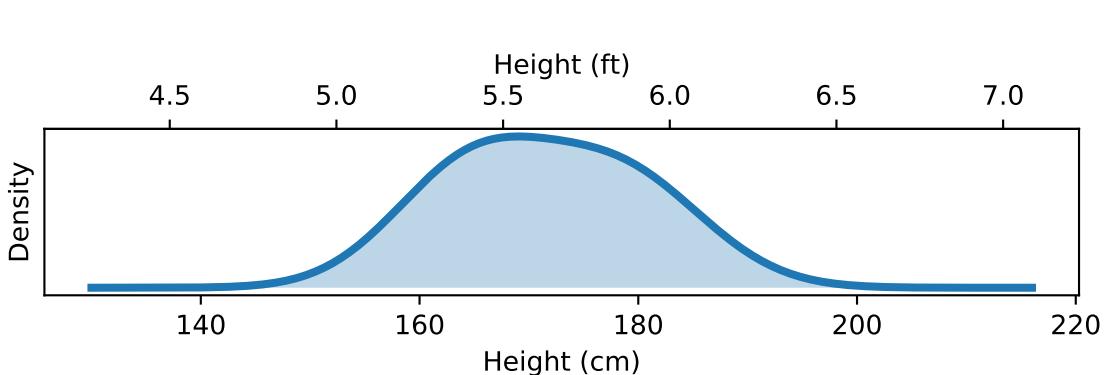


Figure 5.3: Distribution of adult heights. The continuous entropy of the distribution depends on its units of measurement. If heights are measured in feet, this distribution has a continuous entropy of 0.43 bits. If measured in centimeters it's 5.4 bits. If measured in meters it's -1.3 bits. Data taken from <https://ourworldindata.org/human-height>.

The continuous entropy also lacks the reparameterization independence of KL divergence Section 5.1.2.3. In particular, if we transform our random variable  $y = f(x)$ , the entropy transforms. To see this, note that the change of variables tells us that

$$p(y) dy = p(x) dx \implies p(y) = p(x) \left| \frac{dy}{dx} \right|^{-1}, \quad (5.81)$$

Thus the continuous entropy transforms as follows:

$$h(X) = - \int dx p(x) \log p(x) = h(Y) - \int dy p(y) \log \left| \frac{dy}{dx} \right|. \quad (5.82)$$

We pick up a factor in the continuous entropy of the log of the determinant of the Jacobian of the transformation. This changes the value for the continuous entropy even for simply rescaling the random variable such as when we change units. For example in Figure 5.3 we show the distribution of adult human heights (it is bimodal because while both male and female heights are normally distributed, they differ noticeably). The continuous entropy of this distribution depends on the units it is measured in. If measured in feet, the continuous entropy is 0.43 bits. Intuitively this is because human heights mostly span less than a foot. If measured in centimeters it is instead 5.4 bits. There are 30.48 centimeters in a foot,  $\log_2 30.48 = 4.9$  explaining the difference. If we measured the continuous entropy of the same distribution measured in meters we would obtain -1.3 bits!

### 5.2.3 Typical sets

The **typical set** of a probability distribution is the set whose elements have an information content that is close to that of the expected information content from random samples from the distribution. More precisely, for a distribution  $p(\mathbf{x})$  with support  $\mathbf{x} \in \mathcal{X}$ , the  $\epsilon$ -typical set  $\mathcal{A}_\epsilon^N \in \mathcal{X}^N$  for  $p(\mathbf{x})$  is

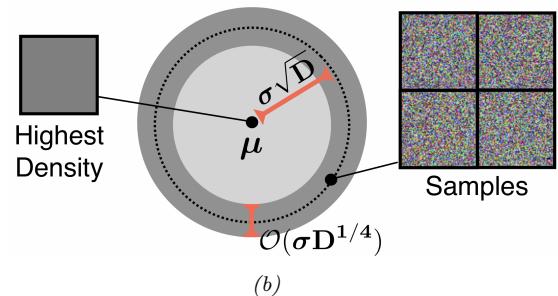
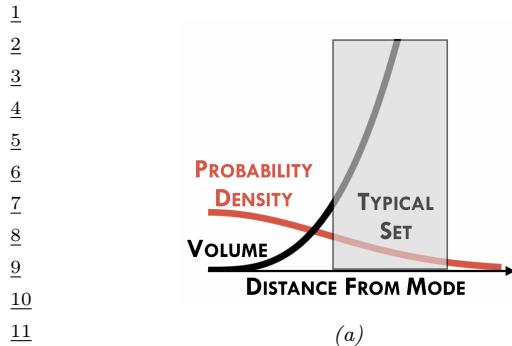


Figure 5.4: (a) Cartoon illustration of why the typical set of a Gaussian is not centered at the mode of the distribution. (b) Illustration of the typical set of a Gaussian, which is concentrated in a thin annulus of thickness  $\sigma D^{1/4}$  and distance  $\sigma D^{1/2}$  from the origin. We also 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.<sup>1</sup>

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

<sup>1</sup> For a more detailed explanation, see this blog post by Ferenc Huszar: <https://www.inference.vc/high-dimensional-gaussian-distributions-are-soap-bubble/>.

1 expected squared distance is given by  $\mathbb{E}[d^2] = \sum_{i=1}^D \mathbb{E}[x_i^2] = D$ , and the variance of the squared  
2 distance is given by  $\mathbb{V}[d^2] = \sum_{i=1}^D \mathbb{V}[x_i^2] = D$ . As  $D$  grows, the coefficient of variation (i.e., the SD  
3 relative to the mean) goes to zero:

$$\lim_{D \rightarrow \infty} \frac{\text{std}[d^2]}{\mathbb{E}[d^2]} = \lim_{D \rightarrow \infty} \frac{\sqrt{D}}{D} = 0 \quad (5.84)$$

8 Thus the expected square distance concentrates around  $D$ , so the expected distance concentrates  
9 around  $\mathbb{E}[d(\mathbf{x})] = \sqrt{D}$ . See [Ver18] for a more rigorous proof.

#### 11 5.2.4 Cross entropy and perplexity

13 A standard way to measure how close a model  $q$  is to a true distribution  $p$  is in terms of the KL  
14 divergence (Section 5.1), given by

$$D_{\text{KL}}(p\|q) = \sum_x p(x) \log \frac{p(x)}{q(x)} = \mathbb{H}(p, q) - \mathbb{H}(p) \quad (5.85)$$

19 where  $\mathbb{H}(p, q)$  is the **cross entropy**

$$\mathbb{H}(p, q) = - \sum_x p(x) \log q(x) \quad (5.86)$$

23 and  $\mathbb{H}(p) = \mathbb{H}(p, p)$  is the entropy, which is a constant independent of the model.

25 In language modeling, it is common to report an alternative performance measure known as the  
26 **perplexity**. This is defined as

$$\text{perplexity}(p, q) \triangleq 2^{\mathbb{H}(p, q)} \quad (5.87)$$

29 We can compute an empirical approximation to the cross entropy as follows. Suppose we approxi-  
30 mate the true distribution with an empirical distribution based on data sampled from  $p$ :

$$p_{\mathcal{D}}(x|\mathcal{D}) = \frac{1}{N} \sum_{n=1}^N \mathbb{I}(x = x_n) \quad (5.88)$$

35 In this case, the cross entropy is given by

$$H = - \frac{1}{N} \sum_{n=1}^N \log p(x_n) = - \frac{1}{N} \log \prod_{n=1}^N p(x_n) \quad (5.89)$$

40 The corresponding perplexity is given by

$$\text{perplexity}(p_{\mathcal{D}}, p) = 2^{-\frac{1}{N} \log(\prod_{n=1}^N p(x_n))} = 2^{\log(\prod_{n=1}^N p(x_n)) - \frac{1}{N}} \quad (5.90)$$

$$= \left( \prod_{n=1}^N p(x_n) \right)^{-1/N} = \sqrt[N]{\prod_{n=1}^N \frac{1}{p(x_n)}} \quad (5.91)$$

In the case of language models, we usually condition on previous words when predicting the next word. For example, in a bigram model, we use a second order Markov model of the form  $p(x_n|x_{n-1})$ . We define the **branching factor** of a language model as the number of possible words that can follow any given word. For example, suppose the model predicts that each word is equally likely, regardless of context, so  $p(x_n|x_{n-1}) = 1/K$ , where  $K$  is the number of words in the vocabulary. Then the perplexity is  $((1/K)^N)^{-1/N} = K$ . If some symbols are more likely than others, and the model correctly reflects this, its perplexity will be lower than  $K$ . However, we have  $\mathbb{H}(p^*) \leq \mathbb{H}(p^*, p)$ , so we can never reduce the perplexity below  $2^{-\mathbb{H}(p^*)}$ .

10

## 11 5.3 Mutual information

12

13 The KL divergence gave us a way to measure how similar two distributions were. How should we  
 14 measure how dependant two random variables are? One thing we could do is turn the question  
 15 of measuring the dependence of two random variables into a question about the similarity of their  
 16 distributions. This gives rise to the notion of **mutual information** (MI) between two random  
 17 variables, which we define below.

18

### 19 5.3.1 Definition

20

21 The mutual information between rv's  $X$  and  $Y$  is defined as follows:

$$22 \quad \mathbb{I}(X; Y) \triangleq D_{\text{KL}}(p(x, y) \| p(x)p(y)) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \quad (5.92)$$

25 (We write  $\mathbb{I}(X; Y)$  instead of  $\mathbb{I}(X, Y)$ , in case  $X$  and/or  $Y$  represent sets of variables; for example, we  
 26 can write  $\mathbb{I}(X; Y, Z)$  to represent the MI between  $X$  and  $(Y, Z)$ .) For continuous random variables,  
 27 we just replace sums with integrals.

28 It is easy to see that MI is always non-negative, even for continuous random variables, since

29

$$30 \quad \mathbb{I}(X; Y) = D_{\text{KL}}(p(x, y) \| p(x)p(y)) \geq 0 \quad (5.93)$$

31

32 We achieve the bound of 0 iff  $p(x, y) = p(x)p(y)$ .

33

### 34 5.3.2 Interpretation

35

36 Knowing that the mutual information is a KL divergence between the joint and factored marginal  
 37 distributions tells us that the MI measures the information gain if we update from a model that treats  
 38 the two variables as independent  $p(x)p(y)$  to one that models their true joint density  $p(x, y)$ .

39 To gain further insight into the meaning of MI, it helps to re-express it in terms of joint and  
 40 conditional entropies, as follows:

$$41 \quad \mathbb{I}(X; Y) = \mathbb{H}(X) - \mathbb{H}(X|Y) = \mathbb{H}(Y) - \mathbb{H}(Y|X) \quad (5.94)$$

42

43 Thus we can interpret the MI between  $X$  and  $Y$  as the reduction in uncertainty about  $X$  after  
 44 observing  $Y$ , or, by symmetry, the reduction in uncertainty about  $Y$  after observing  $X$ . Incidentally,  
 45 this result gives an alternative proof that conditioning, on average, reduces entropy. In particular, we  
 46 have  $0 \leq \mathbb{I}(X; Y) = \mathbb{H}(X) - \mathbb{H}(X|Y)$ , and hence  $\mathbb{H}(X|Y) \leq \mathbb{H}(X)$ .

47

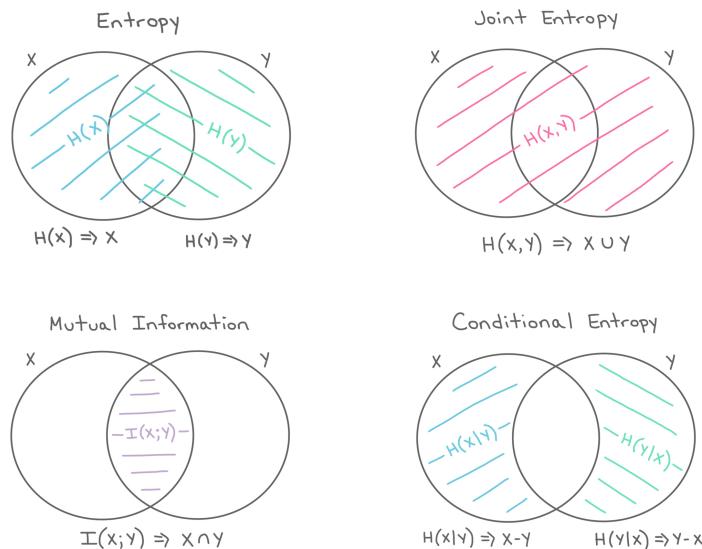


Figure 5.5: The marginal entropy, joint entropy, conditional entropy and mutual information represented as information diagrams. Used with kind permission of Katie Everett.

We can also obtain a different interpretation. One can show that

$$\mathbb{I}(X;Y) = \mathbb{H}(X,Y) - \mathbb{H}(X|Y) - \mathbb{H}(Y|X) \quad (5.95)$$

Finally, one can show that

$$\mathbb{I}(X;Y) = \mathbb{H}(X) + \mathbb{H}(Y) - \mathbb{H}(X,Y) \quad (5.96)$$

See Figure 5.5 for a summary of these equations in terms of an **information diagram**. (Formally, this is a signed measure mapping set expressions to their information-theoretic counterparts [Yeu91a].)

### 5.3.3 Data processing inequality

Suppose we have an unknown variable  $X$ , and we observe a noisy function of it, call it  $Y$ . If we process the noisy observations in some way to create a new variable  $Z$ , it should be intuitively obvious that we cannot increase the amount of information we have about the unknown quantity,  $X$ . This is known as the **data processing inequality**. We now state this more formally, and then prove it.

**Theorem 5.3.1.** Suppose  $X \rightarrow Y \rightarrow Z$  forms a Markov chain, so that  $X \perp Z|Y$ . Then  $\mathbb{I}(X;Y) \geq \mathbb{I}(X;Z)$ .

*Proof.* By the chain rule for mutual information we can expand the mutual information in two different ways:

$$\mathbb{I}(X;Y,Z) = \mathbb{I}(X;Z) + \mathbb{I}(X;Y|Z) \quad (5.97)$$

$$= \mathbb{I}(X;Y) + \mathbb{I}(X;Z|Y) \quad (5.98)$$

1 Since  $X \perp Z|Y$ , we have  $\mathbb{I}(X; Z|Y) = 0$ , so

3

$$\mathbb{I}(X; Z) + \mathbb{I}(X; Y|Z) = \mathbb{I}(X; Y) \quad (5.99)$$

4

5 Since  $\mathbb{I}(X; Y|Z) \geq 0$ , we have  $\mathbb{I}(X; Y) \geq \mathbb{I}(X; Z)$ . Similarly one can prove that  $\mathbb{I}(Y; Z) \geq \mathbb{I}(X; Z)$ .

6

7

8

9

### 10 5.3.4 Sufficient Statistics

11 An important consequence of the DPI is the following. Suppose we have the chain  $\theta \rightarrow X \rightarrow s(X)$ .

12 Then

13

14  $\mathbb{I}(\theta; s(X)) \leq \mathbb{I}(\theta; X) \quad (5.100)$

15

16 If this holds with equality, then we say that  $s(X)$  is a **sufficient statistic** of the data  $X$  for the purposes of inferring  $\theta$ . In this case, we can equivalently write  $\theta \rightarrow s(X) \rightarrow X$ , since we can reconstruct the data from knowing  $s(X)$  just as accurately as from knowing  $\theta$ .

17 An example of a sufficient statistic is the data itself,  $s(X) = X$ , but this is not very useful, since it doesn't summarize the data at all. Hence we define a **minimal sufficient statistic**  $s(X)$  as one which is sufficient, and which contains no extra information about  $\theta$ ; thus  $s(X)$  maximally compresses the data  $X$  without losing information which is relevant to predicting  $\theta$ . More formally, we say  $s$  is a minimal sufficient statistic for  $X$  if  $s(X) = f(s'(X))$  for some function  $f$  and all sufficient statistics  $s'(X)$ . We can summarize the situation as follows:

26  $\theta \rightarrow s(X) \rightarrow s'(X) \rightarrow X \quad (5.101)$

27

28 Here  $s'(X)$  takes  $s(X)$  and adds redundant information to it, thus creating a one-to-many mapping.

29 For example, a minimal sufficient statistic for a set of  $N$  Bernoulli trials is simply  $N$  and  $N_1 = \sum_n \mathbb{I}(X_n = 1)$ , i.e., the number of successes. In other words, we don't need to keep track of the entire sequence of heads and tails and their ordering, we only need to keep track of the total number of heads and tails. Similarlt, for inferring the mean of a Gaussian distribution with known variance we only need to know the empirical mean and number of samples.

30 Earlier in Section 5.1.7 we motivated the exponential family of distributions as being the ones that are minimal in the sense that they contain no other information than constraints on some statistics of the data. It makes sense then that the statistics used to generate exponential family distributions are sufficient. It also hints at the more remarkable fact of the **Pitman-Koopman-Darmois theorem**, which says that for any distribution whose domain is fixed, it is only the exponential family that admits sufficient statistics with bounded dimensionality as the number of samples increases Diaconis [Dia88].

41

### 43 5.3.5 Multivariate mutual information

44 There are several ways to generalize the idea of mutual information to a set of random variables as we discuss below.

47

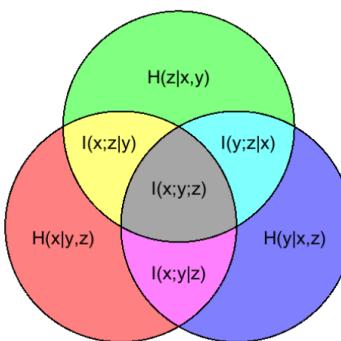


Figure 5.6: Illustration of multivariate mutual information between three random variables. From [https://en.wikipedia.org/wiki/Mutual\\_information](https://en.wikipedia.org/wiki/Mutual_information). Used with kind permission of Wikipedia author PAR.

### 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

$$\underline{3} \quad \underline{4} \quad \mathbb{I}(X; Y; Z) = \mathbb{I}(X; Y) - \mathbb{I}(X; Y|Z) \quad (5.106)$$

$$\underline{5} \quad \underline{6} \quad = \mathbb{I}(X; Z) - \mathbb{I}(X; Z|Y) \quad (5.107)$$

$$\underline{7} \quad \underline{8} \quad = \mathbb{I}(Y; Z) - \mathbb{I}(Y; Z|X) \quad (5.108)$$

This can be interpreted as the change in mutual information between two pairs of variables when conditioning on the third. Note that this quantity is symmetric in its arguments.

By the definition of conditional mutual information, we have

$$\underline{11} \quad \underline{12} \quad \mathbb{I}(X; Z|Y) = \mathbb{I}(Z; X, Y) - \mathbb{I}(Y; Z) \quad (5.109)$$

Hence we can rewrite Equation (5.107) as follows:

$$\underline{15} \quad \underline{16} \quad \mathbb{I}(X; Y; Z) = \mathbb{I}(X; Z) + \mathbb{I}(Y; Z) - \mathbb{I}(X, Y; Z) \quad (5.110)$$

This tells us that the MMI is the difference between how much we learn about  $Z$  given  $X$  and  $Y$  individually vs jointly (see also Section 5.3.5.3).

The 3-way MMI is illustrated in the information diagram in Figure 5.6. The way to interpret such diagrams when we have multiple variables is as follows: the area of a shaded area that includes circles  $A, B, C, \dots$  and excludes circles  $F, G, H, \dots$  represents  $\mathbb{I}(A; B; C; \dots | F, G, H, \dots)$ ; if  $B = C = \emptyset$ , this is just  $\mathbb{I}(A|F, G, H, \dots)$ ; if  $F = G = H = \emptyset$ , this is just  $\mathbb{I}(A; B; C, \dots)$ .

### 5.3.5.3 Synergy and redundancy

The MMI is  $\mathbb{I}(X; Y; Z) = \mathbb{I}(X; Z) + \mathbb{I}(Y; Z) - \mathbb{I}(X, Y; Z)$ . We see that this can be positive, zero or negative. If some of the information about  $Z$  that is provided by  $X$  is also provided by  $Y$ , then there is some **redundancy** between  $X$  and  $Y$  (wrt  $Z$ ). In this case,  $\mathbb{I}(X; Z) + \mathbb{I}(Y; Z) > \mathbb{I}(X, Y; Z)$ , so (from Equation (5.110)) we see that the MMI will be positive. If, by contrast, we learn more about  $Z$  when we see  $X$  and  $Y$  together, we say there is some **synergy** between them. In this case,  $\mathbb{I}(X; Z) + \mathbb{I}(Y; Z) < \mathbb{I}(X, Y; Z)$ , so the MMI will be negative.

### 5.3.5.4 MMI and causality

The sign of the MMI can be used to distinguish between different kinds of directed graphical models, which can sometimes be interpreted causally (see Chapter 38 for a general discussion of causality).

For example, consider a model of the form  $X \leftarrow Z \rightarrow Y$ , where  $Z$  is a “cause” of  $X$  and  $Y$ . For example, suppose  $X$  represents the event it is raining,  $Y$  represents the event that the sky is dark, and  $Z$  represents the event that the sky is cloudy. Conditioning on the common cause  $Z$  renders the children  $X$  and  $Y$  independent, since if I know it is cloudy, noticing that the sky is dark does not change my beliefs about whether it will rain or not. Consequently  $\mathbb{I}(X; Y|Z) \leq \mathbb{I}(X; Y)$ , so  $\mathbb{I}(\{X, Y, Z\}) \geq 0$ .

Now consider the case where  $Z$  is a common effect,  $X \rightarrow Z \leftarrow Y$ . In this case, conditioning on  $Z$  makes  $X$  and  $Y$  dependent, due to the explaining away phenomenon (see Section 4.2.3.2). For example, if  $X$  and  $Y$  are independent random bits, and  $Z$  is the XOR of  $X$  and  $Y$ , then observing  $Z = 1$  means that  $p(X \neq Y|Z = 1) = 1$ , so  $X$  and  $Y$  are now dependent (information-theoretically),

1 not causally), even though they were a priori independent. Consequently  $\mathbb{I}(X;Y|Z) \geq \mathbb{I}(X;Y)$ , so  
2  $\mathbb{I}(X;Y;Z) \leq 0$ .

4 Finally, consider a Markov chain,  $X \rightarrow Y \rightarrow Z$ . We have  $\mathbb{I}(X;Z|Y) \leq \mathbb{I}(X;Z)$  and so the MMI  
5 must be positive.

### 7 5.3.5.5 MMI and entropy

9 We can also write the MMI in terms of entropies. Specifically, we know that

$$\mathbb{I}(X;Y) = \mathbb{H}(X) + \mathbb{H}(Y) - \mathbb{H}(X,Y) \quad (5.111)$$

12 and

$$\mathbb{I}(X;Y|Z) = \mathbb{H}(X,Z) + \mathbb{H}(Y,Z) - \mathbb{H}(Z) - \mathbb{H}(X,Y,Z) \quad (5.112)$$

16 Hence we can rewrite Equation (5.106) as follows:

$$\mathbb{I}(X;Y;Z) = [\mathbb{H}(X) + \mathbb{H}(Y) + \mathbb{H}(Z)] - [\mathbb{H}(X,Y) + \mathbb{H}(X,Z) + \mathbb{H}(Y,Z)] + \mathbb{H}(X,Y,Z) \quad (5.113)$$

19 Contrast this to Equation (5.104).

21 More generally, we have

$$\mathbb{I}(X_1, \dots, X_D) = - \sum_{\mathcal{T} \subseteq \{1, \dots, D\}} (-1)^{|\mathcal{T}|} \mathbb{H}(\mathcal{T}) \quad (5.114)$$

25 For sets of size 1, 2 and 3 this expands as follows:

$$I_1 = H_1 \quad (5.115)$$

$$I_{12} = H_1 + H_2 - H_{12} \quad (5.116)$$

$$I_{123} = H_1 + H_2 + H_3 - H_{12} - H_{13} - H_{23} + H_{123} \quad (5.117)$$

31 We can use the **Mobius inversion formula** to derive the following dual relationship:

$$\mathbb{H}(\mathcal{S}) = - \sum_{\mathcal{T} \subseteq \mathcal{S}} (-1)^{|\mathcal{T}|} \mathbb{I}(\mathcal{T}) \quad (5.118)$$

36 for sets of variables  $\mathcal{S}$ .

37 Using the chain rule for entropy, we can also derive the following expression for the 3-way MMI:

$$\mathbb{I}(X;Y;Z) = \mathbb{H}(Z) - \mathbb{H}(Z|X) - \mathbb{H}(Z|Y) + \mathbb{H}(Z|X,Y) \quad (5.119)$$

### 41 5.3.6 Variational bounds on mutual information

43 In this section, we discuss methods for computing upper and lower bounds on MI that use variational  
44 approximations to the intractable distributions. This can be useful for representation learning  
45 (Chapter 34). This approach was first suggested in [BA03]. For a more detailed overview of  
46 variational bounds on mutual information, see Poole et al. [Poo+19].

1    2 **5.3.6.1 Upper bound**

3 Suppose that the joint  $p(\mathbf{x}, \mathbf{y})$  is intractable to evaluate, but that we can sample from  $p(\mathbf{x})$  and  
4 evaluate the conditional distribution  $p(\mathbf{y}|\mathbf{x})$ . Furthermore, suppose we approximate  $p(\mathbf{y})$  by  $q(\mathbf{y})$ .  
5 Then we can compute an upper bound on the MI as follows:  
6

7    8  $\mathbb{I}(\mathbf{x}; \mathbf{y}) = \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} \left[ \log \frac{p(\mathbf{y}|\mathbf{x})q(\mathbf{y})}{p(\mathbf{y})q(\mathbf{y})} \right] \quad (5.120)$   
9

10    11  $= \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} \left[ \log \frac{p(\mathbf{y}|\mathbf{x})}{q(\mathbf{y})} \right] - D_{\text{KL}}(p(\mathbf{y})\|q(\mathbf{y})) \quad (5.121)$

12    13  $\leq \mathbb{E}_{p(\mathbf{x})} \left[ \mathbb{E}_{p(\mathbf{y}|\mathbf{x})} \left[ \log \frac{p(\mathbf{y}|\mathbf{x})}{q(\mathbf{y})} \right] \right] \quad (5.122)$

14    15  $= \mathbb{E}_{p(\mathbf{x})} [D_{\text{KL}}(p(\mathbf{y}|\mathbf{x})\|q(\mathbf{y}))] \quad (5.123)$

16 This bound is tight if  $q(\mathbf{y}) = p(\mathbf{y})$ .

17 What's happening here is that  $\mathbb{I}(Y; X) = \mathbb{H}(Y) - \mathbb{H}(Y|X)$  and we've assumed we know  $p(\mathbf{y}|\mathbf{x})$   
18 and so can estimate  $\mathbb{H}(Y|X)$  well. While we don't know  $\mathbb{H}(Y)$ , we can upper bound it using some  
19 model  $q(\mathbf{y})$ . Our model can never do better than  $p(\mathbf{y})$  itself (the non-negativity of KL), so our  
20 entropy estimate errs too large, and hence our MI estimate will be an upper bound.  
21

22    23 **5.3.6.2 BA lower bound**

24 Suppose that the joint  $p(\mathbf{x}, \mathbf{y})$  is intractable to evaluate, but that we can evaluate  $p(\mathbf{x})$ . Furthermore,  
25 suppose we approximate  $p(\mathbf{x}|\mathbf{y})$  by  $q(\mathbf{x}|\mathbf{y})$ . Then we can derive the following variational lower bound  
26 on the mutual information:  
27

28    29  $\mathbb{I}(\mathbf{x}; \mathbf{y}) = \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} \left[ \log \frac{p(\mathbf{x}|\mathbf{y})}{p(\mathbf{x})} \right] \quad (5.124)$   
30

31    32  $= \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} \left[ \log \frac{q(\mathbf{x}|\mathbf{y})}{p(\mathbf{x})} \right] + \mathbb{E}_{p(\mathbf{y})} [D_{\text{KL}}(p(\mathbf{x}|\mathbf{y})\|q(\mathbf{x}|\mathbf{y}))] \quad (5.125)$

33    34  $\geq \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} \left[ \log \frac{q(\mathbf{x}|\mathbf{y})}{p(\mathbf{x})} \right] = \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} [\log q(\mathbf{x}|\mathbf{y})] + h(\mathbf{x}) \quad (5.126)$   
35

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    40 **5.3.6.3 NWJ lower bound**

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    46  $q(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{x})e^{f(\mathbf{x}, \mathbf{y})}}{Z(\mathbf{y})} \quad (5.127)$   
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

$$\mathbb{E}_{p(\mathbf{x}, \mathbf{y})} \left[ \log \frac{p(\mathbf{x}) e^{f(\mathbf{x}, \mathbf{y})}}{p(\mathbf{x}) Z(\mathbf{y})} \right] = \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} [f(\mathbf{x}, \mathbf{y})] - \mathbb{E}_{p(\mathbf{y})} [Z(\mathbf{y})] \quad (5.128)$$

4

$$= \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} [f(\mathbf{x}, \mathbf{y})] - \mathbb{E}_{p(\mathbf{y})} \left[ \log \mathbb{E}_{p(\mathbf{x})} \left[ e^{f(\mathbf{x}, \mathbf{y})} \right] \right] \quad (5.129)$$

5

$$\triangleq I_{DV}(X; Y). \quad (5.130)$$

6 This is the **Donsker Varadhan lower bound** [DV75].

7 We can construct a more tractable version of this by using the fact that the log function can be  
8 upper bounded by a straight line using

9

$$\log x \leq \frac{x}{a} + \log a - 1 \quad (5.131)$$

10 If we set  $a = e$ , we get

11

$$\mathbb{I}(X; Y) \geq \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} [f(\mathbf{x}, \mathbf{y})] - e^{-1} \mathbb{E}_{p(\mathbf{y})} Z(\mathbf{y}) \triangleq I_{NWJ}(X; Y) \quad (5.132)$$

12 This is called the **NWJ lower bound** (after the authors of Nguyen, Wainwright, and Jordan  
13 [NWJ10a]), or the f-GAN KL [NCT16a], or the MINE-f score [Bel+18].

#### 14 5.3.6.4 InfoNCE lower bound

15 If we instead explore a multi-sample extension to the DV bound above, we can generate the following  
16 lower bound (see [Poo+19] for the derivation):

17

$$\mathbb{I}_{\text{NCE}} = \mathbb{E} \left[ \frac{1}{K} \sum_{i=1}^K \log \frac{e^{f(\mathbf{x}_i, \mathbf{y}_i)}}{\frac{1}{K} \sum_{j=1}^K e^{f(\mathbf{x}_i, \mathbf{y}_j)}} \right] \quad (5.133)$$

18

$$= \log K - \mathbb{E} \left[ \frac{1}{K} \sum_{i=1}^K \log \left( 1 + \sum_{j \neq i}^K e^{f(\mathbf{x}_i, \mathbf{y}_j) - f(\mathbf{x}_i, \mathbf{y}_i)} \right) \right] \quad (5.134)$$

19 where the expectation is over paired samples from the joint  $p(X, Y)$ . The quantity in Equation (5.134)  
20 is called the **InfoNCE** estimate, and was proposed in [OLV18; Hen+19a]. (NCE stands for “noise  
21 contrastive estimation”, and is discussed in Section 25.4.)

22 The intuition here is that mutual information is a divergence between the joint  $p(\mathbf{x}, \mathbf{y})$  and the  
23 product of the marginals,  $p(\mathbf{x})p(\mathbf{y})$ . In other words, mutual information is a measurement of how  
24 distinct sampling pairs jointly is from sampling  $\mathbf{x}$ s and  $\mathbf{y}$ s independently. The InfoNCE bound  
25 provides a lower bound on the true mutual information by attempting to train a model to distinguish  
26 between these two situations.

27 Although this is a valid lower bound, we may need to use a large batch size  $K$  to estimate the  
28 MI if the MI is large, since  $\mathbb{I}_{\text{NCE}} \leq \log K$ . (Recently [SE20a] proposed to use a multi-label classifier,  
29 rather than a multi-class classifier, to overcome this limitation.)

---

## 1 2 5.4 Data compression (source coding)

3 **Data compression**, also known as **source coding**, is at the heart of information theory. It is also  
4 related to probabilistic machine learning. The reason for this is as follows: if we can model the  
5 probability of different kinds of data samples, then we can assign short **code words** to the most  
6 frequently occurring ones, reserving longer encodings for the less frequent ones. This is similar to  
7 the situation in natural language, where common words (such as “a”, “the”, “and”) are generally  
8 much shorter than rare words. Thus the ability to compress data requires an ability to discover  
9 the underlying patterns, and their relative frequencies, in the data. This has led Marcus Hutter  
10 to propose that compression be used as an objective way to measure performance towards general  
11 purpose AI. More precisely, he is offering 50,000 Euros to anyone who can compress the first 100MB  
12 of (English) Wikipedia better than some baseline. This is known as the **Hutter prize**.<sup>2</sup>

13 In this section, we give a brief summary of some of the key ideas in data compression. For details,  
14 see e.g., [Mac03; CT06; YMT22].

16

### 17 5.4.1 Lossless compression

18 Discrete data, such as natural language, can always be compressed in such a way that we can uniquely  
19 recover the original data. This is called **lossless compression**.

20 Claude Shannon proved that the expected number of bits needed to losslessly encode some data  
21 coming from distribution  $p$  is at least  $\mathbb{H}(p)$ . This is known as the **source coding theorem**. Achieving  
22 this lower bound requires coming up with good probability models, as well as good ways to design  
23 codes based on those models. Because of the non-negativity of the KL divergence,  $\mathbb{H}(p, q) \geq \mathbb{H}(p)$ , so  
24 if we use any model  $q$  other than the true model  $p$  to compress the data, it will take some excess bits.  
25 The number of excess bits is exactly  $D_{\text{KL}}(p\|q)$ .

26 Common techniques for realizing lossless codes include Huffman coding, arithmetic coding and  
27 asymmetric numeral systems [Dud13]. The input to these algorithms is a probability distribution  
28 over strings (which is where ML comes in). This distribution is often represented using a latent  
29 variable model (see e.g., [TBB19; KAH19]).

31

### 32 5.4.2 Lossy compression and the rate-distortion tradeoff

33 To encode real-valued signals, such as images and sound, as a digital signal, we first have to quantize  
34 the signal into a sequence of symbols. A simple way to do this is to use vector quantization. We can  
35 then compress this discrete sequence of symbols using lossless coding methods. However, when we  
36 uncompress, we lose some information. Hence this approach is called **lossy compression**.

37 In this section, we quantify this tradeoff between the size of the representation (number of symbols  
38 we use), and the resulting error. We will use the terminology of the variational information bottleneck  
39 discussed in Section 5.6.2 (except here we are in the unsupervised setting). In particular, we assume  
40 we have a stochastic encoder  $p(\mathbf{z}|\mathbf{x})$ , a stochastic decoder  $d(\mathbf{x}|\mathbf{z})$  and a prior marginal  $m(\mathbf{z})$ .

41 We define the **distortion** of an encoder-decoder pair (as in Section 5.6.2) as follows:

$$\frac{43}{44} D = - \int d\mathbf{x} p(\mathbf{x}) \int d\mathbf{z} e(\mathbf{z}|\mathbf{x}) \log d(\mathbf{x}|\mathbf{z}) \quad (5.135)$$

45 

---

46 2. For details, see <http://prize.hutter1.net>.

47

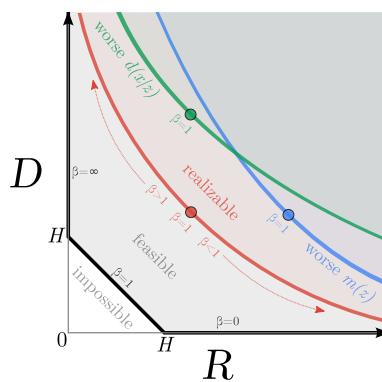


Figure 5.7: Illustration of the rate-distortion tradeoff. See text for details. From Figure 1 of [Ale+18]. Used with kind permission of Alex Alemi.

If the decoder is a deterministic model plus Gaussian noise,  $d(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|f_d(\mathbf{z}), \sigma^2)$ , and the encoder is deterministic,  $e(\mathbf{z}|\mathbf{x}) = \delta(\mathbf{z} - f_e(\mathbf{x}))$ , then this becomes

$$D = \frac{1}{\sigma^2} \mathbb{E}_{p(\mathbf{x})} [| |f_d(f_e(\mathbf{x})) - \mathbf{x}| |^2] \quad (5.136)$$

This is just the expected **reconstruction error** that occurs if we (deterministically) encode and then decode the data using  $f_e$  and  $f_d$ .

We define the **rate** of our model as follows:

$$R = \int d\mathbf{x} p(\mathbf{x}) \int d\mathbf{z} e(\mathbf{z}|\mathbf{x}) \log \frac{e(\mathbf{z}|\mathbf{x})}{m(\mathbf{z})} \quad (5.137)$$

$$= \mathbb{E}_{p(\mathbf{x})} [D_{\text{KL}}(e(\mathbf{z}|\mathbf{x}) || m(\mathbf{z}))] \quad (5.138)$$

$$= \int d\mathbf{x} \int d\mathbf{z} p(\mathbf{x}, \mathbf{z}) \log \frac{p(\mathbf{x}, \mathbf{z})}{p(\mathbf{x})m(\mathbf{z})} \geq \mathbb{I}(\mathbf{x}, \mathbf{z}) \quad (5.139)$$

This is just the average KL between our encoding distribution and the marginal. If we use  $m(\mathbf{z})$  to design an optimal code, then the rate is the *excess* number of bits we need to pay to encode our data using  $m(\mathbf{z})$  rather than the true **aggregate posterior**  $p(\mathbf{z}) = \int d\mathbf{x} p(\mathbf{x})e(\mathbf{z}|\mathbf{x})$ .

There is a fundamental tradeoff between the rate and distortion. To see why, note that a trivial encoding scheme would set  $e(\mathbf{z}|\mathbf{x}) = \delta(\mathbf{z} - \mathbf{x})$ , which simply uses  $\mathbf{x}$  as its own best representation. This would incur 0 distortion (and hence maximize the likelihood), but it would incur a high rate, since each  $e(\mathbf{z}|\mathbf{x})$  distribution would be unique, and far from  $m(\mathbf{z})$ . In other words, there would be no compression. Conversely, if  $e(\mathbf{z}|\mathbf{x}) = \delta(\mathbf{z} - \mathbf{0})$ , the encoder would ignore the input. In this case, the rate would be 0, but the distortion would be high.

We can characterize the tradeoff more precisely using the variational lower and upper bounds on the mutual information from Section 5.3.6. From that section, we know that

$$H - D \leq \mathbb{I}(\mathbf{x}; \mathbf{z}) \leq R \quad (5.140)$$

1 where  $H$  is the (differential) entropy  
2

3  
4 
$$H = - \int d\mathbf{x} p(\mathbf{x}) \log p(\mathbf{x}) \tag{5.141}$$
  
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 
$$J = D + \beta R = \int d\mathbf{x} p(\mathbf{x}) \int d\mathbf{z} e(\mathbf{z}|\mathbf{x}) \left[ -\log d(\mathbf{x}|\mathbf{z}) + \beta \log \frac{e(\mathbf{z}|\mathbf{x})}{m(\mathbf{z})} \right] \tag{5.142}$$
  
28  
29

30 If we set  $\beta = 1$ , and define  $q(\mathbf{z}|\mathbf{x}) = e(\mathbf{z}|\mathbf{x})$ ,  $p(\mathbf{x}|\mathbf{z}) = d(\mathbf{x}|\mathbf{z})$ , and  $p(\mathbf{z}) = m(\mathbf{z})$ , this exactly matches  
31 the VAE objective in Section 22.2. To see this, note that the ELBO from Section 10.1.2 can be  
32 written as

33  
34 
$$\mathcal{L} = -(D + R) = \mathbb{E}_{p(\mathbf{x})} \left[ \mathbb{E}_{e(\mathbf{z}|\mathbf{x})} [\log d(\mathbf{x}|\mathbf{z})] - \mathbb{E}_{e(\mathbf{z}|\mathbf{x})} \left[ \log \frac{e(\mathbf{z}|\mathbf{x})}{m(\mathbf{z})} \right] \right] \tag{5.143}$$
  
35  
36

37 which we recognize as the expected reconstruction error minus the KL term  $D_{\text{KL}}(e(\mathbf{z}|\mathbf{x}) \| m(\mathbf{z}))$ .

38 If we allow  $\beta \neq 1$ , we recover the  $\beta$ -VAE objective discussed in Section 22.3.2. Note, however, that  
39 the  $\beta$ -VAE model cannot distinguish between different solutions on the diagonal line, all of which have  
40  $\beta = 1$ . This is because all such models have the same marginal likelihood (and hence same ELBO),  
41 although they differ radically in terms of whether they learn an interesting latent representation or  
42 not. Thus likelihood is not a sufficient metric for comparing the quality of unsupervised representation  
43 learning methods, as discussed in Section 22.3.2.

44 For further discussion on the inherent conflict between rate, distortion and *perception*, see Blau  
45 and Michaeli [BM19]. For techniques for evaluating rate distortion curves for models see Huang, Cao,  
46 and Grosse [HCG20].

47

---

### 5.4.3 Bits back coding

In the previous section we penalized the rate of our code using the average KL divergence,  $\mathbb{E}_{p(\mathbf{x})} [R(\mathbf{x})]$ , where

$$R(\mathbf{x}) \triangleq \int dz p(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{z}|\mathbf{x})}{m(\mathbf{z})} = \mathbb{H}(p(\mathbf{z}|\mathbf{x}), m(\mathbf{z})) - \mathbb{H}(p(\mathbf{z}|\mathbf{x})). \quad (5.144)$$

The first term is the cross entropy, which is the expected number of bits we need to encode  $\mathbf{x}$ ; the second term is the entropy, which is the minimum number of bits. Thus we are penalizing the *excess* number of bits required to communicate the code to a receiver. How come we don't have to "pay for" the actual (total) number of bits we use, which is the cross entropy?

The reason is that we could in principle get the bits needed by the optimal code given back to us; this is called **bits back coding** [HC93; FH97]. The argument goes as follows. Imagine Alice is trying to (losslessly) communicate some data, such as an image  $\mathbf{x}$ , to Bob. Before they went their separate ways, both Alice and Bob decided to share their encoder  $p(\mathbf{z}|\mathbf{x})$ , marginal  $m(\mathbf{z})$  and decoder distributions  $d(\mathbf{x}|\mathbf{z})$ . To communicate an image, Alice will use a **two part code**. First, she will sample a code  $\mathbf{z} \sim p(\mathbf{z}|\mathbf{x})$  from her encoder, and communicate that to Bob over a channel designed to efficiently encode samples from the marginal  $m(\mathbf{z})$ ; this costs  $-\log_2 m(\mathbf{z})$  bits. Next Alice will use her decoder  $d(\mathbf{x}|\mathbf{z})$  to compute the residual error, and losslessly send that to Bob at the cost of  $-\log_2 d(\mathbf{x}|\mathbf{z})$  bits. The expected total number of bits required here is what we naively expected:

$$\mathbb{E}_{p(\mathbf{z}|\mathbf{x})} [-\log_2 d(\mathbf{x}|\mathbf{z}) - \log_2 m(\mathbf{z})] = D + \mathbb{H}(p(\mathbf{z}|\mathbf{x}), m(\mathbf{z})). \quad (5.145)$$

We see that this is the distortion plus cross entropy, not distortion plus rate. So how do we get the bits back, to convert the cross entropy to a rate term?

The trick is that Bob actually receives more information than we suspected. Bob can use the code  $\mathbf{z}$  and the residual error to perfectly reconstruct  $\mathbf{x}$ . However, Bob also knows what specific code Alice sent,  $\mathbf{z}$ , as well as what encoder she used,  $p(\mathbf{z}|\mathbf{x})$ . When Alice drew the sample code  $\mathbf{z} \sim p(\mathbf{z}|\mathbf{x})$ , she had to use some kind of entropy source in order to generate the random sample. Suppose she did it by picking words sequentially from a compressed copy of Moby Dick, in order to generate a stream of random bits. On Bob's end, he can reverse engineer all of the sampling bits, and thus recover the compressed copy of Moby Dick! Thus Alice can use the extra randomness in the choice of  $\mathbf{z}$  to share more information.

While in the original formulation the bits-back argument was largely theoretical, offering a thought experiment for why we should penalize our models with the KL instead of the cross entropy, recently several practical real world algorithms have been developed that actually achieve the bits-back goal. These include [HHLMF18; AT20; TBB19; YBM20; HLA19].

## 5.5 Error-correcting codes (channel coding)

The idea behind **error correcting codes** is to add redundancy to a signal  $\mathbf{x}$  (which is the result of encoding the original data), such that when it is sent over to the receiver via a noisy transmission line (such as a cell phone connection), the receiver can recover from any corruptions that might occur to the signal. This is called **channel coding**.

In more detail, let  $\mathbf{x} \in \{0, 1\}^m$  be the source message, where  $m$  is called the **block length**. Let  $\mathbf{y}$  be the result of sending  $\mathbf{x}$  over a **noisy channel**. This is a corrupted version of the message.

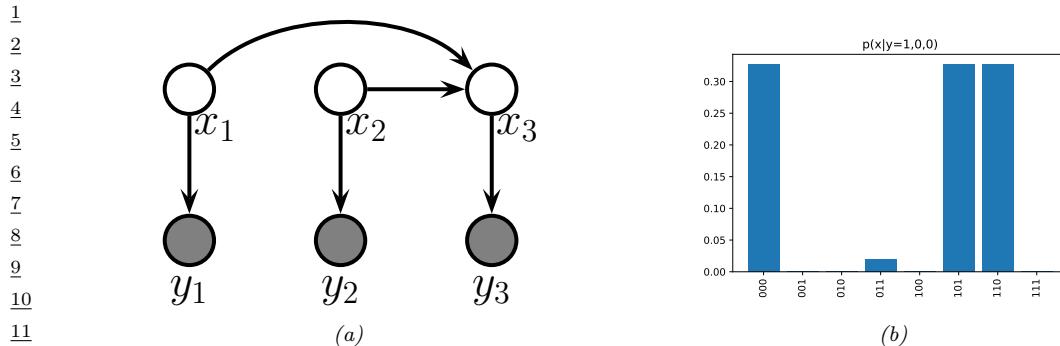


Figure 5.8: (a) A simple error-correcting code PGM-D.  $x_i$  are the sent bits,  $y_i$  are the received bits.  $x_3$  is an even parity check bit computed from  $x_1$  and  $x_2$ . (b) Posterior over codewords given that  $\mathbf{y} = (1, 0, 0)$ ; the probability of a bit flip is 0.2. Generated by [error\\_correcting\\_code\\_demo.py](#).

For example, each message bit may get flipped independently with probability  $\alpha$ , 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.)

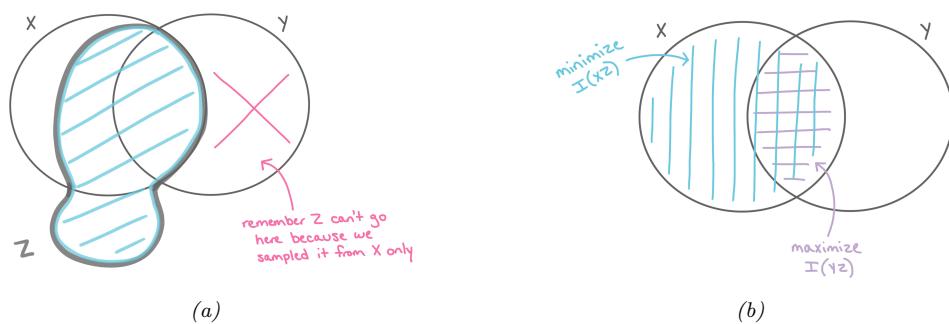
For example, suppose we have two original message bits, and we add one parity bit. This can be modeled using a directed graphical model as shown in Figure 5.8(a). This graph encodes the following joint probability distribution:

$$p(\mathbf{x}, \mathbf{y}) = p(x_1)p(x_2)p(x_3|x_1, x_2) \prod_{i=1}^3 p(y_i|x_i) \quad (5.146)$$

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  $I(Z, Y)$  and minimizes  $I(Z, X)$ . 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  $z$  is a **representation** of  $x$  if  $z$  is a (possibly stochastic) function of  $x$ , and hence can be described by the conditional  $p(z|x)$ . We say that a representation  $z$  of  $x$  is **sufficient** for task  $y$  if  $y \perp x|z$ , or equivalently, if  $\mathbb{I}(z; y) = \mathbb{I}(x; y)$ , i.e.,  $\mathbb{H}(y|z) = \mathbb{H}(y|x)$ . We say that a representation is a **minimal sufficient statistic** if  $z$  is sufficient and there is no other  $z$  with smaller  $\mathbb{I}(z; x)$  value. Thus we would like to find a representation  $z$  that maximizes  $\mathbb{I}(z; y)$  while minimizing  $\mathbb{I}(z; x)$ . 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:

$$p(\mathbf{x}, \mathbf{y}, z) = p(z|x)p(\mathbf{y}|x)p(\mathbf{x}) \quad (5.148)$$

1 Thus  $Z$  can capture any amount of information about  $X$  that it wants, but cannot contain information  
 2 that is unique to  $Y$ , as illustrated in Figure 5.9a. The optimal representation only captures information  
 3 about  $X$  that is useful for  $Y$ ; to prevent us “wasting capacity” and fitting irrelevant details of the  
 4 input,  $Z$  should also minimize information about  $X$ , as shown in Figure 5.9b.  
 5

6 If all the random variables are discrete, and  $\mathbf{z} = e(\mathbf{x})$  is a deterministic function of  $\mathbf{x}$ , then the  
 7 algorithm of [TPB99] can be used to minimize the IB objective in Section 5.6. The objective can  
 8 also be solved analytically if all variables are jointly Gaussian [Che+05] (the resulting method can be  
 9 viewed as a form of supervised PCA). But in general, it is intractable to solve this problem exactly.  
 10 We discuss a tractable approximation in Section 5.6.2. (More details can be found in e.g., [SZ22].)  
 11

## 12 5.6.2 Variational IB

13 In this section, we derive a variational upper bound on Equation (5.147), leveraging ideas from  
 14 Section 5.3.6. This is called the **variational IB** or **VIB** method [Ale+16]. The key trick will be to  
 15 use the non-negativity of the KL divergence to write  
 16

$$17 \quad \int d\mathbf{x} p(\mathbf{x}) \log p(\mathbf{x}) \leq \int d\mathbf{x} p(\mathbf{x}) \log q(\mathbf{x}) \quad (5.149)$$

18 for any distribution  $q$ . (Note that both  $p$  and  $q$  may be conditioned on other variables.)  
 19

20 To explain the method in more detail, let us define the following notation. Let  $e(\mathbf{z}|\mathbf{x}) = p(\mathbf{z}|\mathbf{x})$   
 21 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  
 22 the classifier (decoder), and  $m(\mathbf{z}) \approx p(\mathbf{z})$  represent the marginal. (Note that we get to choose  
 23  $p(\mathbf{z}|\mathbf{x})$ , but the other distributions are derived by approximations of the corresponding marginals  
 24 and conditionals of the exact joint  $p(\mathbf{x}, \mathbf{y}, \mathbf{z})$ .) Also, let  $\langle \cdot \rangle$  represent expectations wrt the relevant  
 25 terms from the  $p(\mathbf{x}, \mathbf{y}, \mathbf{z})$  joint.  
 26

27 With this notation, we can derive a lower bound on  $\mathbb{I}(\mathbf{z}; \mathbf{y})$  as follows:  
 28

$$29 \quad \mathbb{I}(\mathbf{z}; \mathbf{y}) = \int dy dz p(\mathbf{y}, \mathbf{z}) \log \frac{p(\mathbf{y}, \mathbf{z})}{p(\mathbf{y})p(\mathbf{z})} \quad (5.150)$$

$$30 \quad = \int dy dz p(\mathbf{y}, \mathbf{z}) \log p(\mathbf{y}|\mathbf{z}) - \int dy dz p(\mathbf{y}, \mathbf{z}) \log p(\mathbf{y}) \quad (5.151)$$

$$31 \quad = \int dy dz p(\mathbf{z})p(\mathbf{y}|\mathbf{z}) \log p(\mathbf{y}|\mathbf{z}) - \text{const} \quad (5.152)$$

$$32 \quad \geq \int dy dz p(\mathbf{y}, \mathbf{z}) \log d(\mathbf{y}|\mathbf{z}) \quad (5.153)$$

$$33 \quad = \langle \log d(\mathbf{y}|\mathbf{z}) \rangle \quad (5.154)$$

34 where we exploited the fact that  $\mathbb{H}(p(\mathbf{y}))$  is a constant that is independent of our representation.  
 35 Note that we can approximate the expectations by sampling from  
 36

$$37 \quad p(\mathbf{y}, \mathbf{z}) = \int dx p(\mathbf{x})p(\mathbf{y}|\mathbf{x})p(\mathbf{z}|\mathbf{x}) = \int dx p(\mathbf{x}, \mathbf{y})e(\mathbf{z}|\mathbf{x}) \quad (5.155)$$

38 This is just the empirical distribution “pushed through” the encoder.  
 39

Similarly, we can derive an upper bound on  $\mathbb{I}(\mathbf{z}; \mathbf{x})$  as follows:

$$\mathbb{I}(\mathbf{z}; \mathbf{x}) = \int d\mathbf{z} d\mathbf{x} p(\mathbf{x}, \mathbf{z}) \log \frac{p(\mathbf{z}, \mathbf{x})}{p(\mathbf{x})p(\mathbf{z})} \quad (5.156)$$

$$= \int d\mathbf{z} d\mathbf{x} p(\mathbf{x}, \mathbf{z}) \log p(\mathbf{z}|\mathbf{x}) - \int d\mathbf{z} p(\mathbf{z}) \log p(\mathbf{z}) \quad (5.157)$$

$$\leq \int d\mathbf{z} d\mathbf{x} p(\mathbf{x}, \mathbf{z}) \log p(\mathbf{z}|\mathbf{x}) - \int d\mathbf{z} p(\mathbf{z}) \log m(\mathbf{z}) \quad (5.158)$$

$$= \int d\mathbf{z} d\mathbf{x} p(\mathbf{x}, \mathbf{z}) \log \frac{e(\mathbf{z}|\mathbf{x})}{m(\mathbf{z})} \quad (5.159)$$

$$= \langle \log e(\mathbf{z}|\mathbf{x}) \rangle - \langle m(\mathbf{z}) \rangle \quad (5.160)$$

Note that we can approximate the expectations by sampling from  $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x})p(\mathbf{z}|\mathbf{x})$ .

Putting it altogether, we get the following upper bound on the IB objective:

$$\beta \mathbb{I}(\mathbf{x}; \mathbf{z}) - \mathbb{I}(\mathbf{z}; \mathbf{y}) \leq \beta (\langle \log e(\mathbf{z}|\mathbf{x}) \rangle - \langle \log m(\mathbf{z}) \rangle) - \langle \log d(\mathbf{y}|\mathbf{z}) \rangle \quad (5.161)$$

Thus the VIB objective is

$$\mathcal{L}_{\text{VIB}} = \beta (\mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})e(\mathbf{z}|\mathbf{x})} [\log e(\mathbf{z}|\mathbf{x}) - \log m(\mathbf{z})]) - \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})e(\mathbf{z}|\mathbf{x})d(\mathbf{y}|\mathbf{z})} [\log d(\mathbf{y}|\mathbf{z})] \quad (5.162)$$

$$= -\mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})e(\mathbf{z}|\mathbf{x})d(\mathbf{y}|\mathbf{z})} [\log d(\mathbf{y}|\mathbf{z})] + \beta \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [D_{\text{KL}}(e(\mathbf{z}|\mathbf{x}) \| m(\mathbf{z}))] \quad (5.163)$$

We can now take stochastic gradients of this objective and minimize it (wrt the parameters of the encoder, decoder and marginal) using SGD. (We assume the distributions are reparameterizable, as discussed in Section 6.6.4.) For the encoder  $e(\mathbf{z}|\mathbf{x})$ , we often use a conditional Gaussian, and for the decoder  $d(\mathbf{y}|\mathbf{z})$ , we often use a softmax classifier. For the marginal,  $m(\mathbf{z})$ , we should use a flexible model, such as a mixture of Gaussians, since it needs to approximate the **aggregated posterior**  $p(\mathbf{z}) = \int d\mathbf{z} p(\mathbf{x})e(\mathbf{z}|\mathbf{x})$ , which is a mixture of  $N$  Gaussians (assuming  $p(\mathbf{x})$  is an empirical distribution with  $N$  samples, and  $e(\mathbf{z}|\mathbf{x})$  is a Gaussian).

We illustrate this in Figure 5.10, where we fit the an MLP model to MNIST. We use a 2d bottleneck layer before passing to the softmax. On the left we show the embedding learned by a deterministic encoder. We see that each image gets mapped to a point, and there is little overlap between classes, or between instances. On the right we show the embedding learned by a stochastic encoder. Now each image gets mapped to a Gaussian distribution. The classes are still well separated, but individual instances of a class are no longer distinguishable, since such information is not relevant for prediction purposes.

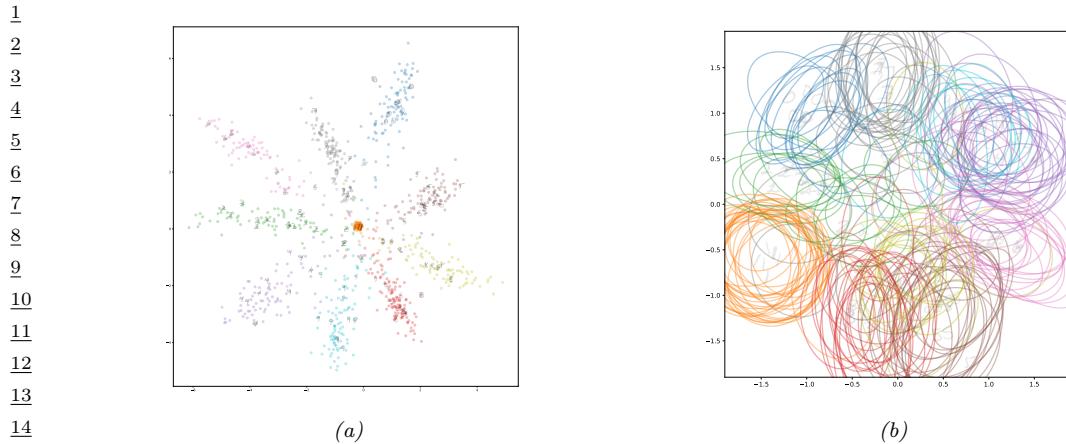
### 5.6.3 Conditional entropy bottleneck

The IB tries to maximize  $\mathbb{I}(Z; Y)$  while minimizing  $\mathbb{I}(Z; X)$ . We can write this objective as

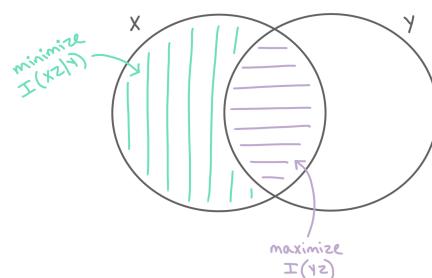
$$\min \mathbb{I}(\mathbf{x}; \mathbf{z}) - \lambda \mathbb{I}(\mathbf{y}; \mathbf{z}) \quad (5.164)$$

for  $\lambda \geq 0$ . However, we see from the information diagram in Figure 5.9b that  $\mathbb{I}(Z; X)$  contains some information that is relevant to  $Y$ . A sensible alternative objective is to minimizes the residual mutual information,  $\mathbb{I}(X; Z|Y)$ . This gives rise to the following objective:

$$\min \mathbb{I}(\mathbf{x}; \mathbf{z}|\mathbf{y}) - \lambda' \mathbb{I}(\mathbf{y}; \mathbf{z}) \quad (5.165)$$



**Figure 5.10:** 2d embeddings of MNIST digits created by an MLP classifier. (a) Deterministic model. (b) Stochastic VIB model. Generated by [VIBDemo2021.ipynb](#). Used with kind permission of Alex Alemi.



<sup>30</sup> Figure 5.11: Conditional entropy bottleneck (CEB) chooses a representation  $Z$  that maximizes  $\mathbb{I}(Z, Y)$  and  
<sup>31</sup> minimizes  $\mathbb{I}(X, Z|Y)$ . Used with kind permission of Katie Everett.

<sup>43</sup> for  $\lambda' \geq 0$ . This is known as the **conditional entropy bottleneck** or **CEB** [Fis20]. See Figure 5.11  
<sup>44</sup> for an illustration.

45 Since  $\mathbb{I}(\mathbf{x}; \mathbf{z}|\mathbf{y}) = \mathbb{I}(\mathbf{x}; \mathbf{z}) - \mathbb{I}(\mathbf{y}; \mathbf{z})$ , we see that the CEB is equivalent to standard IB with  $\lambda' = \lambda + 1$ .  
46 However, it is easier to upper bound  $\mathbb{I}(\mathbf{x}; \mathbf{z}|\mathbf{y})$  than  $\mathbb{I}(\mathbf{x}; \mathbf{z})$ , since we are conditioning on  $\mathbf{y}$ , which  
47

1 provides information about  $\mathbf{z}$ . In particular, we have  
2

$$\mathbb{I}(\mathbf{x}; \mathbf{z}|\mathbf{y}) = \mathbb{I}(\mathbf{x}; \mathbf{z}) - \mathbb{I}(\mathbf{y}; \mathbf{z}) \quad (5.166)$$

$$= \mathbb{H}(\mathbf{z}) - \mathbb{H}(\mathbf{z}|\mathbf{x}) - [\mathbb{H}(\mathbf{z}) - \mathbb{H}(\mathbf{z}|\mathbf{y})] \quad (5.167)$$

$$= -\mathbb{H}(\mathbf{z}|\mathbf{x}) - \mathbb{H}(\mathbf{z}|\mathbf{y}) \quad (5.168)$$

$$= \int d\mathbf{z}d\mathbf{x} p(\mathbf{x}, \mathbf{z}) \log p(\mathbf{z}|\mathbf{x}) - \int d\mathbf{z}d\mathbf{y} p(\mathbf{z}, \mathbf{y}) \log p(\mathbf{z}|\mathbf{y}) \quad (5.169)$$

$$\leq \int d\mathbf{z}d\mathbf{x} p(\mathbf{x}, \mathbf{z}) \log e(\mathbf{z}|\mathbf{x}) - \int d\mathbf{z}d\mathbf{y} p(\mathbf{z}, \mathbf{y}) \log b(\mathbf{z}|\mathbf{y}) \quad (5.170)$$

$$= \langle \log e(\mathbf{z}|\mathbf{x}) \rangle - \langle \log b(\mathbf{z}|\mathbf{y}) \rangle \quad (5.171)$$

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  $\beta$  corresponds to an optimal representation. By  
20 contrast, it is not clear how to measure distance from optimality when using IB.  
21

2223242526272829303132333435363738394041424344454647



# 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  $\Theta$  is the parameter space we are optimizing over. However, this abstraction hides many details, such as whether the problem is constrained or unconstrained, discrete or continuous, convex or non-convex, etc. In the prequel to this book, [Mur22], we discussed some simple optimization algorithms for some common problems that arise in machine learning. In this chapter, we discuss some more advanced methods. For more details on optimization, please consult some of the many excellent textbooks, such as [KW19b; BV04; NW06; Ber15; Ber16] as well as various review articles, such as [BCN18; Sun+19b; PPS18; Pey20].

## 6.2 Automatic differentiation

*This section was written by Roy Frostig.*

This section is concerned with computing (partial) derivatives of complicated functions in an automatic manner. By “complicated” we mean those expressed as a composition of an arbitrary number of more basic operations, such as in deep neural networks. This task is known as **automatic differentiation (AD)**, or **autodiff**. AD is an essential component in optimization and deep learning, and is also used in several other fields across science and engineering. See e.g. Baydin et al. [Bay+15] for a review focused on machine learning and Griewank and Walther [GW08] for a classical textbook.

### 6.2.1 Differentiation in functional form

Before covering automatic differentiation, it is useful to review the mathematics of differentiation. We will use a particular **functional** notation for partial derivatives, rather than the typical one used throughout much of this book. We will refer to the latter as the **named variable** notation for the moment. Named variable notation relies on associating function arguments with names. For instance, given a function  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ , the partial derivative of  $f$  with respect to its first scalar argument, at a

1 point  $\mathbf{a} = (a_1, a_2)$ , might be written:  
2

$$\frac{\partial f}{\partial x_1} \Big|_{\mathbf{x}=\mathbf{a}} \tag{6.2}$$

3  
4 This notation is not entirely self-contained. It refers to a name  $\mathbf{x} = (x_1, x_2)$ , implicit or inferred from  
5 context, suggesting the argument of  $f$ . An alternative expression is:  
6

$$\frac{\partial}{\partial a_1} f(a_1, a_2) \tag{6.3}$$

7  
8 where now  $a_1$  serves both as an argument name (or a symbol in an expression) and as a particular  
9 evaluation point. Tracking names can become an increasingly complicated endeavor as we compose  
10 many functions together, each possibly taking several arguments.  
11

12 A functional notation instead defines derivatives as operators on functions. If a function has  
13 multiple arguments, they are identified by position rather than by name, alleviating the need for  
14 auxiliary variable definitions. Some of the following definitions draw on those in Spivak's *Calculus*  
15 on Manifolds [Spi71], in Sussman and Wisdom's *Functional Differential Geometry* [SW13], and  
16 generally appear more regularly in accounts of differential calculus and geometry. These texts are  
17 recommended for a more formal treatment, and a more mathematically general view, of the material  
18 briefly covered in this section.  
19

20 Beside notation, we will rely on some basic multivariable calculus concepts. This includes the  
21 notion of (partial) derivatives, the differential or Jacobian of a function at a point, its role as a linear  
22 approximation local to the point, and various properties of linear maps, matrices, and transposition.  
23 We will focus on a finite-dimensional setting and write  $\{\mathbf{e}_1, \dots, \mathbf{e}_n\}$  for the standard basis in  $\mathbb{R}^n$ .  
24

25  
26 **Linear and multilinear functions.** We use  $F : \mathbb{R}^n \multimap \mathbb{R}^m$  to denote a function  $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$   
27 that is linear, and by  $F[\mathbf{x}]$  its application to  $\mathbf{x} \in \mathbb{R}^n$ . Recall that such a linear map corresponds  
28 to a matrix in  $\mathbb{R}^{m \times n}$  whose columns are  $F[\mathbf{e}_1], \dots, F[\mathbf{e}_n]$ ; both interpretations will prove useful.  
29 Conveniently, function composition and matrix multiplication expressions look similar: to compose  
30 two linear maps  $F$  and  $G$  we can write  $F \circ G$  or, barely abusing notation, consider the matrix  $FG$ .  
31 Every linear map  $F : \mathbb{R}^n \multimap \mathbb{R}^m$  has a transpose  $F : \mathbb{R}^m \multimap \mathbb{R}^n$ , which is another linear map identified  
32 with transposing the corresponding matrix.  
33

34 Repeatedly using the linear arrow symbol, we can denote by:  
35

$$T : \underbrace{\mathbb{R}^n \multimap \cdots \multimap \mathbb{R}^n}_{k \text{ times}} \multimap \mathbb{R}^m \tag{6.4}$$

36  
37 a multilinear, or more specifically  $k$ -linear, map:  
38

$$T : \underbrace{\mathbb{R}^n \times \cdots \times \mathbb{R}^n}_{k \text{ times}} \rightarrow \mathbb{R}^m \tag{6.5}$$

39  
40 which corresponds to an array (or tensor) in  $\mathbb{R}^{m \times n \times \cdots \times n}$ . We denote by  $T[\mathbf{x}_1, \dots, \mathbf{x}_k] \in \mathbb{R}^m$  the  
41 application of such a  $k$ -linear map to vectors  $\mathbf{x}_1, \dots, \mathbf{x}_k \in \mathbb{R}^n$ .  
42

43  
44

**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  $\partial$  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  $\nabla$  is sometimes described as an operator as well, such that  $\nabla f$  is a function. When  $n = m = 1$ , the Jacobian is scalar-valued, and  $\partial f$  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  $\partial f$  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  $v = 0$ .<sup>1</sup>

**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.

<sup>1</sup> 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  $\partial g$  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  $\partial g_1$  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.<sup>2</sup>

10 Deciding what functions  $f$  to handle as input, and how to represent them, is perhaps the most  
11 load-bearing aspect of this setup. Over what *language* of functions should we operate? By a  
12 language, we mean some formal way of describing functions by composing a set of basic primitive  
13 operations. For primitives, we can think of various differentiable array operations (elementwise  
14 arithmetic, reductions, contractions, indexing and slicing, concatenation, etc.), but we will largely  
15 consider primitives and their derivatives as a given, and focus on how elaborately we can compose  
16 them. AD becomes increasingly challenging with increasingly expressive languages. Considering this,  
17 we introduce it in stages.

18

19 **6.2.2.1 Chain compositions and the chain rule**

20 To start, take only functions that are **chain compositions** of basic operations. Chains are a  
21 convenient class of function representations because derivatives *decompose* along the same structure  
22 according to the aptly-named chain rule.  
23

24 As a toy example, consider  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  composed of three operations in sequence:

$$\underline{25} \quad f = c \circ b \circ a \quad (6.28) \quad \underline{26}$$

27 By the chain rule, its derivatives are given by

$$\underline{28} \quad \partial f(\mathbf{x}) = \partial c(b(a(\mathbf{x}))) \circ \partial b(a(\mathbf{x})) \circ \partial a(\mathbf{x}) \quad (6.29) \quad \underline{29}$$

30 Now consider the JVP against an input perturbation  $\mathbf{v} \in \mathbb{R}^n$ :

$$\underline{31} \quad \partial f(\mathbf{x})[\mathbf{v}] = \partial c(b(a(\mathbf{x}))) [\partial b(a(\mathbf{x})) [\partial a(\mathbf{x})[\mathbf{v}]]] \quad (6.30) \quad \underline{32}$$

33 This expression's bracketing highlights a right-to-left evaluation order that corresponds to **forward-**  
34 **mode automatic differentiation**. Namely, to carry out this JVP, it makes sense to compute  
35 prefixes of the original chain:  
36

$$\underline{37} \quad \mathbf{x}, a(\mathbf{x}), b(a(\mathbf{x})) \quad (6.31) \quad \underline{38}$$

39 alongside the partial JVPs, because each is then immediately used as a subsequent linearization  
40 point, respectively:

$$\underline{41} \quad \partial a(\mathbf{x}), \partial b(a(\mathbf{x})), \partial c(b(a(\mathbf{x}))) \quad (6.32) \quad \underline{42}$$

43 Extending this idea to arbitrary chain compositions gives Algorithm 1.

44 

---

45 2. Materializing the Jacobian as a numerical array, as is commonly required in an optimization context, is a special  
46 case of computing a JVP or VJP against the standard basis vectors in  $\mathbb{R}^n$  or  $\mathbb{R}^m$  respectively.

47

---

**Algorithm 1:** Forward-mode automatic differentiation (JVP) on chains

---

```

1 input:  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  as a chain composition  $f = f_T \circ \dots \circ f_1$ 
2 input: linearization point  $\mathbf{x} \in \mathbb{R}^n$  and input perturbation  $\mathbf{v} \in \mathbb{R}^n$ 
3  $\mathbf{x}_0, \mathbf{v}_0 := \mathbf{x}, \mathbf{v}$ 
4 for  $t := 1, \dots, T$  do
5    $\mathbf{x}_t := f_t(\mathbf{x}_{t-1})$ 
6    $\mathbf{v}_t := \partial f_t(\mathbf{x}_{t-1})[\mathbf{v}_{t-1}]$ 
7 output:  $\mathbf{x}_T$ , equal to  $f(\mathbf{x})$ 
8 output:  $\mathbf{v}_T$ , equal to  $\partial f(\mathbf{x})[\mathbf{v}]$ 

```

---

By contrast, we can transpose Equation (6.29) to consider a VJP against an output perturbation  $\mathbf{u} \in \mathbb{R}^m$ :

$$\partial f(\mathbf{x})^\top[\mathbf{u}] = \partial a(\mathbf{x})^\top [\partial b(a(\mathbf{x}))^\top [\partial c(b(a(\mathbf{x})))^\top[\mathbf{u}]]] \quad (6.33)$$

Transposition reverses the Jacobian maps relative to their order in Equation (6.29), and now the bracketed evaluation corresponds to **reverse-mode automatic differentiation**. To carry out this VJP, we can compute the original chain prefixes  $\mathbf{x}$ ,  $a(\mathbf{x})$ , and  $b(a(\mathbf{x}))$  first, and then read them *in reverse* as successive linearization points:

$$\partial c(b(a(\mathbf{x})))^\top, \partial b(a(\mathbf{x}))^\top, \partial a(\mathbf{x})^\top \quad (6.34)$$

Extending this idea to arbitrary chain compositions gives Algorithm 2.

---

**Algorithm 2:** Reverse-mode automatic differentiation (VJP) on chains

---

```

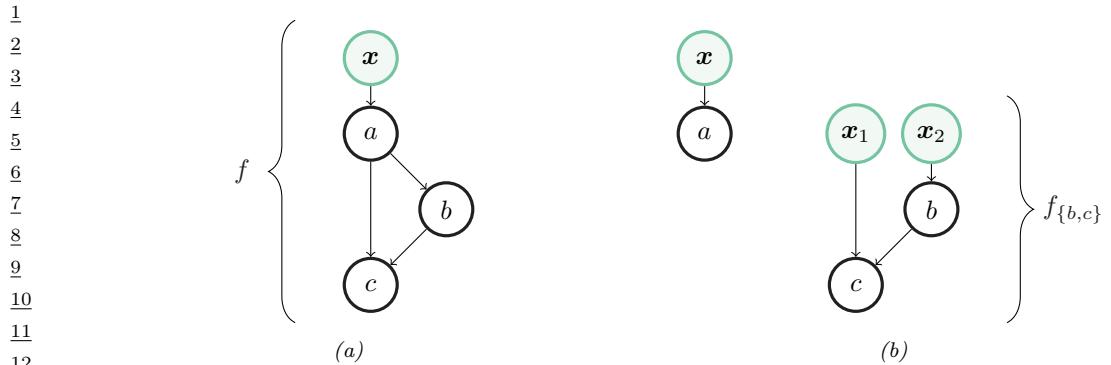
1 input:  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  as a chain composition  $f = f_T \circ \dots \circ f_1$ 
2 input: linearization point  $\mathbf{x} \in \mathbb{R}^n$  and output perturbation  $\mathbf{u} \in \mathbb{R}^n$ 
3  $\mathbf{x}_0 := \mathbf{x}$ 
4 for  $t := 1, \dots, T$  do
5    $\mathbf{x}_t := f_t(\mathbf{x}_{t-1})$ 
6    $\mathbf{u}_{T+1} := \mathbf{u}$ 
7   for  $t := T, \dots, 1$  do
8      $\mathbf{u}_t := \partial f_t(\mathbf{x}_{t-1})^\top[\mathbf{u}_{t+1}]$ 
9   output:  $\mathbf{x}_T$ , equal to  $f(\mathbf{x})$ 
10  output:  $\mathbf{u}_1$ , equal to  $\partial f(\mathbf{x})^\top[\mathbf{u}]$ 

```

---

Although chain compositions impose a very specific structure, they already capture some deep neural network models, such as multi-layer perceptrons (provided matrix multiplication is a primitive operation), as covered in this book's prequel [Mur22, Ch.13].

Reverse-mode AD is faster than forward-mode when the output is scalar valued (as often arises in deep learning, where the output is a loss function). However, reverse-mode AD stores all chain



13 *Figure 6.1: A circuit for a function  $f$  over three primitives, and its decomposition into two circuits without  
14 fan-out. Input nodes are drawn in green.*

15

16

17 prefixes before its backward traversal, so it consumes more memory than forward-mode. There  
18 are ways to combat this memory requirement in special-case scenarios, such as when the chained  
19 operations are each reversible [MDA15; Gom+17; KKL20]. One can also trade off memory for  
20 computation by discarding some prefixes and re-computing them as needed.  
21

### 22 6.2.2.2 From chains to circuits 23

24 When primitives can accept multiple inputs, we can naturally extend chains to **circuits**—directed  
25 acyclic graphs over primitive operations, sometimes also called computation graphs. To set up for  
26 this section, we will distinguish between (i) **input nodes** of a circuit, which symbolize a function’s  
27 arguments, and (ii) **primitive nodes**, each of which is labeled by a primitive operation. We assume  
28 that input nodes have no incoming edges and (without loss of generality) exactly one outgoing edge  
29 each, and that the graph has exactly one sink node. The overall function of the circuit is composition  
30 of operations from the input nodes to the sink, where the output of each operation is input to others  
31 according to its outgoing edges.  
32

What made AD work in Section 6.2.2.1 is the fact that derivatives decompose along chains thanks  
to the aptly-named chain rule. When moving from chains to directed acyclic graphs, do we need  
some sort of “graph rule” in order to decompose our calculation along the circuit’s structure? Circuits  
introduce two new features: **fan-in** and **fan-out**. In graphical terms, fan-in simply refers to multiple  
edges incoming to a node, and fan-out refers to multiple edges outgoing.  
33

What do these mean in functional terms? Fan-in happens when a primitive operation accepts  
multiple arguments. We observed in Section 6.2.1 that multiple arguments can be treated as one, and  
how the chain rule then applies. Fan-out requires slightly more care, specifically for reverse-mode  
differentiation.  
34

The gist of an answer can be illustrated with a small example. Consider the circuit in Figure 6.1a.  
The operation  $a$  precedes  $b$  and  $c$  topologically, with an outgoing edge to each of both. We can cut  $a$   
away from  $\{b, c\}$  to produce two new circuits, shown in Figure 6.1b. The first corresponds to  $a$  and  
the second corresponds to the remaining computation, given by:  
35

36

$$f_{\{b,c\}}(\mathbf{x}_1, \mathbf{x}_2) = c(\mathbf{x}_1, b(\mathbf{x}_2)). \quad (6.35)$$

37

We can recover the complete function  $f$  from  $a$  and  $f_{\{b,c\}}$  with the help of a function  $\text{dup}$  given by:

$$\text{dup}(\mathbf{x}) = (\mathbf{x}, \mathbf{x}) \equiv \begin{pmatrix} I \\ I \end{pmatrix} \mathbf{x} \quad (6.36)$$

so that  $f$  can be written as a chain composition:

$$f = f_{\{b,c\}} \circ \text{dup} \circ a. \quad (6.37)$$

The circuit for  $f_{\{b,c\}}$  contains no fan-out, and composition rules such as Equation (6.25) tell us its derivatives in terms of  $b$ ,  $c$ , and their derivatives, all via the chain rule. Meanwhile, the chain rule applied to Equation (6.37) says that:

$$\partial f(\mathbf{x}) = \partial f_{\{b,c\}}(\text{dup}(a(\mathbf{x}))) \circ \partial \text{dup}(a(\mathbf{x})) \circ \partial a(\mathbf{x}) \quad (6.38)$$

$$= \partial f_{\{b,c\}}(a(\mathbf{x}), a(\mathbf{x})) \circ \begin{pmatrix} I \\ I \end{pmatrix} \circ \partial a(\mathbf{x}). \quad (6.39)$$

The above expression suggests calculating a JVP of  $f$  by right-to-left evaluation. It is similar to the JVP calculation suggested by Equation (6.30), but with a *duplication* operation  $(I \ I)^\top$  in the middle that arises from the Jacobian of  $\text{dup}$ .

Transposing the derivative of  $f$  at  $\mathbf{x}$ :

$$\partial f(\mathbf{x})^\top = \partial a(\mathbf{x})^\top \circ (I \ I) \circ \partial f_{\{b,c\}}(a(\mathbf{x}), a(\mathbf{x}))^\top. \quad (6.40)$$

Considering right-to-left evaluation, this too is similar to the VJP calculation suggested by Equation (6.33), but with a *summation* operation  $(I \ I)$  in the middle that arises from the *transposed* Jacobian of  $\text{dup}$ . The lesson of using  $\text{dup}$  in this small example is that, more generally, in order to handle fan-out in reverse mode AD, we can process operations in topological order—first forward and then in reverse—and then *sum* partial VJPs along multiple outgoing edges.

### Algorithm 3: Foward-mode circuit differentiation (JVP)

```

1 input:  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  composing  $f_1, \dots, f_T$  in topological order, where  $f_1$  is identity
2 input: linearization point  $\mathbf{x} \in \mathbb{R}^n$  and perturbation  $\mathbf{v} \in \mathbb{R}^n$ 
3  $\mathbf{x}_1, \mathbf{v}_1 := \mathbf{x}, \mathbf{v}$ 
4 for  $t := 2, \dots, T$  do
5   let  $[q_1, \dots, q_r] = \text{parents}(t)$ 
6    $\mathbf{x}_t := f_t(\mathbf{x}_{q_1}, \dots, \mathbf{x}_{q_r})$ 
7    $\mathbf{v}_t := \sum_{i=1}^r \partial_i f_t(\mathbf{x}_{q_1}, \dots, \mathbf{x}_{q_r})[\mathbf{v}_{q_i}]$ 
8 output:  $\mathbf{x}_T$ , equal to  $f(\mathbf{x})$ 
9 output:  $\mathbf{v}_T$ , equal to  $\partial f(\mathbf{x})[\mathbf{v}]$ 
```

Algorithms 3 and 4 give a complete description of forward- and reverse-mode differentiation on circuits. For brevity they assume a single argument to the entire circuit function. Nodes are indexed  $1, \dots, T$ . The first is the input node associated and the remaining  $T - 1$  are labeled by their operation  $f_2, \dots, f_T$ . We take  $f_1$  to be the identity. For each  $t$ , if  $f_t$  takes  $k$  arguments, let  $\text{parents}(t)$  be the

---

1  
2   **Algorithm 4:** Reverse-mode circuit differentiation (VJP)  
3   **1 input:**  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  composing  $f_1, \dots, f_T$  in topological order, where  $f_1, f_T$  are identity  
4   **2 input:** linearization point  $\mathbf{x} \in \mathbb{R}^n$  and perturbation  $\mathbf{u} \in \mathbb{R}^m$   
5   **3**  $\mathbf{x}_1 := \mathbf{x}$   
6   **4 for**  $t := 2, \dots, T$  **do**  
7    **5**   let  $[q_1, \dots, q_r] = \text{parents}(t)$   
8    **6**    $\mathbf{x}_t := f_t(\mathbf{x}_{q_1}, \dots, \mathbf{x}_{q_r})$   
9  
10   **7**  $\mathbf{u}_{(T-1) \rightarrow T} := \mathbf{u}$   
11   **8 for**  $t := T - 1, \dots, 2$  **do**  
12    **9**   let  $[q_1, \dots, q_r] = \text{parents}(t)$   
13    **10**    $\mathbf{u}'_t := \sum_{c \in \text{children}(t)} \mathbf{u}_{t \rightarrow c}$   
14    **11**    $\mathbf{u}_{q_i \rightarrow t} := \partial_i f_t(\mathbf{x}_{q_1}, \dots, \mathbf{x}_{q_r})^\top \mathbf{u}'_t$  for  $i = 1, \dots, r$   
15  
16   **12 output:**  $\mathbf{x}_T$ , equal to  $f(\mathbf{x})$   
17   **13 output:**  $\mathbf{u}_{1 \rightarrow 2}$ , equal to  $\partial f(\mathbf{x})^\top \mathbf{u}$

---

18

19

20 ordered list of  $k$  indices of its parent nodes (possibly containing duplicates, due to fan-out), and let  
21 children( $t$ ) be the indices of its children (again possibly duplicate). Algorithm 4 takes a few more  
22 conventions: that  $f_T$  is the identity, that node  $T$  has  $T - 1$  as its only parent, and that the child of  
23 node 1 is node 2.

24 Fan-out is a feature of *graphs*, but arguably not an essential feature of *functions*. One can always  
25 remove all fan-out from a circuit representation by duplicating nodes. Our interest in fan-out is  
26 precisely to avoid this, allowing for an efficient representation and, in turn, efficient memory use in  
27 Algorithms 3 and 4.

28 Reverse-mode AD on circuits has appeared under various names and formulations over the years.  
29 The algorithm is precisely the **backpropagation** algorithm in neural networks, a term introduced  
30 in the 1980s [RHW86b; RHW86a], and has separately come up in the context of control theory  
31 and sensitivity, as summarized in historical notes by Goodfellow, Bengio, and Courville [GBC16,  
32 Section 6.6].

33

#### 34 6.2.2.3 From circuits to programs

35 Graphs are useful for introducing AD algorithms, and they might align well enough with neural  
36 network applications. But computer scientists have spent decades formalizing and studying various  
37 “languages for expressing functions compositionally.” Simply put, this is what programming languages  
38 are for! Can we automatically differentiate numerical functions expressed in, say, Python, Haskell,  
39 or some variant of the lambda calculus? These offer a far more widespread—and intuitively more  
40 expressive—way to describe an input function.<sup>3</sup>

41 In the previous sections, our approach to AD became more complex as we allowed for more  
42 complex graph structure. Something similar happens when we introduce grammatical constructs in a

44 3. In Python, what the language calls a “function” does not always describe a pure function of the arguments listed  
45 in its syntactic definition; its behavior may rely on side effects or global state, as allowed by the language. Here, we  
46 specifically mean a Python function that is pure and functional. JAX’s documentation details this restriction [Bra+18].

47

programming language. How do we adapt AD to handle a language with loops, conditionals, and recursive calls? What about parallel programming constructs? We have partial answers to questions like these today, although they invite a deeper dive into language details such as type systems and implementation concerns [Yu+18; Inn20; Pas+21b].

One example language construct that we already know how to handle, due to Section 6.2.2.2, is a standard `let` expression. In languages with a means of name or variable binding, multiple appearances of the same variable are analogous to fan-out in a circuit. Figure 6.1a corresponds to a function  $f$  that we could write in a functional language as:

```

f(x) =
  let ax = a(x)
  in c(ax, b(ax))

```

in which `ax` indeed appears twice after it is bound.

Understanding the interaction between language capacity and automatic differentiability is an ongoing topic of computer science research [PS08a; AP19; Vyt+19; BMP19; MP21]. In the meantime, functional languages have proven quite effective in recent AD systems, both widely-used and experimental. Systems such as JAX, Dex, and others are designed around pure functional programming models, and internally rely on functional program representations for differentiation [Mac+15; BPS16; Sha+19; FJL18; Bra+18; Mac+19; Dex; Fro+21; Pas+21a].

## 6.3 Stochastic gradient descent

In this section, we consider optimizers for unconstrained differentiable objectives. We consider gradient-based solvers which perform iterative updates of the following form

$$\theta_{t+1} = \theta_t - \eta_t \mathbf{C}_t g_t \quad (6.41)$$

where  $\mathbf{g}_t = \nabla \mathcal{L}(\theta_t)$  is the gradient of the loss, and  $\mathbf{C}_t$  is an optional **conditioning matrix**.

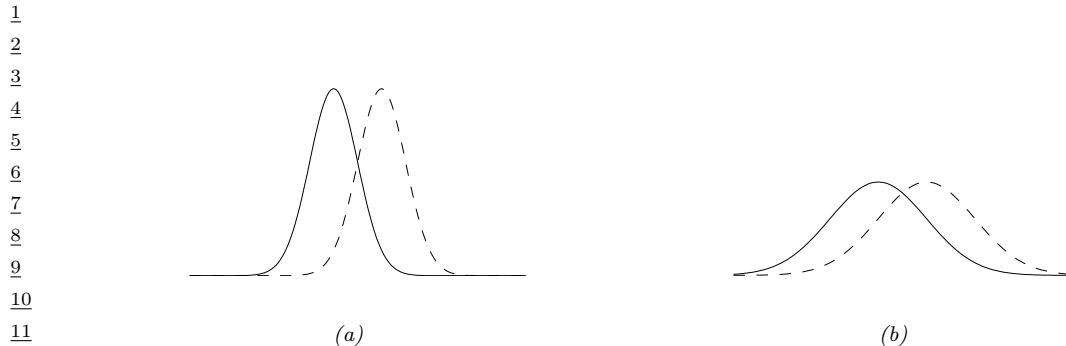
If we set  $\mathbf{C}_t = \mathbf{I}$ , the method is known as **steepest descent** or **gradient descent**. If we set  $\mathbf{C}_t = \mathbf{H}_t^{-1}$ , where  $\mathbf{H}_t = \nabla^2 \mathcal{L}(\theta_t)$  is the Hessian, we get **Newton's method**. There are many variants of Newton's method 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:

$$\hat{\mathbf{g}}_t = \hat{\nabla} \mathcal{L}(\theta_t) = \frac{1}{B} \sum_{n \in \mathcal{B}_t} \nabla \mathcal{L}_n(\theta_t) \quad (6.43)$$



*Figure 6.2: Changing the mean of a Gaussian by a fixed amount (from solid to dotted curve) can have more impact when the (shared) variance is small (as in a) compared to when the variance is large (as in b). Hence the impact (in terms of prediction accuracy) of a change to  $\mu$  depends on where the optimizer is in  $(\mu, \sigma)$  space. From Figure 3 of [Hon+10], reproduced from [Val00]. Used with kind permission of Antti Honkela.*

Since the minibatches are randomly sampled, this is a stochastic, but unbiased, estimate of the gradient. If we insert  $\hat{\mathbf{g}}_t$  into Equation (6.41), the method is called **stochastic gradient descent** or **SGD**.

## 6.4 Natural gradient descent

In this section, we discuss **natural gradient descent (NGD)** [Ama98], which is a second order method for optimizing the parameters of (conditional) probability distributions  $p_{\theta}(\mathbf{y}|\mathbf{x})$ . The key idea is to compute parameter updates by measuring distances between the induced distributions, rather than comparing parameter values directly.

For example, consider comparing two Gaussians,  $p_{\theta} = p(y|\mu, \sigma)$  and  $p_{\theta'} = p(y|\mu', \sigma')$ . The (squared) Euclidean distance between the parameter vectors decomposes as  $\|\theta - \theta'\|^2 = (\mu - \mu')^2 + (\sigma - \sigma')^2$ . However, the predictive distribution has the form  $\exp(-\frac{1}{2\sigma^2}(y - \mu)^2)$ , so changes in  $\mu$  need to be measured relative to  $\sigma$ . This is illustrated in Figure 6.2(a-b), which shows two univariate Gaussian distributions (dotted and solid lines) whose means differ by  $\delta$ . In Figure 6.2(a), they share the same small variance  $\sigma^2$ , whereas in Figure 6.2(b), they share the same large variance. It is clear that the value of  $\delta$  matters much more (in terms of the effect on the distribution) when the variance is small. Thus we see that the two parameters interact with each other, which the Euclidean distance cannot capture. This problem gets much worse when we consider more complex models, such as deep neural networks. By modeling such correlations, NGD can converge much faster than other gradient methods.

### 6.4.1 Defining the natural gradient

The key to NGD is to measure the notion of distance between two probability distributions in terms of the KL divergence. As we show in Section 2.6.4, this can be approximated in terms of the Fisher

information matrix (FIM). In particular, for any given input  $\mathbf{x}$ , we have

$$D_{\text{KL}}(p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}) \| p_{\boldsymbol{\theta}+\boldsymbol{\delta}}(\mathbf{y}|\mathbf{x})) \approx \frac{1}{2} \boldsymbol{\delta}^T \mathbf{F}_{\mathbf{x}} \boldsymbol{\delta} \quad (6.44)$$

where  $\mathbf{F}_{\mathbf{x}}$  is the FIM

$$\mathbf{F}_{\mathbf{x}}(\boldsymbol{\theta}) = -\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})} [\nabla^2 \log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})] = \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})} [(\nabla \log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})) (\nabla \log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}))^T] \quad (6.45)$$

We can compute the average KL between the current and updated distributions using  $\frac{1}{2} \boldsymbol{\delta}^T \mathbf{F} \boldsymbol{\delta}$ , where  $\mathbf{F}$  is the averaged FIM:

$$\mathbf{F}(\boldsymbol{\theta}) = \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [\mathbf{F}_{\mathbf{x}}(\boldsymbol{\theta})] \quad (6.46)$$

NGD uses the inverse FIM as a preconditioning matrix, i.e., we perform updates of the following form:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \mathbf{F}(\boldsymbol{\theta}_t)^{-1} \mathbf{g}_t \quad (6.47)$$

The term

$$\mathbf{F}^{-1} \mathbf{g}_t = \mathbf{F}^{-1} \nabla \mathcal{L}(\boldsymbol{\theta}_t) \triangleq \tilde{\nabla} \mathcal{L}(\boldsymbol{\theta}_t) \quad (6.48)$$

is called the **natural gradient**.

## 6.4.2 Interpretations of NGD

### 6.4.2.1 NGD as a trust region method

In Section 6.5.2 we show that we can interpret standard gradient descent as optimizing a linear approximation to the objective subject to a penalty on the  $\ell_2$  norm of the change in parameters, i.e., if  $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \boldsymbol{\delta}$ , then we optimize

$$M_t(\boldsymbol{\delta}) = \mathcal{L}(\boldsymbol{\theta}_t) + \mathbf{g}_t^T \boldsymbol{\delta} + \eta \|\boldsymbol{\delta}\|_2^2 \quad (6.49)$$

Now let us replace the squared distance with the squared FIM-based distance,  $\|\boldsymbol{\delta}\|_F^2 = \boldsymbol{\delta}^T \mathbf{F} \boldsymbol{\delta}$ . This is equivalent to squared Euclidean distance in the **whitened coordinate system**  $\boldsymbol{\phi} = \mathbf{F}^{\frac{1}{2}} \boldsymbol{\theta}$ , since

$$\|\boldsymbol{\phi}_{t+1} - \boldsymbol{\phi}_t\|_2^2 = \|\mathbf{F}^{\frac{1}{2}}(\boldsymbol{\theta}_t + \boldsymbol{\delta}) - \mathbf{F}^{\frac{1}{2}} \boldsymbol{\theta}_t\|_2^2 = \|\mathbf{F}^{\frac{1}{2}} \boldsymbol{\delta}\|_2^2 = \|\boldsymbol{\delta}\|_F^2 \quad (6.50)$$

The new objective becomes

$$M_t(\boldsymbol{\delta}) = \mathcal{L}(\boldsymbol{\theta}_t) + \mathbf{g}_t^T \boldsymbol{\delta} + \eta \boldsymbol{\delta}^T \mathbf{F} \boldsymbol{\delta} \quad (6.51)$$

Solving  $\nabla_{\boldsymbol{\delta}} M_t(\boldsymbol{\delta}) = \mathbf{0}$  gives the update

$$\boldsymbol{\delta}_t = -\eta \mathbf{F}^{-1} \mathbf{g}_t \quad (6.52)$$

This is the same as the natural gradient direction. Thus we can view NGD as a trust region method, where we use a first-order approximation to the objective, and use FIM-distance in the constraint.

In the above derivation, we assumed  $\mathbf{F}$  was a constant matrix. In most problems, it will change at each point in space, since we are optimizing in a curved space known as a **Riemannian manifold**.

For certain models, we can compute the FIM efficiently, allowing us to capture curvature information, even though we use a first-order approximation to the objective.

1 **6.4.2.2 NGD as a Gauss-Newton method**

3 If  $p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})$  is an exponential family distribution with natural parameters computed by  $\boldsymbol{\eta} = f(\mathbf{x}, \boldsymbol{\theta})$ ,  
4 then one can show [Hes00; PB14] that NGD is identical to the generalized Gauss-Newton (GGN)  
5 method (Section 17.4.1). Furthermore, in the online setting, these methods are equivalent to  
6 performing sequential Bayesian inference using the extended Kalman filter, as shown in [Oll18].  
7

8 **6.4.3 Benefits of NGD**

10 The use of the FIM as a preconditioning matrix, rather than the Hessian, has two advantages. First,  
11  $\mathbf{F}$  is always positive definite, whereas  $\mathbf{H}$  can have negative eigenvalues at saddle points, which are  
12 prevalent in high dimensional spaces. Second, it is easy to approximate  $\mathbf{F}$  online from minibatches,  
13 since it is an expectation (wrt the empirical distribution) of outer products of gradient vectors. This  
14 is in contrast to Hessian-based methods [Byr+16; Liu+18a], which are much more sensitive to noise  
15 introduced by the minibatch approximation.

16 In addition, the connection with trust region optimization makes it clear that NGD updates  
17 parameters in a way that matter most for prediction, which allows the method to take larger steps in  
18 uninformative regions of parameter space, which can help avoid getting stuck on plateaus. This can  
19 also help with issues that arise when the parameters are highly correlated.

20 For example, consider a 2d Gaussian with an unusual, highly coupled parameterization, proposed  
21 in [SD12]:

22

$$\frac{23}{24} p(\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{2\pi} \exp \left[ -\frac{1}{2} \left( (x_1 - \left[ 3\theta_1 + \frac{1}{3}\theta_2 \right])^2 - \frac{1}{2} \left( x_2 - \left[ \frac{1}{3}\theta_1 \right] \right)^2 \right) \right] \quad (6.53)$$

25

26 The objective is the cross entropy loss:

27

$$\frac{28}{29} \mathcal{L}(\boldsymbol{\theta}) = -\mathbb{E}_{p^*(\mathbf{x})} [\log p(\mathbf{x}; \boldsymbol{\theta})] \quad (6.54)$$

30

29 The gradient of this objective is given by

31

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \left( \begin{array}{c} = \mathbb{E}_{p^*(\mathbf{x})} [3(x_1 - [3\theta_1 + \frac{1}{3}\theta_2]) + \frac{1}{3}(x_2 - [\frac{1}{3}\theta_1])] \\ \mathbb{E}_{p^*(\mathbf{x})} [\frac{1}{3}(x_1 - [3\theta_1 + \frac{1}{3}\theta_2])] \end{array} \right) \quad (6.55)$$

32

33 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 Figure 6.3 compares steepest descent in  $\boldsymbol{\theta}$  space with the natural gradient method, which is  
39 equivalent to steepest descent in  $\phi$  space. Both methods start at  $\boldsymbol{\theta} = (1, -1)$ . The global optimum is  
40 at  $\boldsymbol{\theta} = (0, 0)$ . We see that the NG method (blue dots) converges much faster to this optimum and  
41 takes the shortest path, whereas steepest descent takes a very circuitous route. We also see that  
42 the gradient field in the whitened parameter space is more “spherical”, which makes descent much  
43 simpler and faster.

44 Finally, note that since NGD is invariant to how we parameterize the distribution, we will get the  
45 same results even for a standard parameterization of the Gaussian. This is particularly useful if our  
46 probability model is more complex, such as a DNN (see e.g., [SSE18]).

47

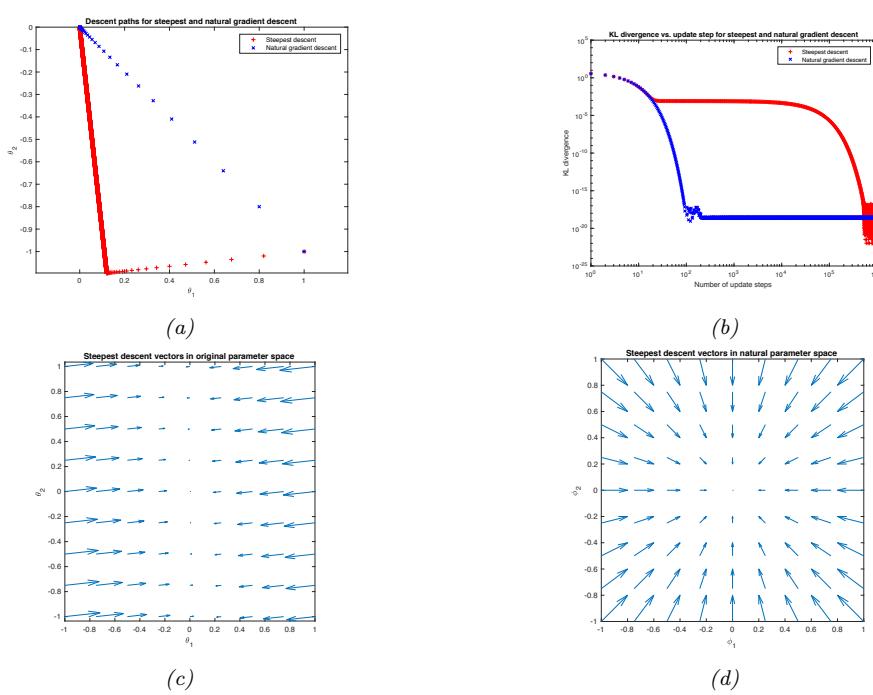


Figure 6.3: Illustration of the benefits of natural gradient vs steepest descent on a 2d problem. (a) Trajectories of the two methods in parameter space (red = steepest descent, blue = NG). They both start in the bottom right, at  $(1, -1)$ . (b) Objective vs number of iterations. (c) Gradient field in the  $\theta$  parameter space. (d) Gradient field in the whitened  $\phi = \mathbf{F}^{\frac{1}{2}}\theta$  parameter space used by NG. Generated by `nat_grad_demo.py`.

#### 6.4.4 Approximating the natural gradient

The main drawback of NGD is the computational cost of computing (the inverse of) the Fisher Information Matrix (FIM). To speed this up, several methods make assumptions about the form of  $\mathbf{F}$ , so it can be inverted efficiently. For example, [LeC+98] uses a diagonal approximation for neural net training; [RMB08] uses a low-rank plus block diagonal approximation; and [GS15] assumes the covariance of the gradients can be modeled by a directed Gaussian graphical model with low treewidth (i.e., the Cholesky factorization of  $\mathbf{F}$  is sparse).

[MG15] propose the **KFAC** method, which stands for “Kronecker-Factored approximate curvature”; this approximates the FIM of a DNN as a block diagonal matrix, where each block is a Kronecker product of two small matrices. This method has shown good results on supervised learning of neural nets [GM16; BGM17; Geo+18; Osa+19b] as well as reinforcement learning of neural policy networks [Wu+17]. The KFAC approximation can be justified using the mean field analysis of [AKO18]. In addition, [ZMG19] prove that KFAC will converge to the global optimum of a DNN if it is overparameterized (i.e., acts like an interpolator).

A simpler approach is to approximate the FIM by replacing the model’s distribution with the empirical distribution. In particular, define  $p_{\mathcal{D}}(\mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{n=1}^N \delta_{\mathbf{x}_n}(\mathbf{x}) \delta_{\mathbf{y}_n}(\mathbf{y})$ ,  $p_{\mathcal{D}}(\mathbf{x}) = \frac{1}{N} \sum_{n=1}^N \delta_{\mathbf{x}_n}(\mathbf{x})$

1 and  $p_{\theta}(\mathbf{x}, \mathbf{y}) = p_{\mathcal{D}}(\mathbf{x})p(\mathbf{y}|\mathbf{x}, \theta)$ . Then we can compute the **empirical Fisher** Martens [Mar16] as  
 2 follows:  
 3

$$\mathbf{F} = \mathbb{E}_{p_{\theta}(\mathbf{x}, \mathbf{y})} [\nabla \log p(\mathbf{y}|\mathbf{x}, \theta) \nabla \log p(\mathbf{y}|\mathbf{x}, \theta)^T] \quad (6.57)$$

$$\approx \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x}, \mathbf{y})} [\nabla \log p(\mathbf{y}|\mathbf{x}, \theta) \nabla \log p(\mathbf{y}|\mathbf{x}, \theta)^T] \quad (6.58)$$

$$= \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \nabla \log p(\mathbf{y}|\mathbf{x}, \theta) \nabla \log p(\mathbf{y}|\mathbf{x}, \theta)^T \quad (6.59)$$

10 This approximation is widely used, since it is simple to compute. In particular, we can compute a  
 11 diagonal approximation using the squared gradient vector. (This is similar to ADAGRAD, but only  
 12 uses the current gradient instead of a moving average of gradients; the latter is a better approach  
 13 when performing stochastic optimization.)

14 Unfortunately, the empirical Fisher does not work as well as the true Fisher [KBH19; Tho+19].  
 15 To see why, note that when we reach a flat part of parameter space where the gradient vector goes  
 16 to zero, the empirical Fisher will become singular, and hence the algorithm will get stuck on this  
 17 plateau. However, the true Fisher takes expectations over the outputs, i.e., it marginalizes out  $\mathbf{y}$ .  
 18 This will allow it to detect small changes in the output if we change the parameters. This is why the  
 19 natural gradient method can “escape” plateaus better than standard gradient methods.

20 An alternative strategy is to use exact computation of  $\mathbf{F}$ , but solve for  $\mathbf{F}^{-1}\mathbf{g}$  approximately  
 21 using truncated conjugate gradient (CG) methods, where each CG step uses efficient methods for  
 22 Hessian-vector products [Pea94]. This is called **Hessian free optimization** [Mar10a]. However,  
 23 this approach can be slow, since it may take many CG iterations to compute a single parameter  
 24 update.

#### 25 6.4.5 Natural gradients for the exponential family

26 In this section, we assume  $\mathcal{L}$  is an expected loss of the following form:

$$27 \quad \mathcal{L}(\boldsymbol{\mu}) = \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\ell(\boldsymbol{\theta})] \quad (6.60)$$

28 where  $q_{\boldsymbol{\mu}}(\boldsymbol{\theta})$  is an exponential family distribution with moment parameters  $\boldsymbol{\mu}$ . This is the basis of  
 29 variational optimization (discussed in Section 6.8.3) and natural evolutionary strategies (discussed in  
 30 the supplementary material).

31 It turns out the gradient wrt the moment parameters is the same as the natural gradient wrt the  
 32 natural parameters  $\boldsymbol{\lambda}$ . This follows from the chain rule:

$$33 \quad \frac{d}{d\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}) = \frac{d\boldsymbol{\mu}}{d\boldsymbol{\lambda}} \frac{d}{d\boldsymbol{\mu}} \mathcal{L}(\boldsymbol{\mu}) = \mathbf{F}(\boldsymbol{\lambda}) \nabla_{\boldsymbol{\mu}} \mathcal{L}(\boldsymbol{\mu}) \quad (6.61)$$

34 where  $\mathcal{L}(\boldsymbol{\mu}) = \mathcal{L}(\boldsymbol{\lambda}(\boldsymbol{\mu}))$ , and where we used Equation (2.193) to write

$$35 \quad \mathbf{F}(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\lambda}} \boldsymbol{\mu}(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\lambda}}^2 A(\boldsymbol{\lambda}) \quad (6.62)$$

36 Hence

$$37 \quad \tilde{\nabla}_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}) = \mathbf{F}(\boldsymbol{\lambda})^{-1} \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\mu}} \mathcal{L}(\boldsymbol{\mu}) \quad (6.63)$$

38 It remains to compute the (regular) gradient wrt the moment parameters. The details on how to  
 39 do this will depend on the form of the  $q$  and the form of  $\mathcal{L}(\boldsymbol{\lambda})$ . We discuss some approaches to this  
 40 problem below.

41

---

### 6.4.5.1 Analytic computation for the Gaussian case

In this section, we assume that  $q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta} | \mathbf{m}, \mathbf{V})$ . We now show how to compute the relevant gradients analytically.

Following Section 2.5.2.5, the natural parameters of  $q$  are

$$\boldsymbol{\lambda}^{(1)} = \mathbf{V}^{-1}\mathbf{m}, \quad \boldsymbol{\lambda}^{(2)} = -\frac{1}{2}\mathbf{V}^{-1} \quad (6.64)$$

and the moment parameters are

$$\boldsymbol{\mu}^{(1)} = \mathbf{m}, \quad \boldsymbol{\mu}^{(2)} = \mathbf{V} + \mathbf{m}\mathbf{m}^\top \quad (6.65)$$

For simplicity, we derive the result for the scalar case. Let  $m = \mu^{(1)}$  and  $v = \mu^{(2)} - (\mu^{(1)})^2$ . By using the chain rule, the gradient wrt the moment parameters are

$$\frac{\partial \mathcal{L}}{\partial \mu^{(1)}} = \frac{\partial \mathcal{L}}{\partial m} \frac{\partial m}{\partial \mu^{(1)}} + \frac{\partial \mathcal{L}}{\partial v} \frac{\partial v}{\partial \mu^{(1)}} = \frac{\partial \mathcal{L}}{\partial m} - 2 \frac{\partial \mathcal{L}}{\partial v} m \quad (6.66)$$

$$\frac{\partial \mathcal{L}}{\partial \mu^{(2)}} = \frac{\partial \mathcal{L}}{\partial m} \frac{\partial m}{\partial \mu^{(2)}} + \frac{\partial \mathcal{L}}{\partial v} \frac{\partial v}{\partial \mu^{(2)}} = \frac{\partial \mathcal{L}}{\partial v} \quad (6.67)$$

It remains to compute the derivatives wrt  $m$  and  $v$ . If  $\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{m}, \mathbf{V})$ , then from **Bonnet's theorem** [Bon64] we have

$$\frac{\partial}{\partial m_i} \mathbb{E} [\ell(\boldsymbol{\theta})] = \mathbb{E} \left[ \frac{\partial}{\partial \theta_i} \ell(\boldsymbol{\theta}) \right] \quad (6.68)$$

And from **Price's theorem** [Pri58] we have

$$\frac{\partial}{\partial V_{ij}} \mathbb{E} [\ell(\boldsymbol{\theta})] = c_{ij} \mathbb{E} \left[ \frac{\partial^2}{\partial \theta_i \partial \theta_j} \ell(\boldsymbol{\theta}) \right] \quad (6.69)$$

where  $c_{ij} = \frac{1}{2}$  is  $i = j$  and  $c_{ij} = 1$  otherwise. (See `gradient_expected_value_gaussian.py` for a “proof by example” of these claims.)

Hence we can state the general result (see Equations 10–11 of [KR21a]):

$$\nabla_{\boldsymbol{\mu}^{(1)}} \mathbb{E}_{q(\boldsymbol{\theta})} [\ell(\boldsymbol{\theta})] = \nabla_{\mathbf{m}} \mathbb{E}_{q(\boldsymbol{\theta})} [\ell(\boldsymbol{\theta})] - 2 \nabla_{\mathbf{V}} \mathbb{E}_{q(\boldsymbol{\theta})} [\ell(\boldsymbol{\theta})] \mathbf{m} \quad (6.70)$$

$$= \mathbb{E}_{q(\boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta})] - \mathbb{E}_{q(\boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}}^2 \ell(\boldsymbol{\theta})] \mathbf{m} \quad (6.71)$$

$$\nabla_{\boldsymbol{\mu}^{(2)}} \mathbb{E}_{q(\boldsymbol{\theta})} [\ell(\boldsymbol{\theta})] = \nabla_{\mathbf{V}} \mathbb{E}_{q(\boldsymbol{\theta})} [\ell(\boldsymbol{\theta})] \quad (6.72)$$

$$= \frac{1}{2} \mathbb{E}_{q(\boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}}^2 \ell(\boldsymbol{\theta})] \quad (6.73)$$

Thus we see that the natural gradients rely on both the gradient and Hessian of the loss function  $\ell(\boldsymbol{\theta})$ . We will see applications of this result in Section 6.8.2.2.

### 6.4.5.2 Stochastic approximation for the general case

In general, it can be hard to analytically compute the natural gradient. However, we can compute a Monte Carlo approximation. To see this, let us assume  $\mathcal{L}$  is an expected loss of the following form:

$$\mathcal{L}(\boldsymbol{\mu}) = \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\ell(\boldsymbol{\theta})] \quad (6.74)$$

1 From Equation (6.63) the natural gradient is given by  
2

$$\underline{3} \quad \nabla_{\boldsymbol{\mu}} \mathcal{L}(\boldsymbol{\mu}) = \mathbf{F}(\boldsymbol{\lambda})^{-1} \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}) \quad (6.75)$$

5 For exponential family distributions, both of these terms on the RHS can be written as expectations,  
6 and hence can be approximated by Monte Carlo, as noted by [KL17a]. To see this, note that  
7

$$\underline{8} \quad \mathbf{F}(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\lambda}} \boldsymbol{\mu}(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})} [\mathcal{T}(\boldsymbol{\theta})] \quad (6.76)$$

$$\underline{9} \quad \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})} [\ell(\boldsymbol{\theta})] \quad (6.77)$$

10 If  $q$  is reparameterizable, we can apply the reparameterization trick (Section 6.6.4) to push the  
11 gradient inside the expectation operator. This lets us sample  $\boldsymbol{\theta}$  from  $q$ , compute the gradients, and  
12 average; we can then pass the resulting stochastic gradients to SGD.  
13

#### 14 6.4.5.3 Natural gradient of the entropy function

16 In this section, we discuss how to compute the natural gradient of the entropy of an exponential  
17 family distribution, which is useful when performing variational inference (Chapter 10). The natural  
18 gradient is given by  
19

$$\underline{20} \quad \tilde{\nabla}_{\boldsymbol{\lambda}} \mathbb{H}(\boldsymbol{\lambda}) = -\nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\log q(\boldsymbol{\theta})] \quad (6.78)$$

21 where, from Equation (2.123), we have  
22

$$\underline{23} \quad \log q(\boldsymbol{\theta}) = \log h(\boldsymbol{\theta}) + \mathcal{T}(\boldsymbol{\theta})^T \boldsymbol{\lambda} - A(\boldsymbol{\lambda}) \quad (6.79)$$

24 Since  $\mathbb{E}[\mathcal{T}(\boldsymbol{\theta})] = \boldsymbol{\mu}$ , we have  
25

$$\underline{26} \quad \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\log q(\boldsymbol{\theta})] = \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q(\boldsymbol{\theta})} [\log h(\boldsymbol{\theta})] + \nabla_{\boldsymbol{\mu}} \boldsymbol{\mu}^T \boldsymbol{\lambda}(\boldsymbol{\mu}) - \nabla_{\boldsymbol{\mu}} A(\boldsymbol{\lambda}) \quad (6.80)$$

28 where  $h(\boldsymbol{\theta})$  is the base measure. Since  $\boldsymbol{\lambda}$  is a function of  $\boldsymbol{\mu}$ , we have  
29

$$\underline{30} \quad \nabla_{\boldsymbol{\mu}} \boldsymbol{\mu}^T \boldsymbol{\lambda} = \boldsymbol{\lambda} + (\nabla_{\boldsymbol{\mu}} \boldsymbol{\lambda})^T \boldsymbol{\mu} = \boldsymbol{\lambda} + (\mathbf{F}_{\boldsymbol{\lambda}}^{-1} \nabla_{\boldsymbol{\lambda}} \boldsymbol{\lambda})^T \boldsymbol{\mu} = \boldsymbol{\lambda} + \mathbf{F}_{\boldsymbol{\lambda}}^{-1} \boldsymbol{\mu} \quad (6.81)$$

31 and since  $\boldsymbol{\mu} = \nabla_{\boldsymbol{\lambda}} A(\boldsymbol{\lambda})$  we have  
32

$$\underline{33} \quad \nabla_{\boldsymbol{\mu}} A(\boldsymbol{\lambda}) = \mathbf{F}_{\boldsymbol{\lambda}}^{-1} \nabla_{\boldsymbol{\lambda}} A(\boldsymbol{\lambda}) = \mathbf{F}_{\boldsymbol{\lambda}}^{-1} \boldsymbol{\mu} \quad (6.82)$$

34 Hence  
35

$$\underline{36} \quad -\nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\log q(\boldsymbol{\theta})] = -\nabla_{\boldsymbol{\mu}} \mathbb{E}_{q(\boldsymbol{\theta})} [\log h(\boldsymbol{\theta})] - \boldsymbol{\lambda} \quad (6.83)$$

38 If we assume that  $h(\boldsymbol{\theta}) = \text{const}$ , as is often the case, we get  
39

$$\underline{40} \quad \tilde{\nabla}_{\boldsymbol{\lambda}} \mathbb{H}(q) = -\boldsymbol{\lambda} \quad (6.84)$$

41

## 42 6.5 Mirror descent

44 In this section, we discuss **mirror descent**, which is like gradient descent, but can leverage non-  
45 Euclidean geometry to potentially speed up convergence, or enforce certain constraints. But to explain  
46 the method, we first need to introduce some background concepts.  
47

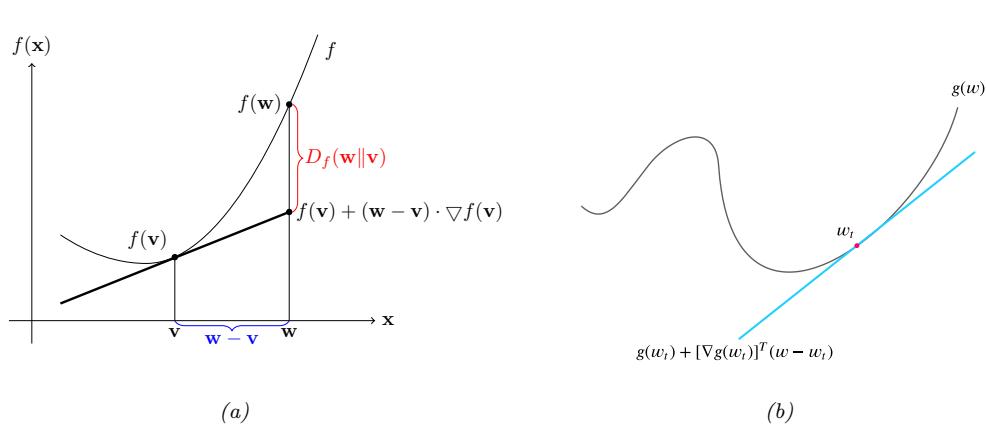


Figure 6.4: (a) Illustration of Bregman divergence. (b) A locally linear approximation to a non-convex function.

### 6.5.1 Bregman divergence

Let  $f : \Omega \rightarrow \mathbb{R}$  be a continuously differentiable, strictly convex function defined on a closed convex set  $\Omega$ . We define the **Bregman divergence** associated with  $f$  as follows [Bre67]:

$$D_f(\mathbf{w}, \mathbf{v}) = f(\mathbf{w}) - f(\mathbf{v}) - (\mathbf{w} - \mathbf{v})^\top \nabla f(\mathbf{v}) \quad (6.85)$$

To understand this, let

$$\hat{f}_v(\mathbf{w}) = f(\mathbf{v}) + (\mathbf{w} - \mathbf{v})^\top \nabla f(\mathbf{v}) \quad (6.86)$$

be a first order Taylor series approximation to  $f$  centered at  $\mathbf{v}$ . Then the Bregman divergence is the difference from this linear approximation:

$$D_f(\mathbf{w}, \mathbf{v}) = f(\mathbf{w}) - \hat{f}_v(\mathbf{w}) \quad (6.87)$$

See Figure 6.4a for an illustration. Since  $f$  is convex, we have  $D_f(\mathbf{v}||\mathbf{w}) \geq 0$ , since  $\hat{f}_v$  is a linear lower bound on  $f$ .

Below we mention some important special cases of Bregman divergences.

- If  $f(\mathbf{w}) = \|\mathbf{w}\|^2$ , then  $D_f(\mathbf{w}, \mathbf{v}) = \|\mathbf{w} - \mathbf{v}\|^2$  is the squared Euclidean distance.
- If  $f(\mathbf{w}) = \mathbf{w}^\top \mathbf{Q} \mathbf{w}$ , then  $D_f(\mathbf{w}, \mathbf{v})$  is the squared Mahalanobis distance (Section 2.3.1).
- If  $\mathbf{w}$  are the natural parameters of an exponential family distribution, and  $f(\mathbf{w}) = \log Z(\mathbf{w})$  is the log normalizer, then the Bregman divergence is the same as the Kullback Leibler divergence, as we show in Section 5.1.7.2.

1 **6.5.2 Proximal point method**

3 Suppose we make a locally linear approximation to the objective at step  $t$  centered at  $\theta_t$ :

4 
$$\hat{\mathcal{L}}_t(\theta) = \mathcal{L}_t(\theta_t) + \mathbf{g}_t^\top(\theta - \theta_t) \quad (6.88)$$

5 where  $\mathbf{g}_t = \nabla_{\theta} \mathcal{L}_t(\theta_t)$ . This is shown in Figure 6.4b.

6 If we optimize this approximation using gradient descent, we may end up at  $-\infty$ . To prevent this,  
7 we can use a **proximal update**, which adds a quadratic penalty to ensure we don't move too far  
8 from where the locally linear approximation is valid:

9 
$$\theta_{t+1} = \text{prox}_{\eta_t \hat{\mathcal{L}}_t}(\theta_t) \triangleq \underset{\theta}{\operatorname{argmin}} \hat{\mathcal{L}}_t(\theta) + \frac{1}{2\eta_t} \|\theta - \theta_t\|_2^2 \quad (6.89)$$

10 We can solve this optimization problem by setting the gradient to 0:

11 
$$\nabla_{\theta} \left[ \mathcal{L}_t(\theta_t) + \mathbf{g}_t^\top(\theta - \theta_t) + \frac{1}{2\eta_t} \|\theta - \theta_t\|_2^2 \right] = \mathbf{g}_t + \frac{1}{\eta_t}(\theta - \theta_t) = \mathbf{0} \quad (6.90)$$

12 This yields the standard gradient descent update:

13 
$$\theta_{t+1} = \theta_t - \eta_t \mathbf{g}_t \quad (6.91)$$

14 However, by changing from Euclidean norm to another distance metric, we can derive mirror descent,  
15 as we show below.

16 **6.5.3 PPM using Bregman divergence**

17 Suppose we replace the Euclidean distance term  $\|\theta - \theta_t\|_2^2$  by a more general Bregman divergence  
18 (Equation (6.85)),

19 
$$D_h(\mathbf{x}, \mathbf{y}) = h(\mathbf{x}) - [h(\mathbf{y}) + \nabla h(\mathbf{y})^\top(\mathbf{x} - \mathbf{y})] \quad (6.92)$$

20 where  $h(\mathbf{x})$  is a strongly convex function. Combined with our linear approximation, this gives the  
21 following update:

22 
$$\theta_{t+1} = \underset{\theta}{\operatorname{argmin}} \hat{\mathcal{L}}(\theta) + \frac{1}{\eta_t} D_h(\theta, \theta_t) \quad (6.93)$$

23 
$$= \underset{\theta}{\operatorname{argmin}} \eta_t \mathbf{g}_t^\top \theta + D_h(\theta, \theta_t) \quad (6.94)$$

24 This is known as **mirror descent** [NY83; BT03]. This can easily be extended to the stochastic  
25 setting in the obvious way.

26 One can show that natural gradient descent (Section 6.4) is a form of mirror descent [RM15a].

27 More precisely, mirror descent in the mean parameter space is equivalent to natural gradient descent  
28 in the canonical parameter space.

29

30 **6.6 Gradients of stochastic functions**

31 In this section, we discuss how to compute the gradient of stochastic functions of the form

32 
$$\mathcal{L}(\psi) = \mathbb{E}_{q_\psi(z)} [\ell(\psi, z)] \quad (6.95)$$

33

1 **6.6.1 Minibatch approximation to finite-sum objectives**

2 In the simplest case,  $q_\psi(\mathbf{z})$  does not depend on  $\psi$ . In this case, we can push gradients inside the  
3 expectation operator,  $\nabla \mathcal{L}(\psi) = \mathbb{E}[\nabla \ell(\psi, \mathbf{z})]$  and then use Monte Carlo sampling for  $\mathbf{z}$  to approximate  
4 the gradient.  
5

6 For example, consider the empirical risk minimization (ERM) problem of minimizing  
7

$$\mathcal{L}(\psi) = \frac{1}{N} \sum_{n=1}^N \ell(\psi, \mathbf{z}_n) \quad (6.96)$$

8 where  $\mathbf{z}_n = (\mathbf{x}_n, \mathbf{y}_n)$  and  
9

$$\ell(\psi, \mathbf{z}_n) = \ell(h(\mathbf{x}_n; \psi), \mathbf{y}_n) \quad (6.97)$$

10 is the per-example loss, where  $h$  is a prediction function. This kind of objective is called a **finite  
11 sum objective**.

12 Now consider trying to minimize this objective. If, at each iteration, we evaluate the objective (and  
13 its gradient) using all  $N$  datapoints, the method is called **batch optimization**. However, this can  
14 be very slow if the dataset is large. Fortunately, we can reformulate it as a stochastic optimization  
15 problem, which will be faster to solve. To do this, note that Equation (6.96) can be written as an  
16 expectation wrt the empirical distribution:  
17

$$\mathcal{L}(\psi) = \mathbb{E}_{\mathbf{z} \sim p_D} [\ell(\psi, \mathbf{z})] \quad (6.98)$$

18 Since the distribution is independent of the parameters, we can easily use Monte Carlo sampling to  
19 approximate the objective and its gradient. In particular, we will sample a **minibatch** of  $B = |\mathcal{B}|$   
20 datapoints from the full set  $\mathcal{D}$  at each iteration. More precisely, we have  
21

$$\mathcal{L}(\psi) \approx \frac{1}{B} \sum_{n \in \mathcal{B}} \ell(h(\mathbf{x}_n; \psi), \mathbf{y}_n) \quad (6.99)$$

$$\nabla \mathcal{L}(\psi) \approx \frac{1}{B} \sum_{n \in \mathcal{B}} \nabla \ell(h(\mathbf{x}_n; \psi), \mathbf{y}_n) \quad (6.100)$$

22 These noisy gradients can then be passed to SGD, which is robust to noisy gradients (see Section 6.3).  
23

24 **6.6.2 Optimizing parameters of a distribution**

25 Now suppose the stochasticity depends on the parameters we are optimizing. For example,  $\mathbf{z}$  could  
26 be an action sampled from a stochastic policy  $q_\psi$ , as in RL (Section 37.3.2). In this case, the gradient  
27 is given by  
28

$$\nabla_\psi \mathbb{E}_{q_\psi(\mathbf{z})} [\ell(\psi, \mathbf{z})] = \nabla_\psi \int \ell(\psi, \mathbf{z}) q_\psi(\mathbf{z}) d\mathbf{z} \quad (6.101)$$

$$= \int [\nabla_\psi \ell(\psi, \mathbf{z})] q_\psi(\mathbf{z}) d\mathbf{z} + \int \ell(\psi, \mathbf{z}) [\nabla_\psi q_\psi(\mathbf{z})] d\mathbf{z} \quad (6.102)$$

The first term can be approximated by Monte Carlo sampling:

$$\int [\nabla_{\psi} \ell(\psi, z)] q_{\psi}(z) dz \approx \frac{1}{S} \sum_{s=1}^S \nabla_{\psi} \ell(\psi, z_s) \quad (6.103)$$

where  $z_s \sim q_{\psi}$ . Note that if  $\ell()$  is independent of  $\psi$ , this term vanishes.

Now consider the second term, that takes the gradients of the distribution itself:

$$I \triangleq \int \ell(\psi, z) [\nabla_{\psi} q_{\psi}(z)] dz \quad (6.104)$$

We can no longer use vanilla Monte Carlo sampling to approximate this integral. However, there are various other ways to approximate this (see [Moh+19a] for an extensive review). We briefly describe the two main methods in Section 6.6.3 and Section 6.6.4.

### 6.6.3 Score function estimator (likelihood ratio trick)

The simplest way to approximate Equation (6.104) is to exploit the **log derivative trick**, which is the following identity:

$$\nabla_{\psi} q_{\psi}(z) = q_{\psi}(z) \nabla_{\psi} \log q_{\psi}(z) \quad (6.105)$$

With this, we can rewrite Equation (6.104) as follows:

$$I = \int \ell(\psi, z) [q_{\psi}(z) \nabla_{\psi} \log q_{\psi}(z)] dz = \mathbb{E}_{q_{\psi}(z)} [\ell(\psi, z) \nabla_{\psi} \log q_{\psi}(z)] \quad (6.106)$$

This is called the **score function estimator** or **SFE** [Fu15]. (The term “score function” refers to the gradient of a log probability distribution, as explained in Section 2.6.1.) It is also called the **likelihood ratio gradient estimator**. We can now easily approximate this with Monte Carlo:

$$I \approx \frac{1}{S} \sum_{s=1}^S \ell(\psi, z_s) \nabla_{\psi} \log q_{\psi}(z_s) \quad (6.107)$$

We only require that the sampling distribution is differentiable, not the objective  $\ell(\psi, z)$  itself. This allows the method to be used for blackbox stochastic optimization problems, such as variational optimization (Section 6.8.3), black-box variational inference (Section 10.3.1), reinforcement learning (Section 37.3.2), etc.

#### 6.6.3.1 Control variates

The score function estimate can have high variance. One way to reduce this is to use **control variates**, in which we replace  $\ell(\psi, z)$  with

$$\hat{\ell}(\psi, z) = \ell(\psi, z) - c(b(\psi, z) - \mathbb{E}[b(\psi, z)]) \quad (6.108)$$

where  $b(\psi, z)$  is a **baseline function** that is correlated with  $\ell(\psi, z)$ , and  $c > 0$  is a coefficient. Since  $\mathbb{E}[\hat{\ell}(\psi, z)] = \mathbb{E}[\ell(\psi, z)]$ , we can use  $\hat{\ell}$  to compute unbiased gradient estimates of  $\ell$ . The advantage is that this new estimate can result in lower variance, as we show in Section 11.6.2.

---

### 6.6.3.2 Rao-Blackwellisation

Suppose  $q_\psi(\mathbf{z})$  is a discrete distribution. In this case, our objective becomes  $\mathcal{L}(\psi) = \sum_{\mathbf{z}} \ell(\psi, \mathbf{z}) q_\psi(\mathbf{z})$ . For simplicity, let us assume  $\ell(\psi, \mathbf{z}) = \ell(\mathbf{z})$ .

We can now easily compute gradients using  $\nabla_\psi \mathcal{L}(\psi) = \sum_{\mathbf{z}} \ell(\mathbf{z}) \nabla_\psi q_\psi(\mathbf{z})$ . Of course, if  $\mathbf{z}$  can take on exponentially many values (e.g., we are optimizing over the space of strings), this expression is intractable. However, suppose we can partition this sum into two sets, a small set  $S_1$  of high probability values and a large set  $S_2$  of all other values. Then we can enumerate over  $S_1$  and use the score function estimator for  $S_2$ :

$$\nabla_\psi \mathcal{L}(\psi) = \sum_{\mathbf{z} \in S_1} \ell(\mathbf{z}) \nabla_\psi q_\psi(\mathbf{z}) + \mathbb{E}_{q_\psi(\mathbf{z}|\mathbf{z} \in S_2)} [\ell(\mathbf{z}) \nabla_\psi \log q_\psi(\mathbf{z})] \quad (6.109)$$

To compute the second expectation, we can use rejection sampling applied to samples from  $q_\psi(\mathbf{z})$ . This procedure is a form of Rao-Blackwellisation as shown in [Liu+19b], and reduces the variance compared to standard SFE (see Section 11.6.1 for details on Rao-Blackwellisation).

### 6.6.4 Reparameterization trick

The score function estimator can have high variance, even when using a control variate. In this section, we derive a lower variance estimator, which can be applied if  $\ell(\psi, \mathbf{z})$  is differentiable wrt  $\mathbf{z}$ . We additionally require that we can compute a sample from  $q_\psi(\mathbf{z})$  by first sampling  $\epsilon$  from some noise distribution  $q_0$  which is independent of  $\psi$ , 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  $\psi$ , we can push the gradient operator inside the expectation, which we can approximate with Monte Carlo:

$$\nabla_\psi \mathcal{L}(\psi) = \mathbb{E}_{q_0(\epsilon)} [\nabla_\psi \ell(\psi, r(\psi, \epsilon))] \approx \frac{1}{S} \sum_{s=1}^S \nabla_\psi \ell(\psi, r(\psi, \epsilon_s)) \quad (6.112)$$

where  $\epsilon_s \sim q_0$ . This is called the **reparameterization gradient** or **pathwise derivative** [Gla03; Fu15; KW14; RMW14a; TLG14; JO18; FMM18], and is widely used in variational inference (Section 10.3.3), and when fitting VAE models (Section 22.2).

#### 6.6.4.1 Example

As a simple example, suppose we define some arbitrary function, such as  $\ell(z) = z^2 - 3z$ , and then define its expected value as  $\mathcal{L}(\psi) = \mathbb{E}_{\mathcal{N}(z|\mu, v)} [\ell(z)]$ , where  $\psi = (\mu, v)$  and  $v = \sigma^2$ . Suppose we want to compute

$$\nabla_\psi \mathcal{L}(\psi) = \left[ \frac{\partial}{\partial \mu} \mathbb{E} [\ell(z)], \frac{\partial}{\partial v} \mathbb{E} [\ell(z)] \right] \quad (6.113)$$

1 Since the Gaussian distribution is reparameterizable, we can sample  $z \sim \mathcal{N}(z|\mu, v)$ , and then use  
2 automatic differentiation to compute each of these gradient terms, and then average.  
3

4 However, in the special case of Gaussian distributions, we can also compute the gradient vector  
5 directly. In particular, in Section 6.4.5.1 we present Bonnet's theorem, which states that

$$\frac{\partial}{\partial \mu} \mathbb{E} [\ell(z)] = \mathbb{E} \left[ \frac{\partial}{\partial z} \ell(z) \right] \quad (6.114)$$

9  
10 Similarly, Price's theorem states that

$$\frac{\partial}{\partial v} \mathbb{E} [\ell(z)] = 0.5 \mathbb{E} \left[ \frac{\partial^2}{\partial z^2} \ell(z) \right] \quad (6.115)$$

11  
12 In `gradient_expected_value_gaussian.py` we show that these two methods are numerically equivalent,  
13 as theory suggests.

#### 18 6.6.4.2 Total derivative

20 To compute the gradient term inside the expectation in Equation (6.112) we need to use the **total**  
21 **derivative**, since the function  $\ell$  depends on  $\psi$  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  $\psi_i$  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  
29 and hence

$$\nabla_{\psi} \ell(\psi, z)^{\text{TD}} = \nabla_{\psi} \ell(\psi, z) + \mathbf{J}^T \nabla_z \ell(\psi, z) \quad (6.117)$$

32  
33 where  $\mathbf{J} = \frac{\partial z^T}{\partial \psi}$  is the  $d_z \times d_\psi$  Jacobian matrix of the noise transformation:  
34

$$\mathbf{J} = \begin{pmatrix} \frac{\partial z_1}{\partial \psi_1} & \dots & \frac{\partial z_1}{\partial \psi_{d_\psi}} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_{d_z}}{\partial \psi_1} & \dots & \frac{\partial z_{d_z}}{\partial \psi_{d_\psi}} \end{pmatrix} \quad (6.118)$$

40  
41 Hence we can compute the gradient using

$$\nabla_{\psi} \mathcal{L}(\psi) = \mathbb{E}_{q_0(\epsilon)} [\nabla_{\psi} \ell(\psi, z) + \mathbf{J}(\psi, \epsilon)^T \nabla_z \ell(\psi, r(\psi, \epsilon))] \quad (6.119)$$

44  
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.

1 **6.6.5 The delta method**

2 The **delta method** [Hoe12] approximates the expectation of a function of a random variable by the  
3 expectation of the function's Taylor expansion. For example, suppose  $\boldsymbol{\theta} \sim q$  with mean  $\mathbb{E}_{q(\boldsymbol{\theta})}[\boldsymbol{\theta}] = \mathbf{m}$ ,  
4 and we use a first order expansion. Then we have

5 
$$\mathbb{E}_{q(\boldsymbol{\theta})}[f(\boldsymbol{\theta})] \approx \mathbb{E}_{q(\boldsymbol{\theta})}[f(\mathbf{m}) + (\boldsymbol{\theta} - \mathbf{m})^\top \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}}] \approx f(\mathbf{m}) \quad (6.120)$$

6 Now let  $f(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$ . Then we have

7 
$$\mathbb{E}_{q(\boldsymbol{\theta})}[\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})] \approx \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}} \quad (6.121)$$

8 This is called the **first-order delta method**.

9 Now let  $f(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}}^2 \mathcal{L}(\boldsymbol{\theta})$ . Then we have

10 
$$\mathbb{E}_{q(\boldsymbol{\theta})}[\nabla_{\boldsymbol{\theta}}^2 \mathcal{L}(\boldsymbol{\theta})] \approx \nabla_{\boldsymbol{\theta}}^2 \mathcal{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}} \quad (6.122)$$

11 This is called the **second-order delta method**.

12 **6.6.6 Gumbel softmax trick**

13 When working with discrete variables, we cannot use the reparameterization trick. However, we can  
14 often relax the discrete variables to continuous ones in a way which allows the trick to be used, as we  
15 explain below.

16 Consider a one-hot vector  $\mathbf{d}$  with  $K$  bits, so  $d_k \in \{0, 1\}$  and  $\sum_{k=1}^K d_k = 1$ . This can be used  
17 to represent a  $K$ -ary categorical variable  $d$ . Let  $P(d) = \text{Cat}(d|\boldsymbol{\pi})$ , where  $\pi_k = P(d_k = 1)$ , so  
18  $0 \leq \pi_k \leq 1$ . Alternatively we can parameterize the distribution in terms of  $(\alpha_1, \dots, \alpha_K)$ , where  
19  $\pi_k = \alpha_k / (\sum_{k'=1}^K \alpha_{k'})$ . We will denote this by  $d \sim \text{Cat}(d|\boldsymbol{\alpha})$ .

20 We can sample a one-hot vector  $\mathbf{d}$  from this distribution by computing

21 
$$\mathbf{d} = \text{onehot}(\underset{k}{\operatorname{argmax}} [\epsilon_k + \log \alpha_k]) \quad (6.123)$$

22 where  $\epsilon_k \sim \text{Gumbel}(0, 1)$  is sampled from the **Gumbel distribution** [Gum54]. We can draw such  
23 samples by first sampling  $u_k \sim \text{Unif}(0, 1)$  and then computing  $\epsilon_k = -\log(-\log(u_k))$ . This is  
24 called the **Gumbel-Max trick** [MTM14], and gives us a reparameterizable representation for the  
25 categorical distribution.

26 Unfortunately, the derivative of the argmax is 0 everywhere except at the boundary of transitions  
27 from one label to another, where the derivative is undefined. However, suppose we replace the argmax  
28 with a softmax, and replace the discrete one-hot vector  $\mathbf{d}$  with a continuous relaxation  $\mathbf{x} \in \Delta^{K-1}$ ,  
29 where  $\Delta^{K-1} = \{\mathbf{x} \in \mathbb{R}^K : x_k \in [0, 1], \sum_{k=1}^K x_k = 1\}$  is the  $K$ -dimensional simplex. Then we can  
30 write

31 
$$x_k = \frac{\exp((\log \alpha_k + \epsilon_k)/\tau)}{\sum_{k'=1}^K \exp((\log \alpha_{k'} + \epsilon_{k'})/\tau)} \quad (6.124)$$

32 where  $\tau > 0$  is a temperature parameter. This is called the **Gumbel-Softmax distribution** [JGP17]  
33 or the **concrete distribution** [MMT17]. This smoothly approaches the discrete distribution as  
34  $\tau \rightarrow 0$ , as illustrated in Figure 6.5.

35 We can now replace  $f(\mathbf{d})$  with  $f(\mathbf{x})$ , which allows us to take reparameterized gradients wrt  $\mathbf{x}$ .

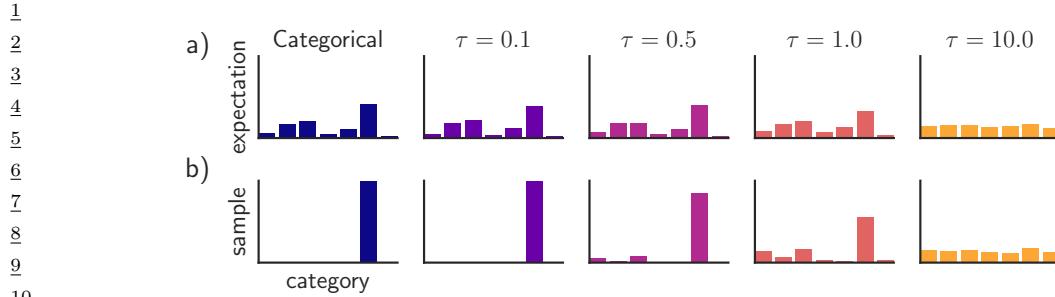


Figure 6.5: Illustration of the Gumbel-Softmax (concrete) distribution with  $K = 7$  states at different temperatures  $\tau$ . The top row shows  $\mathbb{E}[z]$ , and the bottom row shows samples  $z \sim \text{GumbelSoftmax}(\alpha, \tau)$ . The left column shows a discrete (categorical) distribution, which always produces one-hot samples. From Figure 1 of [JGP17]. Used with kind permission of Ben Poole.

### 6.6.7 Stochastic computation graphs

We can represent an arbitrary function containing both deterministic and stochastic components as a **stochastic computation graph**. We can then generalize the AD algorithm (Section 6.2) to leverage score function estimation (Section 6.6.3) and reparameterization (Section 6.6.4) to compute Monte Carlo gradients for complex nested functions. For details, see [Sch+15a; Gaj+19].

### 6.6.8 Straight-through estimator

In this section, we discuss how to approximate the gradient of a quantized version of a signal. For example, suppose we have the following thresholding function, that binarizes its output:

$$f(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \quad (6.125)$$

This does not have a well-defined gradient. However, we can use the **straight-through estimator** proposed in [Ben13] as an approximation. The basic idea is to replace  $g(x) = f'(x)$ , where  $f'(x)$  is the derivative of  $f$  wrt input, with  $g(x) = x$  when computing the backwards pass. See Figure 6.6 for a visualization, and [Yin+19b] for an analysis of why this is a valid approximation.

In practice, we sometimes replace  $g(x) = x$  with the **hard tanh** function, defined by

$$\text{HardTanh}(x) = \begin{cases} x & \text{if } -1 \leq x \leq 1 \\ 1 & \text{if } x > 1 \\ -1 & \text{if } x < -1 \end{cases} \quad (6.126)$$

This ensures the gradients that are backpropagated don't get too large. See Section 22.6 for an application of this approach to discrete autoencoders.

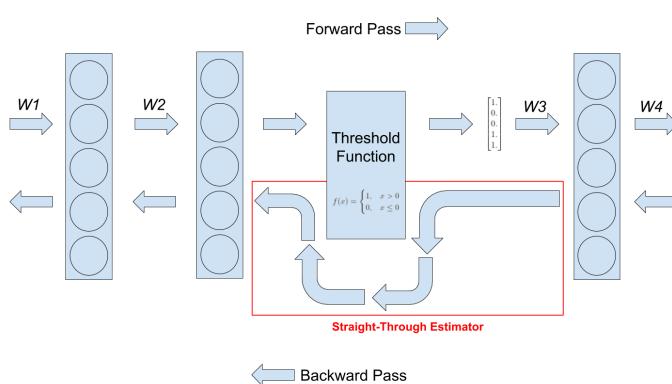


Figure 6.6: Illustration of straight-through estimator when applied to a binary threshold function in the middle of an MLP. From <https://www.hassanaskary.com/python/pytorch/deep%20learning/2020/09/19/intuitive-explanation-of-straight-through-estimators.html>. Used with kind permission of Hassan Askary.

## 6.7 Bound optimization (MM) algorithms

In this section, we consider a class of algorithms known as **bound optimization** or **MM** algorithms. In the context of minimization, MM stands for **majorize-minimize**. In the context of maximization, MM stands for **minorize-maximize**. There are many examples of MM algorithms, such as EM (Section 6.7.3), proximal gradient methods (Section 4.1), the mean shift algorithm for clustering [FH75; Che95; FT05], etc. For more details, see e.g., [HL04; Mai15; SBP17; Nad+19],

### 6.7.1 The general algorithm

In this section, we assume our goal is to *maximize* some function  $LL(\boldsymbol{\theta})$  wrt its parameters  $\boldsymbol{\theta}$ . The basic approach in MM algorithms is to construct a **surrogate function**  $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t)$  which is a tight lowerbound to  $LL(\boldsymbol{\theta})$  such that  $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t) \leq LL(\boldsymbol{\theta})$  and  $Q(\boldsymbol{\theta}^t, \boldsymbol{\theta}^t) = LL(\boldsymbol{\theta})$ . If these conditions are met, we say that  $Q$  minorizes  $LL$ . We then perform the following update at each step:

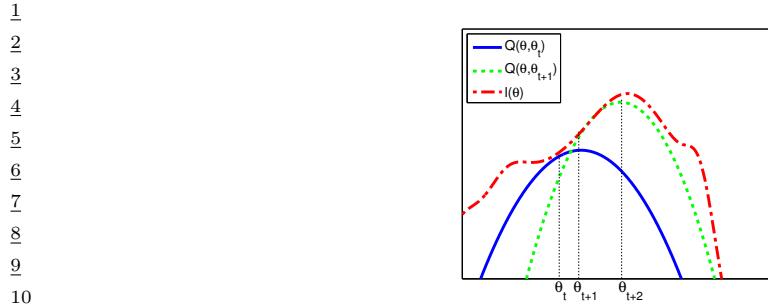
$$\boldsymbol{\theta}^{t+1} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t) \quad (6.127)$$

This guarantees us monotonic increases in the original objective:

$$\ell(\boldsymbol{\theta}^{t+1}) \geq Q(\boldsymbol{\theta}^{t+1}, \boldsymbol{\theta}^t) \geq Q(\boldsymbol{\theta}^t, \boldsymbol{\theta}^t) = \ell(\boldsymbol{\theta}^t) \quad (6.128)$$

where the first inequality follows since  $Q(\boldsymbol{\theta}^t, \boldsymbol{\theta}')$  is a lower bound on  $\ell(\boldsymbol{\theta}^t)$  for any  $\boldsymbol{\theta}'$ ; the second inequality follows from Equation (6.127); and the final equality follows the tightness property. As a consequence of this result, if you do not observe monotonic increase of the objective, you must have an error in your math and/or code. This is a surprisingly powerful debugging tool.

This process is sketched in Figure 6.7. The dashed red curve is the original function (e.g., the log-likelihood of the observed data). The solid blue curve is the lower bound, evaluated at  $\boldsymbol{\theta}^t$ ; this



11 *Figure 6.7: Illustration of a bound optimization algorithm. Adapted from Figure 9.14 of [Bis06]. Generated*  
12 *by emLogLikelihoodMax.py.*

13

14 touches the objective function at  $\theta^t$ . We then set  $\theta^{t+1}$  to the maximum of the lower bound (blue  
15 curve), and fit a new bound at that point (dotted green curve). The maximum of this new bound  
16 becomes  $\theta^{t+2}$ , etc.  
17

18

19 **6.7.2 Example: logistic regression**  
20

21 If  $LL(\theta)$  is a concave function we want to maximize, then one way to obtain a valid lower bound is  
22 to use a bound on its Hessian, i.e., to find a matrix a negative definite matrix  $\mathbf{B}$  such that  $\mathbf{H}(\theta) \succ \mathbf{B}$ .  
23 In this case, one can show (see [BCN18, App. B]) that

24

$$\underline{25} \quad LL(\theta) \geq LL(\theta^t) + (\theta - \theta^t)^T g(\theta^t) + \frac{1}{2}(\theta - \theta^t)^T \mathbf{B}(\theta - \theta^t) \quad (6.129)$$

26

27 where  $g(\theta^t) = \nabla LL(\theta^t)$ . Therefore the following function is a valid lower bound:

28

$$\underline{29} \quad Q(\theta, \theta^t) = \theta^T(g(\theta^t) - \mathbf{B}\theta^t) + \frac{1}{2}\theta^T \mathbf{B}\theta \quad (6.130)$$

30

31 The corresponding update becomes

32

$$\underline{33} \quad \theta^{t+1} = \theta^t - \mathbf{B}^{-1}g(\theta^t) \quad (6.131)$$

34 This is similar to a Newton update, except we use  $\mathbf{B}$ , which is a fixed matrix, rather than  $\mathbf{H}(\theta^t)$ ,  
35 which changes at each iteration. This can give us some of the advantages of second order methods at  
36 lower computational cost.

37 For example, let us fit a multi-class logistic regression model using MM. (We follow the presentation  
38 of [Kri+05], who also consider the more interesting case of *sparse* logistic regression.) The probability  
39 that example  $n$  belongs to class  $c \in \{1, \dots, C\}$  is given by

40

$$\underline{41} \quad p(y_n = c | \mathbf{x}_n, \mathbf{w}) = \frac{\exp(\mathbf{w}_c^T \mathbf{x}_n)}{\sum_{i=1}^C \exp(\mathbf{w}_i^T \mathbf{x}_n)} \quad (6.132)$$

42

43 Because of the normalization condition  $\sum_{c=1}^C p(y_n = c | \mathbf{x}_n, \mathbf{w}) = 1$ , we can set  $\mathbf{w}_C = \mathbf{0}$ . (For example,  
44 in binary logistic regression, where  $C = 2$ , we only learn a single weight vector.) Therefore the  
45 parameters  $\theta$  correspond to a weight matrix  $\mathbf{w}$  of size  $D(C - 1)$ , where  $\mathbf{x}_n \in \mathbb{R}^D$ .

46

If we let  $\mathbf{p}_n(\mathbf{w}) = [p(y_n = 1|\mathbf{x}_n, \mathbf{w}), \dots, p(y_n = C-1|\mathbf{x}_n, \mathbf{w})]$  and  $\mathbf{y}_n = [\mathbb{I}(y_n = 1), \dots, \mathbb{I}(y_n = C-1)]$ , we can write the log-likelihood as follows:

$$LL(\mathbf{w}) = \sum_{n=1}^N \left[ \sum_{c=1}^{C-1} y_{nc} \mathbf{w}_c^\top \mathbf{x}_n - \log \sum_{c=1}^C \exp(\mathbf{w}_c^\top \mathbf{x}_n) \right] \quad (6.133)$$

The gradient is given by the following:

$$\mathbf{g}(\mathbf{w}) = \sum_{n=1}^N (\mathbf{y}_n - \mathbf{p}_n(\mathbf{w})) \otimes \mathbf{x}_n \quad (6.134)$$

where  $\otimes$  denotes kronecker product (which, in this case, is just outer product of the two vectors). The Hessian is given by the following:

$$\mathbf{H}(\mathbf{w}) = - \sum_{n=1}^N (\text{diag}(\mathbf{p}_n(\mathbf{w})) - \mathbf{p}_n(\mathbf{w})\mathbf{p}_n(\mathbf{w})^\top) \otimes (\mathbf{x}_n \mathbf{x}_n^\top) \quad (6.135)$$

We can construct a lower bound on the Hessian, as shown in [Boh92]:

$$\mathbf{H}(\mathbf{w}) \succ -\frac{1}{2} [\mathbf{I} - \mathbf{1}\mathbf{1}^\top/C] \otimes \left( \sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^\top \right) \triangleq \mathbf{B} \quad (6.136)$$

where  $\mathbf{I}$  is a  $(C-1)$ -dimensional identity matrix, and  $\mathbf{1}$  is a  $(C-1)$ -dimensional vector of all 1s. In the binary case, this becomes

$$\mathbf{H}(\mathbf{w}) \succ -\frac{1}{2} \left( 1 - \frac{1}{2} \right) \left( \sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^\top \right) = -\frac{1}{4} \mathbf{X}^\top \mathbf{X} \quad (6.137)$$

This follows since  $p_n \leq 0.5$  so  $-(p_n - p_n^2) \geq -0.25$ .

We can use this lower bound to construct an MM algorithm to find the MLE. The update becomes

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \mathbf{B}^{-1} \mathbf{g}(\mathbf{w}^t) \quad (6.138)$$

For example, let us consider the binary case, so  $\mathbf{g}^t = \nabla LL(\mathbf{w}^t) = \mathbf{X}^\top (\mathbf{y} - \boldsymbol{\mu}^t)$ , where  $\boldsymbol{\mu}^t = [\mathbf{p}_n(\mathbf{w}^t), (1 - \mathbf{p}_n(\mathbf{w}^t))]_{n=1}^N$ . The update becomes

$$\mathbf{w}^{t+1} = \mathbf{w}^t - 4(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{g}^t \quad (6.139)$$

The above is faster (per step) than the IRLS (iteratively reweighted least squares) algorithm (i.e., Newton's method), which is the standard method for fitting GLMs. To see this, note that the Newton update has the form

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \mathbf{H}^{-1} \mathbf{g}(\mathbf{w}^t) = \mathbf{w}^t - (\mathbf{X}^\top \mathbf{S}^t \mathbf{X})^{-1} \mathbf{g}^t \quad (6.140)$$

where  $\mathbf{S}^t = \text{diag}(\boldsymbol{\mu}^t \odot (1 - \boldsymbol{\mu}^t))$ . We see that Equation (6.139) is faster to compute, since we can precompute the constant matrix  $(\mathbf{X}^\top \mathbf{X})^{-1}$ .

1 **6.7.3 The EM algorithm**

3 In this section, we discuss the **expectation maximization (EM)** algorithm [DLR77; MK97], which  
4 is an algorithm designed to compute the MLE or MAP parameter estimate for probability models  
5 that have **missing data** and/or **hidden variables**. It is a special case of an MM algorithm.

6 The basic idea behind EM is to alternate between estimating the hidden variables (or missing  
7 values) during the **E step** (expectation step), and then using the fully observed data to compute the  
8 MLE during the **M step** (maximization step). Of course, we need to iterate this process, since the  
9 expected values depend on the parameters, but the parameters depend on the expected values.

10 In Section 6.7.3.1, we show that EM is a **bound optimization** algorithm, which implies that this  
11 iterative procedure will converge to a local maximum of the log likelihood. The speed of convergence  
12 depends on the amount of missing data, which affects the tightness of the bound [XJ96; MD97;  
13 SRG03; KKS20].

14 We now describe the EM algorithm for a generic model. We let  $\mathbf{y}_n$  be the visible data for example  
15  $n$ , and  $\mathbf{z}_n$  be the hidden data.

17 **6.7.3.1 Lower bound**

19 The goal of EM is to maximize the log likelihood of the observed data:

$$\text{LL}(\boldsymbol{\theta}) = \sum_{n=1}^N \log p(\mathbf{y}_n | \boldsymbol{\theta}) = \sum_{n=1}^N \log \left[ \sum_{\mathbf{z}_n} p(\mathbf{y}_n, \mathbf{z}_n | \boldsymbol{\theta}) \right] \quad (6.141)$$

24 where  $\mathbf{y}_n$  are the visible variables and  $\mathbf{z}_n$  are the hidden variables. Unfortunately this is hard to  
25 optimize, since the log cannot be pushed inside the sum.

26 EM gets around this problem as follows. First, consider a set of arbitrary distributions  $q_n(\mathbf{z}_n)$  over  
27 each hidden variable  $\mathbf{z}_n$ . The observed data log likelihood can be written as follows:

$$\text{LL}(\boldsymbol{\theta}) = \sum_{n=1}^N \log \left[ \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \frac{p(\mathbf{y}_n, \mathbf{z}_n | \boldsymbol{\theta})}{q_n(\mathbf{z}_n)} \right] \quad (6.142)$$

32 Using Jensen's inequality, we can push the log (which is a concave function) inside the expectation  
33 to get the following lower bound on the log likelihood:

$$\text{LL}(\boldsymbol{\theta}) \geq \sum_n \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log \frac{p(\mathbf{y}_n, \mathbf{z}_n | \boldsymbol{\theta})}{q_n(\mathbf{z}_n)} \quad (6.143)$$

$$= \sum_n \underbrace{\mathbb{E}_{q_n} [\log p(\mathbf{y}_n, \mathbf{z}_n | \boldsymbol{\theta})]}_{\mathcal{L}(\boldsymbol{\theta}, q_n | \mathbf{y}_n)} + \mathbb{H}(q_n) \quad (6.144)$$

$$= \sum_n \mathcal{L}(\boldsymbol{\theta}, q_n | \mathbf{y}_n) \triangleq \mathcal{L}(\boldsymbol{\theta}, \{q_n\} | \mathcal{D}) \quad (6.145)$$

43 where  $\mathbb{H}(q)$  is the entropy of probability distribution  $q$ , and  $\mathcal{L}(\boldsymbol{\theta}, \{q_n\} | \mathcal{D})$  is called the **evidence**  
44 **lower bound** or **ELBO**, since it is a lower bound on the log marginal likelihood,  $\log p(\mathbf{y}_{1:N} | \boldsymbol{\theta})$ , also  
45 called the evidence. Optimizing this bound is the basis of variational inference, as we discuss in  
46 Section 10.1.

47

---

### 6.7.3.2 E step

We see that the lower bound is a sum of  $N$  terms, each of which has the following form:

$$\mathbb{L}(\boldsymbol{\theta}, q_n | \mathbf{y}_n) = \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log \frac{p(\mathbf{y}_n, \mathbf{z}_n | \boldsymbol{\theta})}{q_n(\mathbf{z}_n)} \quad (6.146)$$

$$= \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log \frac{p(\mathbf{z}_n | \mathbf{y}_n, \boldsymbol{\theta}) p(\mathbf{y}_n | \boldsymbol{\theta})}{q_n(\mathbf{z}_n)} \quad (6.147)$$

$$= \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log \frac{p(\mathbf{z}_n | \mathbf{y}_n, \boldsymbol{\theta})}{q_n(\mathbf{z}_n)} + \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log p(\mathbf{y}_n | \boldsymbol{\theta}) \quad (6.148)$$

$$= -D_{\text{KL}}(q_n(\mathbf{z}_n) \| p(\mathbf{z}_n | \mathbf{y}_n, \boldsymbol{\theta})) + \log p(\mathbf{y}_n | \boldsymbol{\theta}) \quad (6.149)$$

where  $D_{\text{KL}}(q \| p) \triangleq \sum_z q(z) \log \frac{q(z)}{p(z)}$  is the Kullback-Leibler divergence (or KL divergence for short) between probability distributions  $q$  and  $p$ . We discuss this in more detail in Section 5.1, but the key property we need here is that  $D_{\text{KL}}(q \| p) \geq 0$  and  $D_{\text{KL}}(q \| p) = 0$  iff  $q = p$ . Hence we can maximize the lower bound  $\mathbb{L}(\boldsymbol{\theta}, \{q_n\} | \mathcal{D})$  wrt  $\{q_n\}$  by setting each one to  $q_n^* = p(\mathbf{z}_n | \mathbf{y}_n, \boldsymbol{\theta})$ . This is called the **E step**. This ensures the ELBO is a tight lower bound:

$$\mathbb{L}(\boldsymbol{\theta}, \{q_n^*\} | \mathcal{D}) = \sum_n \log p(\mathbf{y}_n | \boldsymbol{\theta}) = LL(\boldsymbol{\theta} | \mathcal{D}) \quad (6.150)$$

To see how this connects to bound optimization, let us define

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t) = \mathbb{L}(\boldsymbol{\theta}, \{p(\mathbf{z}_n | \mathbf{y}_n; \boldsymbol{\theta}^t)\}) \quad (6.151)$$

Then we have  $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t) \leq LL(\boldsymbol{\theta})$  and  $Q(\boldsymbol{\theta}^t, \boldsymbol{\theta}^t) = LL(\boldsymbol{\theta}^t)$ , as required.

However, if we cannot compute the posteriors  $p(\mathbf{z}_n | \mathbf{y}_n; \boldsymbol{\theta}^t)$  exactly, we can still use an approximate distribution  $q(\mathbf{z}_n | \mathbf{y}_n; \boldsymbol{\theta}^t)$ ; this will yield a non-tight lower-bound on the log-likelihood. This generalized version of EM is known as variational EM [NH98b]. See Section 6.7.6.1 for details.

### 6.7.3.3 M step

In the M step, we need to maximize  $LL(\boldsymbol{\theta}, \{q_n^t\})$  wrt  $\boldsymbol{\theta}$ , where the  $q_n^t$  are the distributions computed in the E step at iteration  $t$ . Since the entropy terms  $\mathbb{H}(q_n)$  are constant wrt  $\boldsymbol{\theta}$ , so we can drop them in the M step. We are left with

$$LL^t(\boldsymbol{\theta}) = \sum_n \mathbb{E}_{q_n^t(\mathbf{z}_n)} [\log p(\mathbf{y}_n, \mathbf{z}_n | \boldsymbol{\theta})] \quad (6.152)$$

This is called the **expected complete data log likelihood**. If the joint probability is in the exponential family (Section 2.5), we can rewrite this as

$$LL^t(\boldsymbol{\theta}) = \sum_n \mathbb{E} [\mathcal{T}(\mathbf{y}_n, \mathbf{z}_n)^T \boldsymbol{\theta} - A(\boldsymbol{\theta})] = \sum_n (\mathbb{E} [\mathcal{T}(\mathbf{y}_n, \mathbf{z}_n)])^T \boldsymbol{\theta} - A(\boldsymbol{\theta}) \quad (6.153)$$

where  $\mathbb{E} [\mathcal{T}(\mathbf{y}_n, \mathbf{z}_n)]$  are called the **expected sufficient statistics**.

1 In the M step, we maximize the expected complete data log likelihood to get  
2

$$\underline{3} \quad \theta^{t+1} = \arg \max_{\theta} \sum_n \mathbb{E}_{q_n^t} [\log p(\mathbf{y}_n, \mathbf{z}_n | \theta)] \quad (6.154)$$

4

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

#### 14 6.7.4 Example: EM for an MVN with missing data

15 It is easy to compute the MLE for a multivariate normal when we have a fully observed data matrix:  
16 we just compute the sample mean and covariance. In this section, we consider the case where we have  
17 **missing data or partially observed data**. For example, we can think of the entries of  $\mathbf{Y}$  as being  
18 answers to a survey; some of these answers may be unknown. There are many kinds of missing data,  
19 as we discuss in Section 22.3.5. In this section, we make the missing at random (MAR) assumption,  
20 for simplicity. Under the MAR assumption, the log likelihood of the visible data has the form  
21

$$\underline{22} \quad \log p(\mathbf{X} | \theta) = \sum_n \log p(\mathbf{x}_n | \theta) = \sum_n \log \left[ \int p(\mathbf{x}_n, \mathbf{z}_n | \theta) d\mathbf{z}_n \right] \quad (6.155)$$

23

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

##### 29 6.7.4.1 E step

30 Suppose we have the parameters  $\theta^{t-1}$  from the previous iteration. Then we can compute the expected  
31 complete data log likelihood at iteration  $t$  as follows:

$$\underline{33} \quad Q(\theta, \theta^{t-1}) = \mathbb{E} \left[ \sum_{n=1}^N \log \mathcal{N}(\mathbf{y}_n | \mu, \Sigma) | \mathcal{D}, \theta^{t-1} \right] \quad (6.156)$$

34

$$\underline{35} \quad = -\frac{N}{2} \log |2\pi\Sigma| - \frac{1}{2} \sum_n \mathbb{E} [(\mathbf{y}_n - \mu)^T \Sigma^{-1} (\mathbf{y}_n - \mu)] \quad (6.157)$$

36

$$\underline{37} \quad = -\frac{N}{2} \log |2\pi\Sigma| - \frac{1}{2} \text{tr}(\Sigma^{-1} \sum_n \mathbb{E} [(\mathbf{y}_n - \mu)(\mathbf{y}_n - \mu)^T]) \quad (6.158)$$

38

$$\underline{39} \quad = -\frac{N}{2} \log |\Sigma| - \frac{ND}{2} \log(2\pi) - \frac{1}{2} \text{tr}(\Sigma^{-1} \mathbb{E} [\mathbf{S}(\mu)]) \quad (6.159)$$

40

41 where  
42

$$\underline{43} \quad \mathbb{E} [\mathbf{S}(\mu)] \triangleq \sum_n \left( \mathbb{E} [\mathbf{y}_n \mathbf{y}_n^T] + \mu \mu^T - 2\mu \mathbb{E} [\mathbf{y}_n]^T \right) \quad (6.160)$$

44

(We drop the conditioning of the expectation on  $\mathcal{D}$  and  $\boldsymbol{\theta}^{t-1}$  for brevity.) We see that we need to compute  $\sum_n \mathbb{E}[\mathbf{y}_n]$  and  $\sum_n \mathbb{E}[\mathbf{y}_n \mathbf{y}_n^\top]$ ; these are the expected sufficient statistics.

To compute these quantities, we use the results from Section 2.3.3. We have

$$p(\mathbf{z}_n | \mathbf{x}_n, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{z}_n | \mathbf{m}_n, \mathbf{V}_n) \quad (6.161)$$

$$\mathbf{m}_n \triangleq \boldsymbol{\mu}_h + \boldsymbol{\Sigma}_{hv} \boldsymbol{\Sigma}_{vv}^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_v) \quad (6.162)$$

$$\mathbf{V}_n \triangleq \boldsymbol{\Sigma}_{hh} - \boldsymbol{\Sigma}_{hv} \boldsymbol{\Sigma}_{vv}^{-1} \boldsymbol{\Sigma}_{vh} \quad (6.163)$$

where we partition  $\boldsymbol{\mu}$  and  $\boldsymbol{\Sigma}$  into blocks based on the hidden and visible indices  $h$  and  $v$ . Hence the expected sufficient statistics are

$$\mathbb{E}[\mathbf{y}_n] = (\mathbb{E}[\mathbf{z}_n]; \mathbf{x}_n) = (\mathbf{m}_n; \mathbf{x}_n) \quad (6.164)$$

To compute  $\mathbb{E}[\mathbf{y}_n \mathbf{y}_n^\top]$ , we use the result that  $\text{Cov}[\mathbf{y}] = \mathbb{E}[\mathbf{y}\mathbf{y}^\top] - \mathbb{E}[\mathbf{y}]\mathbb{E}[\mathbf{y}^\top]$ . Hence

$$\mathbb{E}[\mathbf{y}_n \mathbf{y}_n^\top] = \mathbb{E} \left[ \begin{pmatrix} \mathbf{z}_n \\ \mathbf{x}_n \end{pmatrix} \begin{pmatrix} \mathbf{z}_n^\top & \mathbf{x}_n^\top \end{pmatrix} \right] = \begin{pmatrix} \mathbb{E}[\mathbf{z}_n \mathbf{z}_n^\top] & \mathbb{E}[\mathbf{z}_n] \mathbf{x}_n^\top \\ \mathbf{x}_n \mathbb{E}[\mathbf{z}_n]^\top & \mathbf{x}_n \mathbf{x}_n^\top \end{pmatrix} \quad (6.165)$$

$$\mathbb{E}[\mathbf{z}_n \mathbf{z}_n^\top] = \mathbb{E}[\mathbf{z}_n] \mathbb{E}[\mathbf{z}_n]^\top + \mathbf{V}_n \quad (6.166)$$

#### 6.7.4.2 M step

By solving  $\nabla Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t-1)}) = \mathbf{0}$ , we can show that the M step is equivalent to plugging these ESS into the usual MLE equations to get

$$\boldsymbol{\mu}^t = \frac{1}{N} \sum_n \mathbb{E}[\mathbf{y}_n] \quad (6.167)$$

$$\boldsymbol{\Sigma}^t = \frac{1}{N} \sum_n \mathbb{E}[\mathbf{y}_n \mathbf{y}_n^\top] - \boldsymbol{\mu}^t (\boldsymbol{\mu}^t)^\top \quad (6.168)$$

Thus we see that EM is *not* equivalent to simply replacing variables by their expectations and applying the standard MLE formula; that would ignore the posterior variance and would result in an incorrect estimate. Instead we must compute the expectation of the sufficient statistics, and plug that into the usual equation for the MLE.

#### 6.7.4.3 Initialization

To get the algorithm started, we can compute the MLE based on those rows of the data matrix that are fully observed. If there are no such rows, we can just estimate the diagonal terms of  $\boldsymbol{\Sigma}$  using the observed marginal statistics. We are then ready to start EM.

#### 6.7.4.4 Example

As an example of this procedure in action, let us consider an imputation problem, where we have  $N = 100$  10-dimensional data cases, which we assume to come from a Gaussian. We generate synthetic data where 50% of the observations are missing at random. First we fit the parameters

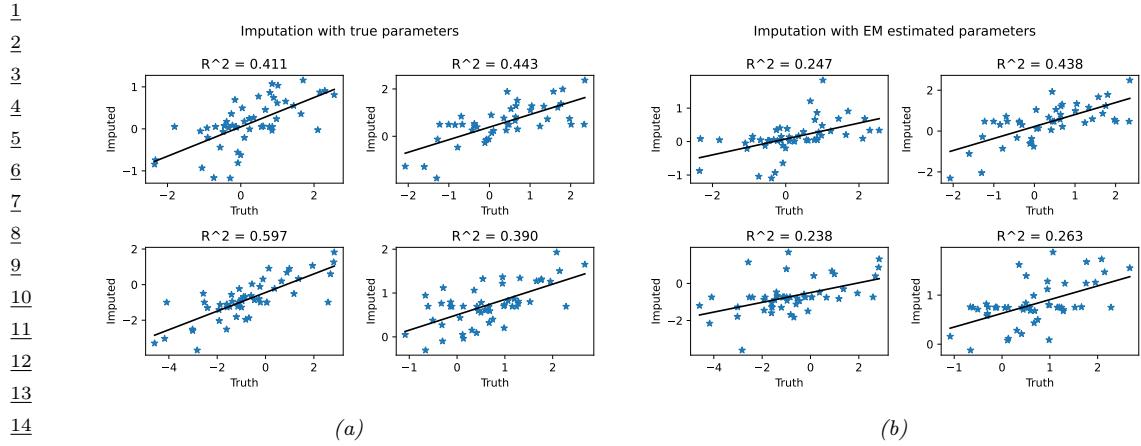


Figure 6.8: Illustration of data imputation using a multivariate Gaussian. (a) Scatter plot of true values vs imputed values using true parameters. (b) Same as (a), but using parameters estimated with EM. We just show the first four variables, for brevity. Generated by `gauss_imputation_em_demo.py`.

using EM. Call the resulting parameters  $\hat{\theta}$ . We can now use our model for predictions by computing  $\mathbb{E} [z_n | \mathbf{x}_n, \hat{\theta}]$ . Figure 6.8 indicates that the results obtained using the learned parameters are almost as good as with the true parameters. Not surprisingly, performance improves with more data, or as the fraction of missing data is reduced.

### 6.7.5 Example: robust linear regression using Student- $t$ likelihood

In this section, we discuss how to use EM to fit a linear regression model that uses the Student distribution for its likelihood, instead of the more common Gaussian distribution, in order to achieve robustness, as first proposed in [Zel76]. More precisely, the likelihood is given by

$$p(y|\mathbf{x}, \mathbf{w}, \sigma^2, \nu) = \mathcal{T}(y|\mathbf{w}^\top \mathbf{x}, \sigma^2, \nu) \quad (6.169)$$

At first blush it may not be apparent how to do this, since there is no missing data, and there are no hidden variables. However, it turns out that we can introduce ‘‘artificial’’ hidden variables to make the problem easier to solve; this is a common trick. The key insight is that we can represent the Student distribution as a Gaussian scale mixture, as we discussed in Section 29.2.3.1.

We can apply the GSM version of the Student distribution to our problem by associating a latent scale  $z_n \in \mathbb{R}_+$  with each example. The complete data log likelihood is therefore given by

$$\log p(\mathbf{y}, \mathbf{z} | \mathbf{X}, \mathbf{w}, \sigma^2, \nu) = \sum_n -\frac{1}{2} \log(2\pi z_n \sigma^2) - \frac{1}{2z_n \sigma^2} (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 \quad (6.170)$$

$$+ \left(\frac{\nu}{2} - 1\right) \log(z_n) - z_n \frac{\nu}{2} + \text{const} \quad (6.171)$$

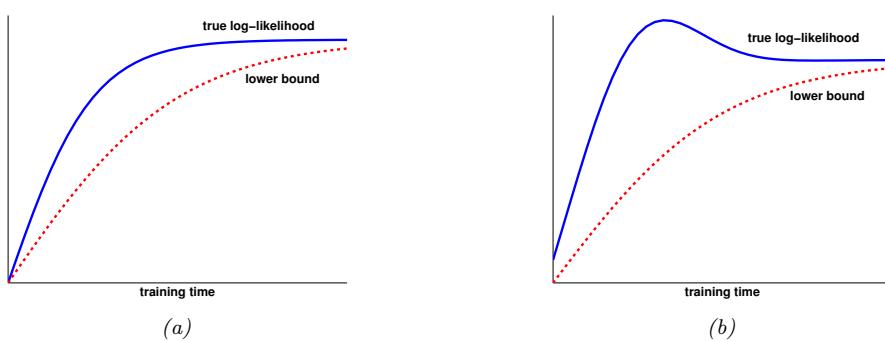


Figure 6.9: Illustration of possible behaviors of variational EM. (a) The lower bound increases at each iteration, and so does the likelihood. (b) The lower bound increases but the likelihood decreases. In this case, the algorithm is closing the gap between the approximate and true posterior. This can have a regularizing effect. Adapted from Figure 6 of [SJJ96]. Generated by `var_em_bound.py`.

Ignoring terms not involving  $\mathbf{w}$ , and taking expectations, we have

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t) = - \sum_n \frac{\lambda_n^t}{2\sigma^2} (y_n - \mathbf{w}^T \mathbf{x}_n)^2 \quad (6.172)$$

where  $\lambda_n^t \triangleq \mathbb{E}[1/z_n | y_n, \mathbf{x}_n, \mathbf{w}^t]$ . We recognize this as a weighted least squares objective, with weight  $\lambda_n^t$  per data point .

We now discuss how to compute these weights. Using the results from Section 2.2.2.8, one can show that

$$p(z_n | y_n, \mathbf{x}_n, \boldsymbol{\theta}) = \text{IG}\left(\frac{\nu + 1}{2}, \frac{\nu + \delta_n}{2}\right) \quad (6.173)$$

where  $\delta_n = \frac{(y_n - \mathbf{x}^T \mathbf{x}_n)^2}{\sigma^2}$  is the standardized residual. Hence

$$\lambda_n = \mathbb{E}[1/z_n] = \frac{\nu^t + 1}{\nu^t + \delta_n^t} \quad (6.174)$$

So if the residual  $\delta_n^t$  is large, the point will be given low weight  $\lambda_n^t$ , which makes intuitive sense, since it is probably an outlier.

### 6.7.6 Extensions to EM

There are many variations and extensions of the EM algorithm, as discussed in [MK97]. We summarize a few of these below.

#### 6.7.6.1 Variational EM

Suppose in the E step we pick  $q_n^* = \operatorname{argmin}_{q_n \in \mathcal{Q}} D_{\text{KL}}(q_n \| p(\mathbf{z}_n | \mathbf{x}_n, \boldsymbol{\theta}))$ . Because we are optimizing over the space of functions, this is called variational inference (see Section 10.1 for details). If the

1 family of distributions  $\mathcal{Q}$  is rich enough to contain the true posterior,  $q_n = p(\mathbf{z}_n | \mathbf{x}_n, \boldsymbol{\theta})$ , then we can  
2 make the KL be zero. But in general, we might choose a more restrictive class for computational  
3 reasons. For example, we might use  $q_n(\mathbf{z}_n) = \mathcal{N}(\mathbf{z}_n | \boldsymbol{\mu}_n, \text{diag}(\boldsymbol{\sigma}_n))$  even if the true posterior is  
4 correlated.  
5

6 The use of an approximate  $q$  distribution inside the E step of EM is called **variational EM**  
7 [NH98a]. Unlike regular EM, variational EM is not guaranteed to increase the actual log likelihood  
8 itself (see Figure 6.9), but it does monotonically increase the variational lower bound. We can control  
9 the tightness of this lower bound by varying the variational family  $\mathcal{Q}$ ; in the limit in which  $q_n = p_n$ ,  
10 corresponding to exact inference, we recover the same behavior as regular EM. See Section 10.2.5 for  
11 further discussion.

12

### 13 6.7.6.2 Hard EM

14 Suppose we use a degenerate posterior approximation in the context of variational EM, corresponding  
15 to a point estimate,  $q(\mathbf{z} | \mathbf{x}_n) = \delta_{\hat{\mathbf{z}}_n}(\mathbf{z})$ , where  $\hat{\mathbf{z}}_n = \text{argmax}_{\mathbf{z}} p(\mathbf{z} | \mathbf{x}_n)$ . This is equivalent to **hard**  
16 **EM**, where we ignore uncertainty about  $\mathbf{z}_n$  in the E step.  
17

18 The problem with this degenerate approach is that it is very prone to overfitting, since the number  
19 of latent variables is proportional to the number of datacases [WCS08].

20

### 21 6.7.6.3 Monte Carlo EM

22 Another approach to handling an intractable E step is to use a Monte Carlo approximation to the  
23 expected sufficient statistics. That is, we draw samples from the posterior,  $\mathbf{z}_n^s \sim p(\mathbf{z}_n | \mathbf{x}_n, \boldsymbol{\theta}^t)$ , and  
24 then compute the sufficient statistics for each completed vector,  $(\mathbf{x}_n, \mathbf{z}_n^s)$ , and then average the  
25 results. This is called **Monte Carlo EM** or **MCEM** [WT90; Nea12].

26 One way to draw samples is to use MCMC (see Chapter 12). However, if we have to wait for  
27 MCMC to converge inside each E step, the method becomes very slow. An alternative is to use  
28 stochastic approximation, and only perform “brief” sampling in the E step, followed by a partial  
29 parameter update. This is called **stochastic approximation EM** [DLM99] and tends to work  
30 better than MCEM.  
31

### 32 6.7.6.4 Generalized EM

33 Sometimes we can perform the E step exactly, but we cannot perform the M step exactly. However,  
34 we can still monotonically increase the log likelihood by performing a “partial” M step, in which we  
35 merely increase the expected complete data log likelihood, rather than maximizing it. For example,  
36 we might follow a few gradient steps. This is called the **generalized EM** or **GEM** algorithm.  
37  
38

### 39 6.7.6.5 ECM algorithm

40 The **ECM** algorithm stands for “expectation conditional maximization”, and refers to optimizing the  
41 parameters in the M step sequentially, if they turn out to be dependent. The **ECME** algorithm,  
42 which stands for “ECM either” [LR95], is a variant of ECM in which we maximize the expected  
43 complete data log likelihood (the  $Q$  function) as usual, or the observed data log likelihood, during  
44 one or more of the conditional maximization steps. The latter can be much faster, since it ignores  
45 the results of the E step, and directly optimizes the objective of interest. A standard example of  
46 the results of the E step, and directly optimizes the objective of interest. A standard example of  
47

this is when fitting the Student T distribution. For fixed  $\nu$ , we can update  $\Sigma$  as usual, but then to update  $\nu$ , we replace the standard update of the form  $\nu^{t+1} = \arg \max_{\nu} Q((\mu^{t+1}, \Sigma^{t+1}, \nu), \theta^t)$  with  $\nu^{t+1} = \arg \max_{\nu} \log p(\mathcal{D} | \mu^{t+1}, \Sigma^{t+1}, \nu)$ . See [MK97] for more information.

### 6.7.6.6 Online EM

When dealing with large or streaming datasets, it is important to be able to learn online, as we discussed in Section 20.7.5. There are two main approaches to **online EM** in the literature. The first approach, known as **incremental EM** [NH98a], optimizes the lower bound  $Q(\theta, q_1, \dots, q_N)$  one  $q_n$  at a time; however, this requires storing the expected sufficient statistics for each data case.

The second approach, known as **stepwise EM** [SI00; LK09; CM09], is based on stochastic gradient descent. This optimizes a local upper bound on  $LL_n(\theta) = \log p(x_n | \theta)$  at each step. (See [Mai13; Mai15] for a more general discussion of stochastic and incremental bound optimization algorithms.)

## 6.8 The Bayesian learning rule

in this section, we discuss the “**Bayesian learning rule**” [KR21a], which provides a unified framework for deriving many standard (and non-standard) optimization and inference algorithms used in the ML community.

To motivate the BLR, recall the standard **empirical risk minimization** or **ERM** problem, which has the form  $\theta_* = \operatorname{argmin}_{\theta} \bar{\ell}(\theta)$ , where

$$\bar{\ell}(\theta) = \sum_{n=1}^N \ell(y_n, f_{\theta}(x_n)) + R(\theta) \quad (6.175)$$

where  $f_{\theta}(x)$  is a prediction function,  $\ell(y, \hat{y})$  is a loss function, and  $R(\theta)$  is some kind of regularizer.

Although the regularizer can prevent overfitting, the ERM method can still result in parameter estimates that are not robust. A better approach is to fit a *distribution* over possible parameter values,  $q(\theta)$ . If we minimize the expected loss, we will find parameter settings that will work well even if they are slightly perturbed, as illustrated in Figure 6.10, which helps with robustness and generalization. Of course, if the distribution  $q$  collapses to a single delta function, we will end up with the ERM solution. To prevent this, we add a penalty term, that measures the KL divergence from  $q(\theta)$  to some prior  $\pi_0(\theta) \propto \exp(R(\theta))$ . This gives rise to the following BLR objective:

$$\mathcal{L}(q) = \mathbb{E}_{q(\theta)} \left[ \sum_{n=1}^N \ell(y_n, f_{\theta}(x_n)) \right] + D_{\text{KL}}(q(\theta) \| \pi_0(\theta)) \quad (6.176)$$

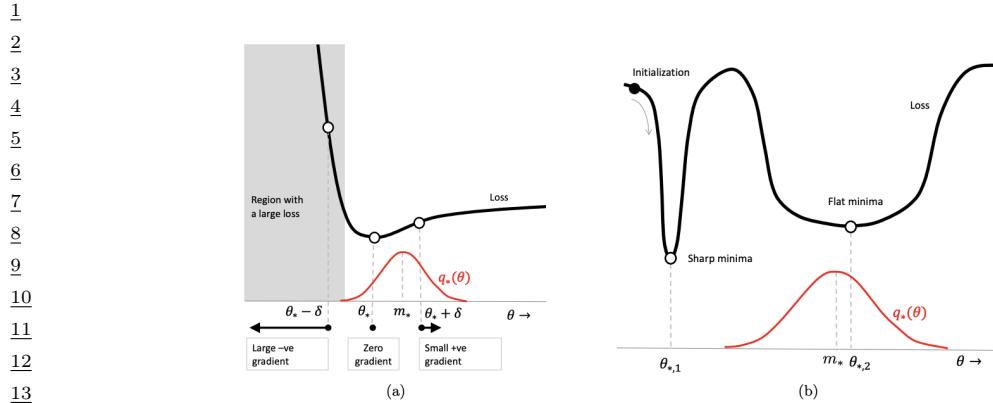
We can rewrite the KL term as

$$D_{\text{KL}}(q(\theta) \| \pi_0(\theta)) = \mathbb{E}_{q(\theta)} [R(\theta)] + \mathbb{H}(q(\theta)) \quad (6.177)$$

and hence can rewrite the BLR objective as follows:

$$\mathcal{L}(q) = \mathbb{E}_{q(\theta)} [\bar{\ell}(\theta)] - \mathbb{H}(q(\theta)) \quad (6.178)$$

Below we show that different approximations to this objective recover a variety of different methods in the literature.



*Figure 6.10: Illustration of the robustness obtained by using a Bayesian approach to parameter estimation.*  
 (a) When the minimum  $\theta_*$  lies next to a “wall”, the Bayesian solution shifts away from the boundary to avoid large losses due to perturbations of the parameters. (b) The Bayesian solution prefers flat minima over sharp minima, to avoid large losses due to perturbations of the parameters. From Figure 1 of [KR21a]. Used with kind permission of Emtiyaz Khan.

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.)

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 Let  $\mathcal{D} = \{(x_n, y_n) : n = 1 : N\}$  be the data we condition on. The Bayesian posterior can be written  
 32 as

$$33 \quad p(\theta|\mathcal{D}) = \frac{1}{Z(\mathcal{D})} \pi_0(\theta) \prod_{n=1}^N p(y_n|f_{\theta}(x_n)) \quad (6.180)$$

37 This can be derived by minimizing the BLR, since

$$39 \quad \mathcal{L}(q) = -\mathbb{E}_{q(\theta)} \left[ \sum_{n=1}^N \log p(y_n|f_{\theta}(x_n)) \right] + D_{\text{KL}}(q(\theta)\|\pi_0(\theta)) \quad (6.181)$$

$$42 \quad = \mathbb{E}_{q(\theta)} \left[ \log \frac{q(\theta)}{\frac{\pi_0(\theta)}{Z(\mathcal{D})} \prod_{n=1}^N p(y_n|f_{\theta}(x_n))} \right] - \log Z(\mathcal{D}) \quad (6.182)$$

$$45 \quad = D_{\text{KL}}(q(\theta)\|p(\theta|\mathcal{D})) - \log Z(\mathcal{D}) \quad (6.183)$$

47

Since  $Z(\mathcal{D})$  is a constant, we can minimize the loss by by setting  $q(\boldsymbol{\theta}) = p(\boldsymbol{\theta}|\mathcal{D})$ .

Of course, we can use other kinds of loss, not just log likelihoods. This results in a framework known as **generalized Bayesian inference** [BHW16; KJD19; KJD21]. See Section 14.1.3 for more discussion.

### 6.8.1.2 Optimization of BLR using natural gradient descent

In general, we cannot compute the exact posterior  $q(\boldsymbol{\theta}) = p(\boldsymbol{\theta}|\mathcal{D})$ , so we seek an approximation. We will assume that  $q(\boldsymbol{\theta})$  is an exponential family distribution, such as a multivariate Gaussian, where the mean represents the standard point estimate of  $\boldsymbol{\theta}$  (as in ERM), and the covariance represents our uncertainty (as in Bayes). Hence  $q$  can be written as follows:

$$q(\boldsymbol{\theta}) = h(\boldsymbol{\theta}) \exp[\boldsymbol{\lambda}^\top \mathcal{T}(\boldsymbol{\theta}) - A(\boldsymbol{\lambda})] \quad (6.184)$$

where  $\boldsymbol{\lambda}$  are the natural parameters,  $\mathcal{T}(\boldsymbol{\theta})$  are the sufficient statistics,  $A(\boldsymbol{\lambda})$  is the log partition function, and  $h(\boldsymbol{\theta})$  is the base measure, which is usually a constant. The BLR loss becomes

$$\mathcal{L}(\boldsymbol{\lambda}) = \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] - \mathbb{H}(q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})) \quad (6.185)$$

We can optimize this using natural gradient descent (Section 6.4). The update becomes

$$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t - \eta_t \tilde{\nabla}_{\boldsymbol{\lambda}} \left[ \mathbb{E}_{q_{\boldsymbol{\lambda}_t}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] - \mathbb{H}(q_{\boldsymbol{\lambda}_t}) \right] \quad (6.186)$$

where  $\tilde{\nabla}_{\boldsymbol{\lambda}}$  denotes the natural gradient. We discuss how to compute these natural gradients in Section 6.4.5. In particular, we can convert it to regular gradients wrt the moment parameters  $\boldsymbol{\mu}_t = \boldsymbol{\mu}(\boldsymbol{\lambda}_t)$ . This gives

$$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t - \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}_t}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] + \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{H}(q_{\boldsymbol{\mu}_t}) \quad (6.187)$$

From Equation (6.84) we have

$$\nabla_{\boldsymbol{\mu}} \mathbb{H}(q) = -\boldsymbol{\lambda} - \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\log h(\boldsymbol{\theta})] \quad (6.188)$$

Hence the update becomes

$$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t - \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}_t}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] - \eta_t \boldsymbol{\lambda}_t - \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\log h(\boldsymbol{\theta})] \quad (6.189)$$

$$= (1 - \eta_t) \boldsymbol{\lambda}_t - \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta}) + \log h(\boldsymbol{\theta})] \quad (6.190)$$

For distributions  $q$  with constant base measure  $h(\boldsymbol{\theta})$ , this simplifies to

$$\boldsymbol{\lambda}_{t+1} = (1 - \eta_t) \boldsymbol{\lambda}_t - \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] \quad (6.191)$$

Hence at the fixed point we have

$$\boldsymbol{\lambda}_* = (1 - \eta) \boldsymbol{\lambda}_* - \eta \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] \quad (6.192)$$

$$\boldsymbol{\lambda}_* = \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [-\bar{\ell}(\boldsymbol{\theta})] = \tilde{\nabla}_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})} [-\bar{\ell}(\boldsymbol{\theta})] \quad (6.193)$$

1 **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 ?? from the  
4 BLR by using the fixed point condition in Equation (6.193) to write  
5

$$\underline{6} \quad \lambda_* = \nabla_{\mu} \mathbb{E}_{q_*} [-\bar{\ell}(\theta)] = \lambda_0 + \sum_{i=1}^N \underbrace{\nabla_{\mu} \mathbb{E}_{q_*} [\log p(y_i|\theta)]}_{\tilde{\lambda}_i(y_i)} \quad (6.194)$$

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

$$\underline{26} \quad \bar{\ell}(\theta) = \sum_{n=1}^N \ell(y_n, f_{\theta}(x_n)) + R(\theta) \quad (6.195)$$

29 we will instead minimize  
30

$$\underline{31} \quad \mathcal{L}(\lambda) = \mathbb{E}_{q(\theta|\lambda)} [\bar{\ell}(\theta)] - \mathbb{H}(q(\theta|\lambda)) \quad (6.196)$$

33 Below we show that different approximations to this objective recover a variety of different optimization  
34 methods that are used in the literature. (We discuss such algorithms in more detail in Chapter 6.)  
35

36 **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

$$\underline{42} \quad 2 \log h(\theta) = -D \log(2\pi) - \theta^T \theta \quad (6.197)$$

44 Hence  
45

$$\underline{46} \quad \nabla_{\mu} \mathbb{E}_q [\log h(\theta)] = \nabla_{\mu} (-D \log(2\pi) - \mu^T \mu - D) = -\mu = -\lambda = -m \quad (6.198)$$

Thus the BLR update becomes

$$\mathbf{m}_{t+1} = (1 - \eta_t) \mathbf{m}_t + \eta_t \mathbf{m}_t - \eta_t \nabla_{\boldsymbol{\theta}} \mathbb{E}_{q_{\boldsymbol{m}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] \quad (6.199)$$

We can remove the expectation using the first order delta method (Section 6.6.5):

$$\nabla_{\boldsymbol{m}} \mathbb{E}_{q_{\boldsymbol{m}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] \approx \nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}} \quad (6.200)$$

Putting these together gives the gradient descent update:

$$\mathbf{m}_{t+1} = \mathbf{m}_t - \eta_t \nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}_t} \quad (6.201)$$

### 6.8.2.2 Newton's method

In this section, we show how to derive Newton's second order optimization method as a special case of BLR, as first shown in [Kha+18].

Suppose we assume  $q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{m}, \mathbf{S}^{-1})$ . The natural parameters are

$$\boldsymbol{\lambda}^{(1)} = \mathbf{S}\mathbf{m}, \quad \boldsymbol{\lambda}^{(2)} = -\frac{1}{2}\mathbf{S} \quad (6.202)$$

The mean (moment) parameters are

$$\boldsymbol{\mu}^{(1)} = \mathbf{m}, \quad \boldsymbol{\mu}^{(2)} = \mathbf{S}^{-1} + \mathbf{m}\mathbf{m}^T \quad (6.203)$$

Since the base measure is constant, from Equation (6.191) we have

$$\mathbf{S}_{t+1} \mathbf{m}_{t+1} = (1 - \eta_t) \mathbf{S}_t \mathbf{m}_t - \eta_t \nabla_{\boldsymbol{\mu}^{(1)}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] \quad (6.204)$$

$$\mathbf{S}_{t+1} = (1 - \eta_t) \mathbf{S}_t + 2\eta_t \nabla_{\boldsymbol{\mu}^{(2)}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] \quad (6.205)$$

In Section 6.4.5.1 we show that

$$\nabla_{\boldsymbol{\mu}^{(1)}} \mathbb{E}_{q(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] = \mathbb{E}_{q(\boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})] - \mathbb{E}_{q(\boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta})] \mathbf{m} \quad (6.206)$$

$$\nabla_{\boldsymbol{\mu}^{(2)}} \mathbb{E}_{q(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] = \frac{1}{2} \mathbb{E}_{q(\boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta})] \quad (6.207)$$

Hence the update for the precision matrix becomes

$$\mathbf{S}_{t+1} = (1 - \eta_t) \mathbf{S}_t + \eta_t \mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta})] \quad (6.208)$$

For the precision weighted mean, we have

$$\mathbf{S}_{t+1} \mathbf{m}_{t+1} = (1 - \eta_t) \mathbf{S}_t \mathbf{m}_t - \eta_t \mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})] + \eta_t \mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta})] \mathbf{m}_t \quad (6.209)$$

$$= \mathbf{S}_{t+1} \mathbf{m}_t - \eta_t \mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})] \quad (6.210)$$

Hence

$$\mathbf{m}_{t+1} = \mathbf{m}_t - \eta_t \mathbf{S}_{t+1}^{-1} \mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})] \quad (6.211)$$

We can recover Newton's method in three steps. First set the learning rate to  $\eta_t = 1$ , based on an assumption that the objective is convex. Second, treat the iterate as  $\mathbf{m}_t = \boldsymbol{\theta}_t$ . Third, apply the delta method to get

$$\mathbf{S}_{t+1} = \mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta})] \approx \nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}_t} \quad (6.212)$$

and

$$\mathbb{E}_q [\nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})] \approx \nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}_t} \quad (6.213)$$

This gives Newton's update:

$$\mathbf{m}_{t+1} = \mathbf{m}_t - [\nabla_{\mathbf{m}}^2 \bar{\ell}(\mathbf{m}_t)]^{-1} [\nabla_{\mathbf{m}} \bar{\ell}(\mathbf{m}_t)] \quad (6.214)$$

### 6.8.2.3 Variational online Gauss-Newton

In this section, we describe the **Variational Online Gauss-Newton** or **VOGN** method of [Kha+18]. This is an approximate second order optimization method that can be derived from the BLR in several steps, as we show below.

First, we use a diagonal Gaussian approximation to the posterior,  $q_t(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}_t, \mathbf{S}_t^{-1})$ , where  $\mathbf{S}_t = \text{diag}(\mathbf{s}_t)$  is a vector of precisions. Following Section 6.8.2.2, we get the following updates:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \frac{1}{\mathbf{s}_{t+1}} \odot \mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})] \quad (6.215)$$

$$\mathbf{s}_{t+1} = (1 - \eta_t) \mathbf{s}_t + \eta_t \mathbb{E}_{q_t} [\text{diag}(\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta}))] \quad (6.216)$$

where  $\odot$  is elementwise multiplication, and the division by  $\mathbf{s}_{t+1}$  is also elementwise.

Second, we use the delta approximation to replace expectations by plugging in the mean. Third we use a minibatch approximation to the gradient and diagonal Hessian:

$$\hat{\nabla}_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta}) = \frac{N}{M} \sum_{i \in \mathcal{M}} \nabla_{\boldsymbol{\theta}} \ell(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i)) + \nabla_{\boldsymbol{\theta}} R(\boldsymbol{\theta}) \quad (6.217)$$

$$\hat{\nabla}_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta}) = \frac{N}{M} \sum_{i \in \mathcal{M}} \nabla_{\boldsymbol{\theta}}^2 \ell(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i)) + \nabla_{\boldsymbol{\theta}}^2 R(\boldsymbol{\theta}) \quad (6.218)$$

where  $M$  is the minibatch size.

For some non-convex problems, such as DNNs, the Hessian may be not be positive definite, so we can get better results using a Gauss-Newton approximation, based on the squared gradients instead of the Hessian:

$$\hat{\nabla}_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta}) \approx \frac{N}{M} \sum_{i \in \mathcal{M}} [\nabla_{\boldsymbol{\theta}} \ell(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i))]^2 + \nabla_{\boldsymbol{\theta}}^2 R(\boldsymbol{\theta}) \quad (6.219)$$

This is also faster to compute.

Putting all this together gives rise to the **Online Gauss-Newton** or **OGN** method of [Osa+19a].

If we drop the delta approximation, and work with expectations. we get the **Variational Online Gauss-Newton** or **VOGN** method of [Kha+18]. We can approximate the expectations by sampling. In particular, VOGN uses the following **weight perturbation** method

$$\mathbb{E}_{q_t} \left[ \hat{\nabla}_{\theta} \bar{\ell}(\theta) \right] \approx \hat{\nabla}_{\theta} \bar{\ell}(\theta_t + \epsilon_t) \quad (6.220)$$

where  $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \text{diag}(s_t))$ . It also also possible to approximate the Fisher information matrix directly; this results in the **Variational Online Generalized Gauss-Newton** or **VOGNN** method of [Osa+19a].

#### 6.8.2.4 Adaptive learning rate SGD

In this section, we show how to derive an update rule which is very similar to the **RMSprop** [Hin14] method, which is widely used in deep learning. The approach we take is similar to that VOGN in Section 6.8.2.3. We use the same diagonal Gaussian approximation,  $q_t(\theta) = \mathcal{N}(\theta | \theta_t, S_t^{-1})$ , where  $S_t = \text{diag}(s_t)$  is a vector of precisions. We then use the delta method to eliminate expectations:

$$\theta_{t+1} = \theta_t - \eta_t \frac{1}{s_{t+1}} \odot \nabla_{\theta} \bar{\ell}(\theta_t) \quad (6.221)$$

$$s_{t+1} = (1 - \eta_t)s_t + \eta_t \text{diag}(\nabla_{\theta}^2 \bar{\ell}(\theta_t)) \quad (6.222)$$

where  $\odot$  is elementwise multiplication. If we allow for different learning rates we get

$$\theta_{t+1} = \theta_t - \alpha_t \frac{1}{s_{t+1}} \odot \nabla_{\theta} \bar{\ell}(\theta_t) \quad (6.223)$$

$$s_{t+1} = (1 - \beta_t)s_t + \beta_t \text{diag}(\hat{\nabla}_{\theta}^2 \bar{\ell}(\theta_t)) \quad (6.224)$$

Now suppose we replace the diagonal Hessian approximation with the sum of the squares per-sample gradients:

$$\text{diag}(\nabla_{\theta}^2 \bar{\ell}(\theta_t)) \approx \hat{\nabla} \bar{\ell}(\theta_t) \odot \hat{\nabla} \bar{\ell}(\theta_t) \quad (6.225)$$

If we also change some scaling factors we can get the RMSprop updates:

$$\theta_{t+1} = \theta_t - \alpha \frac{1}{\sqrt{v_{t+1}} + c\mathbf{1}} \odot \hat{\nabla}_{\theta} \bar{\ell}(\theta_t) \quad (6.226)$$

$$v_{t+1} = (1 - \beta)v_t + \beta[\hat{\nabla} \bar{\ell}(\theta_t) \odot \hat{\nabla} \bar{\ell}(\theta_t)] \quad (6.227)$$

This allows us to use standard deep learning optimizers to get a Gaussian approximation to the posterior for the parameters [Osa+19a].

It is also possible to derive the Adam optimizer [KB15] from BLR by adding a momentum term to RMSprop. See [KR21a; Ait18] for details.

#### 6.8.3 Variational optimization

Consider an objective defined in terms of discrete variables. Such objectives are not differentiable and so are hard to optimize. One advantage of BLR is that it optimizes the parameters of a probability

<sup>1</sup> distribution, and such expected loss objectives are usually differentiable and smooth. This is called  
<sup>2</sup> “**variational optimization**” [Bar17], since we are optimizing over a probability distribution.  
<sup>3</sup>

<sup>4</sup> For example, consider the case of a **binary neural network** where  $\theta_d \in \{0, 1\}$  indicates if  
<sup>5</sup> weight  $d$  is used or not, we can optimize over the parameters of a Bernoulli distribution,  $q(\boldsymbol{\theta}|\boldsymbol{\lambda}) =$   
<sup>6</sup>  $\prod_{d=1}^D \text{Ber}(\theta_d|p_d)$ , 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  
<sup>7</sup> the **BayesBiNN** approach [MBK20].

<sup>8</sup> If we ignore the entropy and regularizer term, we get the following simplified objective:

$$\mathcal{L}(\boldsymbol{\lambda}) = \int \bar{\ell}(\boldsymbol{\theta}) q(\boldsymbol{\theta}|\boldsymbol{\lambda}) d\boldsymbol{\theta} \quad (6.228)$$

<sup>12</sup> This method has various names: **stochastic relaxation** [SB12; SB13; MMP13], **stochastic ap-**  
<sup>13</sup> **proximation** [HHC12; Hu+12], etc. It is closely related to **evolutionary strategies**, which we  
<sup>14</sup> discuss in the supplementary material.

<sup>15</sup> In the case of functions with continuous domains, we can use a Gaussian for  $q(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ . The  
<sup>16</sup> resulting integral in Equation (6.228) can then sometimes be solved in closed form, as explained in  
<sup>17</sup> [Mob16]. By starting with a broad variance, and gradually reducing it, we hope the method can  
<sup>18</sup> avoid poor local optima, similar to simulated annealing (see supplementary material). However, we  
<sup>19</sup> generally get better results by including the entropy term, because then we can automatically learn  
<sup>20</sup> to adapt the variance. In addition, we can often work with natural gradients, which results in faster  
<sup>21</sup> convergence.

## <sup>24</sup> 6.9 Bayesian optimization

<sup>25</sup> In this section, we discuss **Bayesian optimization** or **BayesOpt**, which is a model-based approach  
<sup>26</sup> to black-box optimization, designed for the case where the objective function  $f : \mathcal{X} \rightarrow \mathbb{R}$  is expensive  
<sup>27</sup> to evaluate (e.g., if it requires running a simulation, or training and testing a particular neural net  
<sup>28</sup> architecture).

<sup>30</sup> Since the true function  $f$  is expensive to evaluate, we want to make as few function calls (i.e., make as  
<sup>31</sup> few **queries**  $\mathbf{x}$  to the **oracle**  $f$ ) as possible. This suggests that we should build a **surrogate function**  
<sup>32</sup> (also called a **response surface model**) based on the data collected so far,  $\mathcal{D}_n = \{(\mathbf{x}_i, y_i) : i = 1 : n\}$ ,  
<sup>33</sup> which we can use to decide which point to query next. There is an inherent tradeoff between picking  
<sup>34</sup> the point  $\mathbf{x}$  where we think  $f(\mathbf{x})$  is large (we follow the convention in the literature and assume  
<sup>35</sup> we are trying to maximize  $f$ ), and picking points where we are uncertain about  $f(\mathbf{x})$  but where  
<sup>36</sup> observing the function value might help us improve the surrogate model. This is another instance of  
<sup>37</sup> the exploration-exploitation dilemma.

<sup>38</sup> In the special case where the domain we are optimizing over is finite, so  $\mathcal{X} = \{1, \dots, A\}$ , the  
<sup>39</sup> BayesOpt problem becomes similar to the **best arm identification** problem in the bandit literature  
<sup>40</sup> (Section 36.4). An important difference is that in bandits, we care about the cost of every action we  
<sup>41</sup> take, whereas in optimization, we usually only care about the cost of the final solution we find. In  
<sup>42</sup> other words, in bandits, we want to minimize cumulative regret, whereas in optimization we want to  
<sup>43</sup> minimize simple or final regret.

<sup>44</sup> Another related topic is **active learning**. Here the goal is to identify the whole function  $f$  with  
<sup>45</sup> as few queries as possible, whereas in BayesOpt, the goal is just to identify the maximum of the  
<sup>46</sup> function.

<sup>47</sup>

Bayesian optimization is a large topic, and we only give a brief overview below. For more details, see e.g., [Sha+16; Fra18; Gar22]. (See also <https://distill.pub/2020/bayesian-optimization/> for an interactive tutorial.)

### 6.9.1 Sequential model-based optimization

BayesOpt is an instance of a strategy known as sequential model-based optimization (**SMBO**) [HHLB11]. In this approach, we alternate between querying the function at a point, and updating our estimate of the surrogate based on the new data. More precisely, at each iteration  $n$ , we have a labeled dataset  $\mathcal{D}_n = \{(\mathbf{x}_i, y_i) : i = 1 : n\}$ , which records points  $\mathbf{x}_i$  that we have queried, and the corresponding function values  $y_i = f(\mathbf{x}_i) + \epsilon_i$ , where  $\epsilon_i$  is an optional noise term. We use this data to estimate a probability distribution over the true function  $f$ ; we will denote this by  $p(f|\mathcal{D}_n)$ . We then choose the next point to query  $\mathbf{x}_{n+1}$  using an **acquisition function**  $\alpha(\mathbf{x}; \mathcal{D}_n)$ , which computes the expected utility of querying  $\mathbf{x}$ . (We discuss acquisition functions in Section 6.9.3). After we observe  $y_{n+1} = f(\mathbf{x}_{n+1}) + \epsilon_{n+1}$ , we update our beliefs about the function, and repeat. See Algorithm 5 for some pseudocode.

---

**Algorithm 5:** Bayesian optimization

---

```

1 Collect initial dataset  $\mathcal{D}_0 = \{(\mathbf{x}_i, y_i) : \}$  from random queries  $\mathbf{x}_i$ ;
2 Initialize model  $p(f|\mathcal{D}_0)$  ;
3 for  $n = 1, 2, \dots$  until convergence do
4   Choose next query point  $\mathbf{x}_{n+1} = \operatorname{argmax}_{\mathbf{x} \in \mathcal{X}} \alpha(\mathbf{x}; \mathcal{D}_n)$ ;
5   Measure function value,  $y_{n+1} = f(\mathbf{x}_{n+1}) + \epsilon_n$  ;
6   Augment dataset,  $\mathcal{D}_{n+1} = \{\mathcal{D}_n, (\mathbf{x}_{n+1}, y_{n+1})\}$  ;
7   Update model,  $p(f|\mathcal{D}_{n+1}) \propto p(f|\mathcal{D}_n)p(y_{n+1}|\mathbf{x}_{n+1}, f)$ 
```

---

This method is illustrated in Figure 6.11. The goal is to find the global optimum of the solid black curve. In the first row, we show the 2 previously queried points,  $x_1$  and  $x_2$ , and their corresponding function values.  $y_1 = f(x_1)$  and  $y_2 = f(x_2)$ . Our uncertainty about the value of  $f$  at those locations is 0 (if we assume no observation noise), as illustrated by the posterior credible interval (shaded blue area) becoming “pinched”. Consequently the acquisition function (shown in green at the bottom) also has value 0 at those previously queried points. The red triangle represents the maximum of the acquisition function, which becomes our next query,  $x_3$ . In the second row, we show the result of observing  $y_3 = f(x_3)$ ; this further reduces our uncertainty about the shape of the function. In the third row, we show the result of observing  $y_4 = f(x_4)$ . This process repeats until we run out of time, or until we are confident there are no better unexplored points to query.

The two main “ingredients” that we need to provide to a BayesOpt algorithm are (1) a way to represent and update the posterior surrogate  $p(f|\mathcal{D}_n)$ , and (2) a way to define and optimize the acquisition function  $\alpha(\mathbf{x}; \mathcal{D}_n)$ . We discuss both of these topics below.

### 6.9.2 Surrogate functions

In this section, we discuss ways to represent and update the posterior over functions,  $p(f|\mathcal{D}_n)$ .

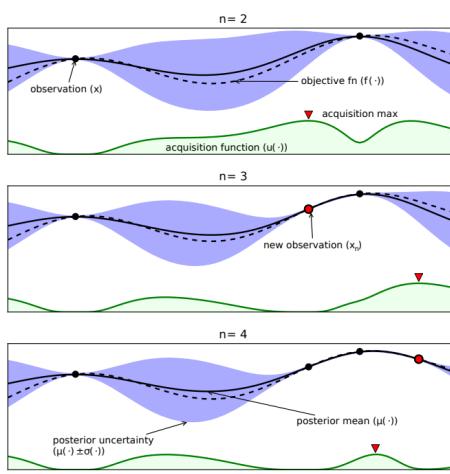


Figure 6.11: Illustration of sequential Bayesian optimization over three iterations. The rows correspond to a training set of size  $t = 2, 3, 4$ . The solid black line is the true, but unknown, function  $f(x)$ . The dotted black line is the posterior mean,  $\mu(x)$ . The shaded blue intervals are the 95% credible interval derived from  $\mu(x)$  and  $\sigma(x)$ . The solid black dots correspond to points whose function value has already been computed, i.e.,  $x_n$  for which  $f(x_n)$  is known. The green curve at the bottom is the acquisition function. The red dot is the proposed next point to query, which is the maximum of the acquisition function. From Figure 1 of [Sha+16]. Used with kind permission of Nando de Freitas.

24  
25  
26

### 6.9.2.1 Gaussian processes

In BayesOpt, it is very common to use a Gaussian process or GP for our surrogate. GPs are explained in detail in Chapter 18, but the basic idea is that they represent  $p(f(\mathbf{x})|\mathcal{D}_n)$  as a Gaussian,  $p(f(\mathbf{x})|\mathcal{D}_n) = \mathcal{N}(f|\mu_n(\mathbf{x}), \sigma_n^2(\mathbf{x}))$ , where  $\mu_n(\mathbf{x})$  and  $\sigma_n(\mathbf{x})$  are functions that can be derived from the training data  $\mathcal{D}_n = \{(\mathbf{x}_i, y_i) : i = 1 : n\}$  using a simple closed-form equation. The GP requires specifying a kernel function  $\mathcal{K}_{\theta}(\mathbf{x}, \mathbf{x}')$ , which measures similarities between input points  $\mathbf{x}, \mathbf{x}'$ . The intuition is that if two inputs are similar, so  $\mathcal{K}_{\theta}(\mathbf{x}, \mathbf{x}')$  is large, then the corresponding function values are also likely to be similar, so  $f(\mathbf{x})$  and  $f(\mathbf{x}')$  should be positively correlated. This allows us to interpolate the function between the labeled training points; in some cases, it also lets us extrapolate beyond them.

GPs work well when we have little training data, and they support closed form Bayesian updating. However, exact updating takes  $O(N^3)$  for  $N$  samples, which becomes too slow if we perform many function evaluations. There are various methods (Section 18.5.3) for reducing this to  $O(NM^2)$  time, where  $M$  is a parameter we choose, but this sacrifices some of the accuracy.

In addition, the performance of GPs depends heavily on having a good kernel. We can estimate the kernel parameters  $\theta$  by maximizing the marginal likelihood, as discussed in Section 18.6.1. However, since the sample size is small (by assumption), we can often get better performance by marginalizing out  $\theta$  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  $\Phi$  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

$$\alpha_{EI}(\mathbf{x}; \mathcal{D}_n) = \mathbb{E}_{\mathcal{D}_n} [U(\mathbf{x}, y)] = \mathbb{E}_{\mathcal{D}_n} [(f(\mathbf{x}) - V_n)\mathbb{I}(f(\mathbf{x}) > V_n)] \quad (6.231)$$

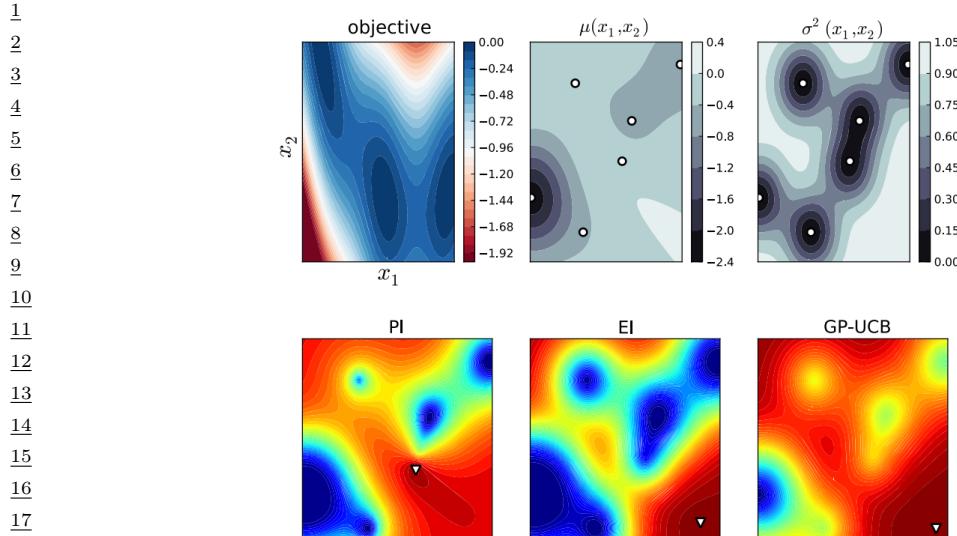


Figure 6.12: The first row shows the objective function, (the Branin function defined on  $\mathbb{R}^2$ ), and its posterior mean and variance using a GP estimate. White dots are the observed data points. The second row shows 3 different acquisition functions (probability of improvement, expected improvement, 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  $\beta_n$ , 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:

$$\underline{4} \quad V_{n+1}(\mathbf{x}, y) = \max_{\mathbf{x}'} \mathbb{E}_{p(f|\mathbf{x}, y, \mathcal{D}_n)} [f(\mathbf{x}')] \quad (6.238)$$

$$\underline{5} \quad V_{n+1}(\mathbf{x}) = \mathbb{E}_{p(y|\mathbf{x}, \mathcal{D}_n)} [V_{n+1}(\mathbf{x}, y)] \quad (6.239)$$

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 We can formalize such problems by defining the search space in terms of a tree or DAG (directed  
41 acyclic graph), where different subsets of the parameters are defined at each leaf. Applying GPs to  
42

this setting requires non-standard kernels, such as those discussed in [Swe+13; Jen+17]. Alternatively, we can use other forms of Bayesian regression, such as ensembles of random forests [HHLB11], which can easily handle conditional parameter spaces.

#### 6.9.4.3 Multi-fidelity surrogates

In some cases, we can construct surrogate functions with different levels of accuracy, each of which may take variable amounts of time to compute. In particular, let  $f(\mathbf{x}, s)$  be an approximation to the true function at  $\mathbf{x}$  with fidelity  $s$ . The goal is to solve  $\max_{\mathbf{x}} f(\mathbf{x}, 0)$  by observing  $f(\mathbf{x}, s)$  at a sequence of  $(\mathbf{x}_i, s_i)$  values, such that the total cost  $\sum_{i=1}^n c(s_i)$  is below some budget. For example, in the context of hyperparameter selection,  $s$  may control how long we run the parameter optimizer for, or how large the validation set is.

In addition to choosing what fidelity to use for an experiment, we may choose to terminate expensive trials (queries) early, if the results of their cheaper proxies suggest they will not be worth running to completion (see e.g., [Str19; Li+17b; FKH17]). Alternatively, we may choose to resume an earlier aborted run, to collect more data on it, as in the **freeze-thaw algorithm** [SSA14].

#### 6.9.4.4 Constraints

If we want to maximize a function subject to known constraints, we can simply build the constraints into the acquisition function. But if the constraints are unknown, we need to estimate the support of the feasible set in addition to estimating the function. In [GSA14], they propose the weighted EI criterion, given by  $\alpha_{wEI}(\mathbf{x}; \mathcal{D}_n) = \alpha_{EI}(\mathbf{x}; \mathcal{D}_n)h(\mathbf{x}; \mathcal{D}_n)$ , where  $h(\mathbf{x}; \mathcal{D}_n)$  is a GP with a Bernoulli observation model that specifies if  $\mathbf{x}$  is feasible or not. Of course, other methods are possible. For example, [HL+16b] propose a method based on predictive entropy search.

## 6.10 Optimal Transport

*This section is written by Marco Cuturi.*

In this section, we focus on **optimal transport** theory, a set of tools that have been proposed, starting with work by [Mon81], to compare two probability distributions. We start from a simple example involving only matchings, and work from there towards various extensions.

### 6.10.1 Warm-up: Matching optimally two families of points

Consider two families  $(\mathbf{x}_1, \dots, \mathbf{x}_n)$  and  $(\mathbf{y}_1, \dots, \mathbf{y}_n)$ , each consisting in  $n > 1$  distinct points taken from a set  $\mathcal{X}$ . A *matching* between these two families is a bijective mapping that assigns to each point  $\mathbf{x}_i$  another point  $\mathbf{y}_j$ . Such an assignment can be encoded by pairing indices  $(i, j) \in \{1, \dots, n\}^2$  such that they define a *permutation*  $\sigma$  in the symmetric group  $\mathcal{S}_n$ . With that convention and given a permutation  $\sigma$ ,  $\mathbf{x}_i$  would be assigned to  $\mathbf{y}_{\sigma_i}$ , the  $\sigma_i$ -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

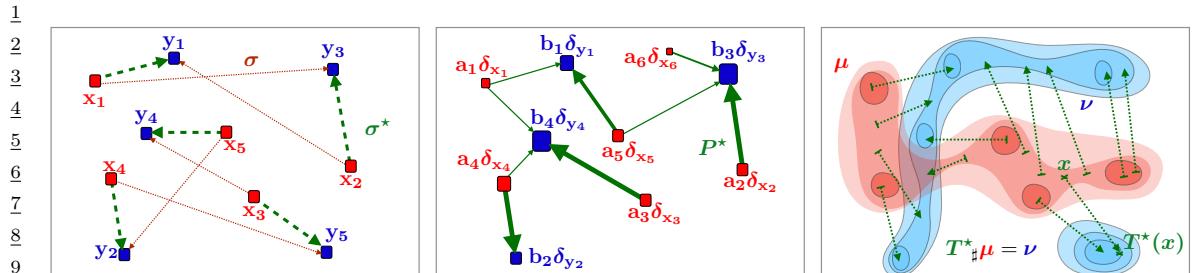


Figure 6.13: (left) Matching a family of 5 points to another is equivalent to considering a permutation in  $\{1, \dots, n\}$ . When to each pair  $(\mathbf{x}_i, \mathbf{y}_j) \in \mathbb{R}^2$  is associated a cost equal to the distance  $\|\mathbf{x}_i - \mathbf{y}_j\|$ , the optimal matching problem involves finding a permutation  $\sigma$  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  $\mu$  is associated a point  $y = T(x)$  in the support of  $\nu$ . The push-forward constraint  $T_{\#}\mu = \nu$  ensures that  $\nu$  is recovered by applying map  $T$  to all points in the support of  $\mu$ ; the optimal map  $T^*$  is that which minimizes the distance between  $x$  and  $T(x)$ , averaged over  $\mu$ .

23

24

vector of skills held by a job seeker  $i$  and  $\mathbf{y}_j$  a vector quantifying desirable skills associated with a job posting  $j$ ; in that case  $C_{ij}$  could quantify the numbers of hours required for worker  $i$  to carry out job  $j$ . We will assume without loss of generality that the values  $C_{ij}$  are obtained by evaluating a cost function  $c : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  on the pair  $(\mathbf{x}_i, \mathbf{y}_j)$ , namely  $C_{ij} = c(\mathbf{x}_i, \mathbf{y}_j)$ . In many applications of optimal transport, such cost functions have a geometric interpretation and are typically distance functions on  $\mathcal{X}$  as in Fig. 6.13, in which  $\mathcal{X} = \mathbb{R}^2$ , or as will be later discussed in §6.10.2.4.

**Least-cost Matchings.** Equipped with a cost function  $c$ , the *optimal* matching (or assignment) problem is that of finding a permutation that reaches the smallest total cost, as defined by the function

$$\min_{\sigma} E(\sigma) = \sum_{i=1}^n c(\mathbf{x}_i, \mathbf{y}_{\sigma_i}). \quad (6.241)$$

The optimal matching problem is arguably one of the simplest combinatorial optimization problems, tackled as early as the 19th century [JB65]. Although a naive enumeration of all permutations would require evaluating objective  $E$  a total of  $n!$  times, the Hungarian algorithm [Kuh55] was shown to provide the optimal solution in polynomial time [Mun57], and later refined to require in the worst case  $O(n^3)$  operations.

### 6.10.2 From Optimal Matchings to Kantorovich and Monge formulations

The optimal matching problem is relevant to many applications, but it suffers from a few limitations. One could argue that most of the optimal transport literature arises from the necessity to overcome

these limitations and extend (6.241) to more general settings. An obvious issue arises when the number of points available in both families is not the same. The second limitation arises when considering a continuous setting, namely when trying to match (or morph) two probability densities, rather than families of atoms (discrete measures).

### 6.10.2.1 Mass splitting

Suppose again that all points  $\mathbf{x}_i$  and  $\mathbf{y}_j$  describe skills, respectively held by a worker  $i$  and needed for a task  $j$  to be fulfilled in a factory. Since finding a matching is equivalent to finding a permutation in  $\{1, \dots, n\}$ , problem (6.241) cannot handle cases in which the number of workers is larger (or smaller) than the number of tasks. More problematically, the assumption that every single task is indivisible, or that workers are only able to dedicate themselves to a single task, is hardly realistic. Indeed, certain tasks may require more (or less) dedication than that provided by a single worker, whereas some workers may only be able to work part-time, or, on the contrary, be willing to put extra hours. The rigid machinery of permutations falls short of handling such cases, since permutations are by definition one-to-one associations. The Kantorovich formulation allows for *mass-splitting*, the idea that the effort provided by a worker or needed to complete a given task can be split. In practice, to each of the  $n$  workers is associated, in addition to  $\mathbf{x}_i$ , a positive number  $\mathbf{a}_i > 0$ . That number represents the amount of time worker  $i$  is able to provide. Similarly, we introduce numbers  $\mathbf{b}_j > 0$  describing the amount of time needed to carry out each of the  $m$  tasks ( $n$  and  $m$  do not necessarily coincide). Worker  $i$  is therefore described as a pair  $(\mathbf{a}_i, \mathbf{x}_i)$ , mathematically equivalent to a *weighted Dirac measure*  $\mathbf{a}_i \delta_{\mathbf{x}_i}$ . The overall workforce available to the factory is described as a discrete measure  $\sum_i \mathbf{a}_i \delta_{\mathbf{x}_i}$ , whereas its tasks are described in  $\sum_j \mathbf{b}_j \delta_{\mathbf{y}_j}$ . If one assumes further that the factory has a balanced workload, namely that  $\sum_i \mathbf{a}_i = \sum_j \mathbf{b}_j$ , then the Kantorovich [Kan42] formulation of optimal transport is:

$$\text{OT}_C(\mathbf{a}, \mathbf{b}) \triangleq \min_{P \in \mathbb{R}_+^{n \times m}, P\mathbf{1}_m = \mathbf{a}, P^T \mathbf{1}_m = \mathbf{b}} \langle P, C \rangle \triangleq \sum_{i,j} P_{ij} C_{ij}. \quad (\text{K})$$

The interpretation behind such matrices is simple: each coefficient  $P_{ij}$  describes an allocation of time for worker  $i$  to spend on task  $j$ . The  $i$ -th row-sum must be equal to the total  $\mathbf{a}_i$  for the time constraint of worker  $i$  to be satisfied, whereas the  $j$ -th column-sum must be equal to  $\mathbf{b}_j$ , reflecting that the time needed to complete task  $j$  has been budgeted.

### 6.10.2.2 Monge formulation and optimal push-forward maps

By introducing mass-splitting, the Kantorovich formulation of optimal transport allows for a far more general comparison between discrete measures of different sizes and weights (middle plot of Fig. 6.13). Naturally, this flexibility comes with a downside: one can no longer associate to each point  $\mathbf{x}_i$  another point  $\mathbf{y}_j$  to which it is uniquely associated, as was the case with the classical matching problem. Interestingly, this property can be recovered in the limit where the measures become densities. Indeed, the Monge [Mon81] formulation of optimal transport allows to recover precisely that property, on the condition (loosely speaking) that measure  $\mu$  admits a density. In that setting, the analogous mathematical object guaranteeing that  $\mu$  is mapped onto  $\nu$  is that of *push-forward* maps morphing  $\mu$  to  $\nu$ , namely maps  $T$  such that for any measurable set  $A \subset \mathcal{X}$ ,  $\mu(T^{-1}(A)) = \nu(A)$ . When  $T$  is differentiable, and  $\mu, \nu$  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:

3

$$\underline{q}(T(x)) = p(x)|J_T(x)|, \quad (6.242)$$

4 where  $|J_T(x)|$  stands for the determinant of the Jacobian matrix of  $T$  evaluated at  $x$ .

5 Writing  $T_\sharp\mu = \nu$  when  $T$  does satisfy these conditions, the Monge [Mon81] problem consists in  
6 finding the best map  $T$  that minimizes the average cost between  $\mathbf{x}$  and its displacement  $T(\mathbf{x})$ ,

7

$$\inf_{T: T_\sharp\mu = \nu} \int_{\mathcal{X}} c(\mathbf{x}, T(\mathbf{x})) \mu(d\mathbf{x}). \quad (\text{M})$$

8  $T$  is therefore a map that pushes forward  $\mu$  to  $\nu$  globally, but which results, on average, in the smallest  
9 average cost. While very intuitive, the Monge problem turns out to be extremely difficult to solve in  
10 practice, since it is non-convex. Indeed, one can easily check that the constraint  $\{T_\sharp\mu = \nu\}$  is not  
11 convex, since one can easily find counter-examples for which  $T_\sharp\mu = \nu$  and  $T'_\sharp\nu$  yet  $(\frac{1}{2}T + \frac{1}{2}T')_\sharp\mu \neq \nu$ .  
12 Luckily, Kantorovich's approach also works for continuous measures, and yields a comparatively  
13 much simpler linear program.

14

### 15 6.10.2.3 Kantorovich formulation

16 The Kantorovich problem (K) can also be extended to a continuous setting: Instead of optimizing  
17 over a subset of matrices in  $\mathbb{R}^{n \times m}$ , consider  $\Pi(\mu, \nu)$ , the subset of joint probability distributions  
18  $\mathcal{P}(\mathcal{X} \times \mathcal{X})$  with marginals  $\mu$  and  $\nu$ , namely

19

$$\Pi(\mu, \nu) \triangleq \{\pi \in \mathcal{P}(\mathcal{X}^2) : \forall A \subset \mathcal{X}, \pi(A \times \mathcal{X}) = \mu(A) \text{ and } \pi(\mathcal{X} \times A) = \nu(A)\}. \quad (6.243)$$

20 Note that  $\Pi(\mu, \nu)$  is not empty since it always contains the product measure  $\mu \otimes \nu$ . With this  
21 definition, the continuous formulation of (K) can be obtained as

22

$$\text{OT}_c(\mu, \nu) \triangleq \inf_{\pi \in \Pi(\mu, \nu)} \int_{\mathcal{X}^2} c d\pi. \quad (\text{K2})$$

23 Notice that (K2) subsumes directly (K), since one can check that they coincide when  $\mu, \nu$  are  
24 discrete measures, with respective probability weights  $\mathbf{a}, \mathbf{b}$  and locations  $(\mathbf{x}_1, \dots, \mathbf{x}_n)$  and  $(\mathbf{y}_1, \dots, \mathbf{y}_m)$ .

### 25 6.10.2.4 Wasserstein distances

26 When  $c$  is equal to a metric  $d$  exponentiated by an integer, the optimal value of the Kantorovich  
27 problem is called the Wasserstein *distance* between  $\mu$  and  $\nu$ :

28

$$W_p(\mu, \nu) \triangleq \left( \inf_{\pi \in \Pi(\mu, \nu)} \int_{\mathcal{X}^2} d(\mathbf{x}, \mathbf{y})^p d\pi(\mathbf{x}, \mathbf{y}) \right)^{1/p}. \quad (6.244)$$

29 While the symmetry and the fact that  $W_p(\mu, \nu) = 0 \Rightarrow \mu = \nu$  are relatively easy to prove provided  $d$   
30 is a metric, proving the triangle inequality is slightly more challenging, and builds on a result known  
31 as the gluing lemma ([Vil08, p.23]). The  $p$ -th power of  $W_p(\mu, \nu)$  is often abbreviated as  $W_p^p(\mu, \nu)$ .

32

1 **6.10.3 Solving optimal transport**

2 **6.10.3.1 Duality and cost concavity**

3 Both (K) and (K2) are linear programs: their constraints and objective functions only involve  
4 summations. In that sense they admit a dual formulation (here, again, (DK2) subsumes (DK)):

5 
$$\max_{\mathbf{f} \in \mathbb{R}^n, \mathbf{g} \in \mathbb{R}^m} \mathbf{f}^T \mathbf{a} + \mathbf{g}^T \mathbf{b} \quad (\text{DK})$$
  
6 
$$\mathbf{f} \oplus \mathbf{g} \leq C$$

7 
$$\sup_{f \oplus g \leq c} \int_{\mathcal{X}} f \, d\mu + \int_{\mathcal{X}} g \, d\nu; \quad (\text{DK2})$$
  
8

9 where the sign  $\oplus$  denotes tensor addition for vectors,  $\mathbf{f} \oplus \mathbf{g} = [\mathbf{f}_i + \mathbf{g}_j]_{ij}$ , or functions,  $f \oplus g : \mathbf{x}, \mathbf{y} \mapsto$   
10  $f(\mathbf{x}) + g(\mathbf{y})$ . In other words, the dual problem looks for a pair of vectors (or functions) that attain  
11 the highest possible expectation when summed against  $\mathbf{a}$  and  $\mathbf{b}$  (or integrated against  $\mu, \nu$ ), pending  
12 the constraint that they do not differ too much across points  $\mathbf{x}, \mathbf{y}$ , as measured by  $c$ .

13 The dual problems in (K) and (K2) have two variables. Focusing on the continuous formulation, a  
14 closer inspection shows that it is possible, given a function  $f$  for the first measure, to compute the  
15 best possible candidate for function  $g$ . That function  $g$  should be as large as possible, yet satisfy the  
16 constraint that  $g(\mathbf{y}) \leq c(\mathbf{x}, \mathbf{y}) - f(\mathbf{x})$  for all  $\mathbf{x}, \mathbf{y}$ , making

17 
$$\forall \mathbf{y} \in \mathcal{X}, \bar{f}(\mathbf{y}) \triangleq \inf_{\mathbf{x}} c(\mathbf{x}, \mathbf{y}) - f(\mathbf{x}), \quad (6.245)$$
  
18

19 the optimal choice.  $\bar{f}$  is called the  $c$ -transform of  $f$ . Naturally, one may choose to start instead from  
20  $g$ , to define an alternative  $c$ -transform:  
21

22 
$$\forall \mathbf{x} \in \mathcal{X}, \tilde{g}(\mathbf{x}) \triangleq \inf_{\mathbf{y}} c(\mathbf{x}, \mathbf{y}) - g(\mathbf{y}). \quad (6.246)$$
  
23

24 Since these transformations can only improve solutions, one may even think of applying alternatively  
25 these transformations to an arbitrary  $f$ , to define  $\bar{f}$ ,  $\tilde{\bar{f}}$  and so on. One can show, however, that this  
26 has little interest, since  
27

28 
$$\tilde{\bar{f}} = \bar{f}. \quad (6.247)$$
  
29

30 This remark allows, nonetheless, to narrow down the set of candidate functions to those that have  
31 already undergone such transformations. This reasoning yields the so-called set of  $c$ -concave functions,  
32  $\mathcal{F}_c \triangleq \{f \mid \exists g : \mathcal{X} \rightarrow \mathbb{R}, f = \tilde{g}\}$ , which can be shown, equivalently, to be the set of functions  $f$  such  
33 that  $f = \bar{f}$ . One can therefore focus our attention to  $c$ -concave functions to solve (DK2) using a  
34 so-called semi-dual formulation,

35 
$$\sup_{f \in \mathcal{F}_c} \int_{\mathcal{X}} f \, d\mu + \int_{\mathcal{X}} \bar{f} \, d\nu. \quad (\text{DK2})$$
  
36

37 Going from (DK2) to (DK2), we have removed a dual variable  $g$  and narrowed down the feasible set  
38 to  $\mathcal{F}_c$ , at the cost of introducing the highly non-linear transform  $\bar{f}$ . This reformulation is, however,  
39 very useful, in the sense that it allows to restrict our attention on  $c$ -concave functions, notably for  
40 two important classes of cost functions  $c$ : distances and squared-Euclidean norms.

41

1 **6.10.3.2 Kantorovich-Rubinstein duality and Lipschitz potentials**

3 A striking result illustrating the interest of  $c$ -concavity is provided when  $c$  is a metric  $d$ , namely when  
4  $p = 1$  in (6.244). In that case, one can prove (exploiting notably the triangle inequality of the  $d$ )  
5 that a  $d$ -concave function  $f$  is 1-Lipschitz (one has  $|f(\mathbf{x}) - f(\mathbf{y})| \leq d(\mathbf{x}, \mathbf{y})$  for any  $\mathbf{x}, \mathbf{y}$ ) and such  
6 that  $\bar{f} = -f$ . This result translates therefore in the following identity:  
7

$$\underline{8} \quad W_1(\mu, \nu) = \sup_{f \text{1-Lipschitz}} \int_{\mathcal{X}} f(d\mu - d\nu). \quad (6.248)$$

10 This result has numerous practical applications. This supremum over 1-Lipschitz functions can be  
11 efficiently approximated using Wavelet coefficients of densities in low-dimensions [SJ08], or heuristically  
12 in more general cases by training neural networks parameterized to be 1-Lipschitz [ACB17] using  
13 ReLU activation functions, and bounds on the entries of the weight matrices.  
14

15 **6.10.3.3 Monge maps as gradients of convex functions: the Brenier theorem**

17 Another application of  $c$ -concavity lies in the case  $c(\mathbf{x}, \mathbf{y}) = \frac{1}{2}\|\mathbf{x} - \mathbf{y}\|^2$ , which corresponds, up to the  
18 factor  $\frac{1}{2}$ , to the squared  $W_2$  distance used between densities in an Euclidean space. The remarkable  
19 result, shown first by [Bre91], is that the Monge map solving (M) between two measures for that  
20 cost (taken for granted  $\mu$  is regular enough, here assumed to have a density w.r.t. Lebesgue measure)  
21 exists and is necessarily the gradient of a convex function. In loose terms, one can show that

$$\underline{22} \quad T^* = \arg \min_{T: T_\# \mu = \nu} \int_{\mathcal{X}} \frac{1}{2}\|\mathbf{x} - T(\mathbf{x})\|_2^2 \mu(d\mathbf{x}). \quad (\text{M})$$

25 exists, and is the gradient of a convex function  $u: \mathbb{R}^d \rightarrow \mathbb{R}$ , namely  $T^* = \nabla u$ . Conversely, for any  
26 convex function  $u$ , the optimal transport map between  $\mu$  and the displacement  $\nabla u_\# \mu$  is necessarily  
27 equal to  $\nabla u$ .

28 We provide a sketch of the proof: one can always exploit, for any reasonable cost function  $c$   
29 (e.g. lower bounded and lower semi continuous), primal-dual relationships: Consider an optimal  
30 coupling  $P^*$  for (K2), as well as an optimal  $c$ -concave dual function  $f^*$  for (DK2). This implies in  
31 particular that  $(f^*, g^* = \bar{f}^*)$  is optimal for (DK2). Complementary slackness conditions for this pair  
32 of linear programs imply that if  $\mathbf{x}_0, \mathbf{y}_0$  is in the support of  $P^*$ , then necessarily (and sufficiently)  
33  $f^*(\mathbf{x}_0) + \bar{f}^*(\mathbf{y}_0) = c(\mathbf{x}_0, \mathbf{y}_0)$ . Suppose therefore that  $\mathbf{x}_0, \mathbf{y}_0$  is indeed in the support of  $P^*$ . From the  
34 equality  $f^*(\mathbf{x}_0) + \bar{f}^*(\mathbf{y}_0) = c(\mathbf{x}_0, \mathbf{y}_0)$  one can trivially obtain that  $\bar{f}^*(\mathbf{y}_0) = c(\mathbf{x}_0, \mathbf{y}_0) - f^*(\mathbf{x}_0)$ . Yet,  
35 recall also that, by definition,  $\bar{f}^*(\mathbf{y}_0) = \inf_{\mathbf{x}} c(\mathbf{x}, \mathbf{y}_0) - f^*(\mathbf{x})$ . Therefore,  $\mathbf{x}_0$  has the special property  
36 that it minimizes  $\mathbf{x} \rightarrow c(\mathbf{x}, \mathbf{y}_0) - f^*(\mathbf{x})$ . If, at this point, one recalls that  $c$  is assumed in this section  
37 to be  $c(\mathbf{x}, \mathbf{y}) = \frac{1}{2}\|\mathbf{x} - \mathbf{y}\|^2$ , one has therefore that  $\mathbf{x}_0$  verifies

$$\underline{38} \quad \mathbf{x}_0 \in \operatorname{argmin}_{\mathbf{x}} \frac{1}{2}\|\mathbf{x} - \mathbf{y}_0\|^2 - f^*(\mathbf{x}). \quad (6.249)$$

40 Assuming  $f^*$  is differentiable, which one can prove by  $c$ -concavity, this yields the identity  
41

$$\underline{42} \quad \mathbf{y}_0 - \mathbf{x}_0 - \nabla f^*(\mathbf{x}_0) = 0 \Rightarrow \mathbf{y}_0 = \mathbf{x}_0 - \nabla f^*(\mathbf{x}_0) = \nabla \left( \frac{1}{2}\|\cdot\|^2 - f^* \right) (\mathbf{x}_0). \quad (6.250)$$

43 Therefore, if  $(\mathbf{x}_0, \mathbf{y}_0)$  is in the support of  $P^*$ ,  $\mathbf{y}_0$  is uniquely determined, which proves  $P^*$  is in fact a  
44 Monge map “disguised” as a coupling, namely  
45

$$\underline{46} \quad P^* = (\operatorname{Id}, \nabla \left( \frac{1}{2}\|\cdot\|^2 - f^* \right))_\# \mu. \quad (6.251)$$

The end of the proof can be worked out as follows: For any function  $h : \mathcal{X} \rightarrow \mathbf{R}$ , one can show, using the definitions of  $c$ -transforms and the Legendre transform, that  $\frac{1}{2}\|\cdot\|^2 - h$  is convex if and only if  $h$  is  $c$ -concave. An intermediate step in that proof relies on showing that  $\frac{1}{2}\|\cdot\|^2 - \bar{h}$  is equal to the Legendre transform of  $\frac{1}{2}\|\cdot\|^2 - h$ . The function  $\frac{1}{2}\|\cdot\|^2 - f^*$  above is therefore convex, by  $c$ -concavity of  $f^*$ , and the optimal transport map is itself the gradient of a convex function.

Knowing that an optimal transport map for the squared-Euclidean cost is necessarily the gradient of a convex function can prove very useful to solve (DK2). Indeed, this knowledge can be leveraged to restrict estimation to relevant families of functions, namely gradients of input-convex neural networks[AXK17], as proposed in [Mak+20] or [Kor+20], as well as arbitrary convex functions with desirable smoothness and strong-convexity constants [PdC20].

### 6.10.3.4 Closed forms for univariate and Gaussian distributions

Many metrics between probability distributions have closed form expressions for simple cases. The Wasserstein distance is no exception, and can be computed in close form in two important scenarios. When distributions are univariate and the cost  $c(\mathbf{x}, \mathbf{y})$  is either a convex function of the difference  $\mathbf{x} - \mathbf{y}$ , or when  $\partial c / \partial \mathbf{x} \partial \mathbf{y} < 0$  a.e., then the Wasserstein distance is essentially a comparison between the quantile functions of  $\mu$  and  $\nu$ . Recall that for a measure  $\rho$ , its quantile function  $Q_\rho$  is a function that takes values in  $[0, 1]$  and is valued in the support of  $\rho$ , and corresponds to the (generalized) inverse map of  $F_\rho$ , the cumulative distribution function (cdf) of  $\rho$ . 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  $\mu$  and that of  $\nu$ . 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  $\mu$  and  $\nu$ . Note finally that the Monge map is also available in closed form, and is equal to  $Q_\nu \circ F_\mu$ .

The second closed form applies to so-called elliptically contoured distributions, chiefly among them Gaussian multivariate distributions[Gel90]. For two Gaussians  $\mathcal{N}(\mathbf{m}_1, \Sigma_1)$  and  $\mathcal{N}(\mathbf{m}_2, \Sigma_2)$  their 2-Wasserstein distance decomposes as

$$W_2^2(\mathcal{N}(\mathbf{m}_1, \Sigma_1), \mathcal{N}(\mathbf{m}_2, \Sigma_2)) = \|\mathbf{m}_1 - \mathbf{m}_2\|^2 + \mathcal{B}^2(\Sigma_1, \Sigma_2) \quad (6.253)$$

where the Bures metric  $\mathcal{B}$  reads:

$$\mathcal{B}^2(\Sigma_1, \Sigma_2) = \text{trace} \left( \Sigma_1 + \Sigma_2 - 2 \left( \Sigma_1^{\frac{1}{2}} \Sigma_2 \Sigma_1^{\frac{1}{2}} \right)^{\frac{1}{2}} \right). \quad (6.254)$$

Notice in particular that these quantities are well-defined even when the covariance matrices are not invertible, and that they collapse to the distance between means as both covariances become 0. When the first covariance matrix is invertible, one has that the optimal Monge map is given by

$$T \triangleq \mathbf{x} \mapsto A(\mathbf{x} - \mathbf{m}_1) + \mathbf{m}_2, \text{ where } A \triangleq \Sigma_1^{-\frac{1}{2}} \left( \Sigma_1^{\frac{1}{2}} \Sigma_2 \Sigma_1^{\frac{1}{2}} \right)^{\frac{1}{2}} \Sigma_1^{-\frac{1}{2}} \quad (6.255)$$

1 It is easy to show that  $T^*$  is indeed optimal: The fact that  $T_{\sharp}\mathcal{N}(\mathbf{m}_1, \Sigma_1) = \mathcal{N}(\mathbf{m}_2, \Sigma_2)$  follows from  
 2 the knowledge that the affine push-forward of a Gaussian is another Gaussian. Here  $T$  is designed  
 3 to push precisely the first Gaussian onto the second (and  $A$  designed to recover random variables  
 4 with variance  $\Sigma_2$  when starting from random variables with variance  $\Sigma_1$ ). 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

### 9 6.10.3.5 Exact evaluation using linear program solvers

10 We have hinted, using duality and  $c$ -concavity, that methods based on stochastic optimization over  
 11 1-Lipschitz or convex neural networks can be employed to estimate Wasserstein distances when  $c$  is  
 12 the Euclidean distance or its square. These approaches are, however, non-convex and can only reach  
 13 local optima. Apart from these two cases, and the closed forms provided above, the only reliable  
 14 approach to compute Wasserstein distances appears when both  $\mu$  and  $\nu$  are discrete measures: In  
 15 that case, one can instantiate and solve the discrete (K) problem, or its dual (DK) formulation.  
 16 The primal problem is a canonical example of network flow problems, and can be solved with the  
 17 network-simplex method in  $O(nm(n+m)\log(n+m))$  complexity [AMO88], or, alternatively, with  
 18 the comparable auction algorithm [BC89]. These approaches suffer from computational limitations:  
 19 their cubic cost is intractable for large scale scenarios; their combinatorial flavor makes it harder to  
 20 solve to parallelize simultaneously the computation of multiple optimal transport problems with a  
 21 common cost matrix  $C$ .  
 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  $\mu, \nu$  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  $\mu$  and  $\nu$ . 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

40 A computational approach to speed-up the resolution of (K) was proposed in [Cut13], building  
 41 on earlier contributions [Wil69; KY94] and a filiation to the Schrödinger bridge problem in the  
 42 special case where  $c = d^2$  [Léo14]. The idea rests upon regularizing the transportation cost by the  
 43 Kullback-Leibler divergence of the coupling to the product measure of  $\mu, \nu$ ,  
 44

$$45 \quad W_{c,\gamma}(\mu, \nu) \triangleq \inf_{\pi \in \Pi(\mu, \nu)} \int_{\mathcal{X}^2} d(\mathbf{x}, \mathbf{y})^p d\pi(\mathbf{x}, \mathbf{y}) + \gamma D_{\text{KL}}(\pi \parallel \mu \otimes \nu). \quad (6.256)$$

46

When instantiated on discrete measures, this problem is equivalent to the following  $\gamma$ -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  $P_\gamma^*$  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  $\gamma$  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.<sup>4</sup> Submodularity has an extremely simple definition. However, the “simplest things are often the most complicated to understand fully” [Sam74], and while submodularity has been studied extensively over the years, it continues to yield new and surprising insights and properties, some of which are extremely relevant to data science, machine learning, and artificial intelligence. A submodular function operates on subsets of some finite *ground set*,  $V$ . Finding a guaranteed good subset of  $V$  would ordinarily require an amount of computation exponential in the size of  $V$ . Submodular functions, however, have certain properties that make optimization either tractable or approximable where otherwise neither would be possible. The properties are quite natural, however, so submodular functions are both flexible and widely applicable to real problems. Submodularity involves an intuitive and natural diminishing returns property, stating that adding an element to a smaller set helps

4. A greatly extended version of the material in this section may be found at [Bil22].

more than adding it to a larger set. Like convexity, submodularity allows one to efficiently find provably optimal or near-optimal solutions. In contrast to convexity, however, where little regarding maximization is guaranteed, submodular functions can be both minimized and (approximately) maximized. Submodular maximization and minimization, however, require very different algorithmic solutions and have quite different applications. It is sometimes said that submodular functions are a discrete form of convexity. This is not quite true, as submodular functions are like both convex and concave functions, but also have properties that are similar simultaneously to both convex and concave functions at the same time, but then some properties of submodularity are like neither convexity nor concavity. Convexity and concavity, for example, can be conveyed even as univariate functions. This is impossible for submodularity, as submodular functions are defined based only on the response of the function to changes amongst different variables in a multidimensional discrete space.

14

### 15 **6.11.1 Intuition, Examples, and Background**

16

17 Let us define a *set function*  $f : 2^V \rightarrow \mathbb{R}$  as one that assigns a value to every subset of  $V$ . The 18 notation  $2^V$  is the power set of  $V$ , and has size  $2^{|V|}$  which means that  $f$  lives in space  $\mathbb{R}^{2^n}$  — i.e., 19 since there are  $2^n$  possible subsets of  $V$ ,  $f$  can return  $2^n$  distinct values. We use the notation  $X + v$  20 as shorthand for  $X \cup \{v\}$ . Also, the value of an element in a given context is so widely used a 21 concept, we have a special notation for it — the incremental value *gain* of  $v$  in the context if  $X$  is 22 defined as  $f(v|X) = f(X + v) - f(X)$ . Thus, while  $f(v)$  is the value of element  $v$ ,  $f(v|X)$  is the 23 value of element  $v$  if you already have  $X$ . We also define the gain of set  $X$  in the context of  $Y$  as 24  $f(X|Y) = f(X \cup Y) - f(Y)$ .

25

#### 26 **6.11.1.1 Coffee, Lemon, Milk, Tea**

27

28 As a simple example, will explore the manner in which the value of everyday items may interact and 29 combine, namely coffee, lemon, milk, and tea. Consider the value relationships amongst the four 30 items coffee ( $c$ ), lemon ( $l$ ), milk ( $m$ ), and tea ( $t$ ) as shown in Figure 6.14. Suppose you just woke up, 31 and there is a function  $f : 2^V \rightarrow \mathbb{R}$  that provides the average valuation for any subset of the items in 32  $V$  where  $V = \{c, l, m, t\}$ . You can think of this function as giving the average price a typical person 33 would be willing to pay for any subset of items. Since nothing should cost nothing, we would expect 34 that  $f(\emptyset) = 0$ . Clearly, one needs either coffee or tea in the morning, so  $f(c) > 0$  and  $f(t) > 0$ , and 35 coffee is usually more expensive than tea, so that  $f(c) > f(t)$  pound for pound. Also more items cost 36 more, so that, for example,  $0 < f(c) < f(c, m) < f(c, m, t) < f(c, l, m, t)$ . Thus, the function  $f$  is 37 strictly *monotone*, or  $f(X) < f(Y)$  whenever  $X \subset Y$ .

38 The next thing we note is that coffee and tea may substitute for each other — they both have 39 the same effect, waking you up. They are mutually redundant, and they decrease each other's 40 value since once you have had a cup of coffee, a cup of tea is less necessary and less desirable. Thus, 41  $f(c, t) < f(c) + f(t)$ , which is known as a *subadditive* relationship, the whole is less than the sum 42 of the parts. On the other hand, some items complement each other. For example, milk and coffee 43 are better combined together than when both are considered in isolation, or  $f(m, c) > f(m) + f(c)$ , 44 a *superadditive* relationship, the whole is more than the sum of the parts. A few of the items 45 do not affect each others' price. For example, lemon and milk cost the same together as apart, so 46  $f(l, m) = f(l) + f(m)$ , an *additive* or *modular* relationship — such a relationship is perhaps midway 47

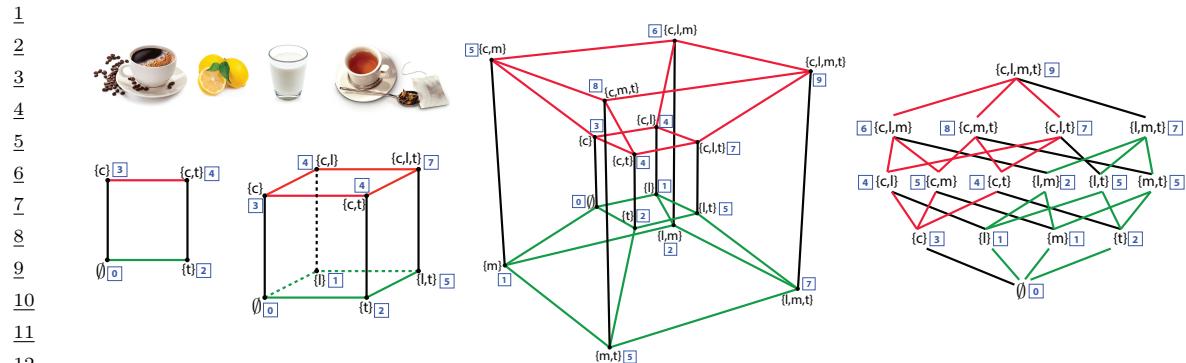


Figure 6.14: The value relationships between coffee ( $c$ ), lemon ( $l$ ), milk ( $m$ ), and tea ( $t$ ). On the left, we first see a simple square showing the relationships between coffee and tea, and see that they are substitutive (or submodular). In this, and all of the shapes, the vertex label set is indicated in curly braces and the value at that vertex is a blue integer in a box. We next see a three-dimensional cube that adds lemon to coffee and tea set. We see that tea and lemon are complementary (supermodular) but coffee and lemon are additive (modular, or independent). We next see a four-dimensional hypercube (tesseract) showing all of the value relationships described in the text. The four-dimensional hypercube is also shown as a lattice (on the right) showing the same relationships as well as two (red and green, also shown in the tesseract) of the eight three-dimensional cubes contained within.

between a subadditive and a superadditive relationship, and can be seen as a form of independence.

Things become more interesting when we consider three or more items together. For example, once you have tea, lemon becomes less valuable when you acquire milk since there might be those that prefer milk to lemon in their tea. Similarly, milk becomes less valuable once you have acquired lemon since there are those who prefer lemon in their tea to milk. So, once you have tea, lemon and milk are substitutive, you would never use both as the lemon would only curdle the milk. These are *submodular* relationships,  $f(l|m, t) < f(l|t)$  and  $f(m|l, t) < f(m|t)$  each of which implies that  $f(l, t) + f(m, t) > f(l, m, t) + f(t)$ . The value of lemon (respectively milk) with tea decreases in the larger context of having milk (respectively lemon) with tea, typical of submodular relationships.

Not all of the items are in a submodular relationship, as sometimes the presence of an item can increase the value of another item. For example, once you have milk, then tea becomes still more valuable when you also acquire lemon, since tea with the choice of either lemon or milk is more valuable than tea with the option only of milk. Similarly, once you have milk, lemon becomes more valuable when you acquire tea, since lemon with milk alone is not nearly as valuable as lemon with tea, even if milk is at hand. This means that  $f(t|l, m) > f(t|m)$  and  $f(l|t, m) > f(l|m)$  implying  $f(l, m) + f(m, t) < f(l, m, t) + f(m)$ . These are known as *supermodular* relationships, where the value increases as the context increases.

We have asked for a set of relationships amongst various subsets of the four items  $V = \{c, l, m, t\}$ , Is there a function that offers a value to each  $X \subseteq V$  that satisfies all of the above relationships? Figure 6.14 in fact shows such a function. On the left, we see a two-dimensional square whose vertices indicate the values over subsets of  $\{c, t\}$  and we can quickly verify that the sum of the blue boxes on north-west (corresponding to  $f(\{c\})$ ) and south-east corners (corresponding to  $f(\{t\})$ ) is greater than the sum of the north-east and south-west corners, expressing the required submodular relationship.

1 Next on the right is a three-dimensional cube that adds the relationship with lemon. Now we have  
 2 six squares and we see that the values at each of the vertices all satisfy the above requirements  
 3 — we verify this by considering the valuations at the four corners of every one of the six faces of  
 4 the cube. Since  $|V| = 4$  we need a four-dimensional hypercube to show all values and this may be  
 5 shown in two ways. It is first shown as a tesseract, a well-known three-dimensional projection of a  
 6 four-dimensional hypercube. In the figure, all vertices are labeled both with subsets of  $V$  as well as  
 7 the function value  $f(X)$  as the blue number in a box. The figure on the right shows a *lattice* version  
 8 of the four-dimensional hypercube, where corresponding three-dimensional cubes are shown in green  
 9 and red.  
 10

11 We thus see that a set function is defined for all subsets of a ground set, and that they correspond  
 12 to valuations at all vertices of the hypercube. For the particular function over valuations of subsets  
 13 of coffee, lemon, milk, and tea, we have seen submodular, supermodular, and modular relationships  
 14 all in one function. Therefore, the overall function  $f$  defined in Figure 6.14 is neither submodular,  
 15 supermodular, nor modular. For combinatorial auctions, there is often a desire to have a diversity  
 16 of such manners of relationships [LLN06] — representation of these relationships can be handled  
 17 by a difference of submodular functions [NB05; IB12] or a sum of a submodular and supermodular  
 18 function [BB18] (further described below). In machine learning, however, most of the time we are  
 19 interested in functions that are submodular (or modular, or supermodular) everywhere.

20

### 21 6.11.2 Submodular Basic Definitions

22 For a function to be submodular, it must satisfy the submodular relationship for all subsets. We  
 23 arrive at the following definition.  
 24

25 **Definition 6.11.1 (Submodular Function).** A given set function  $f : 2^V \rightarrow \mathbb{R}$  is submodular if for  
 26 all  $X, Y \subseteq V$ , we have the following inequality:

27

$$28 \quad f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y) \tag{6.261}$$

29

30 There are also many other equivalent definitions of submodularity [Bil22] some of which are more  
 31 intuitive and easier to understand. For example, submodular functions are those set functions that  
 32 satisfy the property of diminishing returns. If we think of a function  $f(X)$  as measuring the value of  
 33 a set  $X$  that is a subset of a larger set of data items  $X \subseteq V$ , then the submodular property means  
 34 that the incremental “value” of adding a data item  $v$  to set  $X$  decreases as the size of  $X$  grows. This  
 35 gives us a second classic definition of submodularity.

36 **Definition 6.11.2 (Submodular Function via Diminishing Returns).** A given set function  $f : 2^V \rightarrow \mathbb{R}$   
 37 is submodular if for all  $X, Y \subseteq V$ , where  $X \subseteq Y$  and for all  $v \notin Y$ , we have the following inequality:  
 38

39

$$f(X + v) + f(X) \geq f(Y + v) + f(Y) \tag{6.262}$$

40

41 The property that the incremental value of lemon with tea is less than the incremental value  
 42 of lemon once milk is already in the tea is equivalent to Equation 6.261 if we set  $X = \{m, t\}$  and  
 43  $Y = \{l, t\}$  (i.e.  $f(m, t) + f(l, t) > f(l, m, t) + f(t)$ ). It is naturally also equivalent to Equation 6.262  
 44 if we set  $X = \{t\}$ ,  $Y = \{m, t\}$ , and with  $v = l$  (i.e.,  $f(l|m, t) < f(l|t)$ ).

45 There are many functions that are submodular, one famous one being Shannon entropy seen  
 46 as a function of subsets of random variables. We first point out that there are non-negative (i.e.,  
 47

$f(A) \geq 0, \forall A$ ), monotone non-decreasing (i.e.,  $f(A) \leq f(B)$  whenever  $A \subseteq B$ ) submodular functions that are not entropic [Yeu91b; ZY97; ZY98], so submodularity is not just a trivial restatement of the class of entropy functions. When a function is monotone non-decreasing, submodular, and *normalized* so that  $f(\emptyset) = 0$ , it is often referred to as a **polymatroid function**. Thus, while the entropy function is a polymatroid function, it does not encompass all polymatroid functions even though all polymatroid functions satisfy the properties Claude Shannon mentioned as being natural for an “information” function (see Section 6.11.7).

A function  $f$  is supermodular if and only if  $-f$  is submodular. If a function is both submodular and supermodular, it is known as a *modular* function. It is always the case that modular functions may take the form of a vector-scalar pair  $(m, c)$  where  $m : 2^V \rightarrow \mathbb{R}$  and where  $c \in \mathbb{R}$  is a constant, and where for any  $A \subseteq V$ , we have that  $m(A) = c + \sum_{v \in A} m_v$ . If the modular function is normalized, so that  $m(\emptyset) = 0$ , then  $c = 0$  and the modular function can be seen simply as a vector  $m \in \mathbb{R}^V$ . Hence, we sometimes say that the modular function  $x \in \mathbb{R}^V$  offers a value for set  $A$  as the partial sum  $x(A) = \sum_{v \in A} x(v)$ . Many combinatorial problems use modular functions as objectives. For example, the graph cut problem uses modular function defined over the edges, judges a cut in a graph as the modular function applied to the edges that comprise the cut.

As can be seen from the above, and by considering Figure 6.14, a submodular function, and in fact any set function,  $f : 2^V \rightarrow \mathbb{R}$  can be seen as a function defined only on the vertices of the  $n$ -dimensional unit hypercube  $[0, 1]^n$ . Given any set  $X \subseteq V$ , we define  $\mathbf{1}_X \in \{0, 1\}^V$  to be the characteristic vector of set  $X$  defined as  $\mathbf{1}_X(v) = 1$  if  $v \in X$  and  $\mathbf{1}_X(v) = 0$  otherwise. This gives us a way to map from any set  $X \subseteq V$  to a binary vector  $\mathbf{1}_X$ . We also see that  $\mathbf{1}_X$  is itself a modular function since  $\mathbf{1}_X \in \{0, 1\}^V \subset \mathbb{R}^V$ .

Submodular functions share a number of properties in common with both convex and concave functions [Lov83], including wide applicability, generality, multiple representations, and closure under a number of common operators (including mixtures, truncation, complementation, and certain convolutions). There is one important submodular closure property that we state here — that is that if we take a non-negative weighted (or conical) combinations of submodular functions, we preserve submodularity. In other words, if we have a set of  $k$  submodular functions,  $f_i : 2^V \rightarrow \mathbb{R}$ ,  $i \in [k]$ , and we form  $f(X) = \sum_{i=1}^k \omega_i f_i(X)$  where  $\omega_i \geq 0$  for all  $i$ , then Definition 6.11.1 immediately implies that  $f$  is also submodular. When we consider Definition 6.11.1, we see that submodular functions live in a cone in  $2^n$ -dimensional space defined by the intersection of an exponential number of half-spaces each one of which is defined by one of the inequalities of the form  $f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y)$ . Each submodular function is therefore a point in that cone. It is therefore not surprising that taking conical combinations of such points stays within this cone.

### 6.11.3 Example Submodular Functions

As mentioned above, there are many functions that are submodular besides entropy. Perhaps the simplest such function is  $f(A) = \sqrt{|A|}$  which is the composition of the square-root function (which is concave) with the cardinality  $|A|$  of the set  $A$ . The gain function is  $f(A + v) - f(A) = \sqrt{k+1} - \sqrt{k}$  if  $|A| = k$ , which we know to be a decreasing in  $k$ , thus establishing the submodularity of  $f$ . In fact, if  $\phi : \mathbb{R} \rightarrow \mathbb{R}$  is any concave function, then  $f(A) = \phi(|A|)$  will be submodular for the same

1 reason.<sup>5</sup> 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  $\phi_u$  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  $\phi_u$ , any addition to the argument  $m_u(A)$  by adding, say,  $v$  to  $A$  would diminish as  $A$  gets larger. In  
 15 order to produce a set larger than  $A$  that has a much larger valuation, one must use a feature  $u' \neq u$   
 16 that has not yet diminished as much.

17 *Facility location* is another well-known submodular function — perhaps an appropriate nickname  
 18 would be the “ $k$ -means of submodular functions,” due to its applicability, utility, ease-of-use (it  
 19 needs only an affinity matrix), and similarity to  $k$ -medoids problems. The facility location function  
 20 is defined using an affinity matrix as follows:  $f(A) = \sum_{v \in V} \max_{a \in A} \text{sim}(a, v)$  where  $\text{sim}(a, v)$  is a  
 21 non-negative measure of the affinity (or similarity) between element  $a$  and  $v$ . Here, every element  
 22  $v \in V$  must have a representative within the set  $A$  and the representative for each  $v \in V$  is chosen  
 23 to be the element  $a \in A$  most similar to  $v$ . This function is also a form of dispersion or diversity  
 24 function because, in order to maximize it, every element  $v \in V$  must have some element similar to  
 25 it in  $A$ . The overall score is then the sum of the similarity between each element  $v \in V$  and  $v$ 's  
 26 representative. This function is monotone (since as  $A$  includes more elements to become  $B \supseteq A$ , it is  
 27 possible only to find an element in  $B$  more similar to a given  $v$  than an element in  $A$ ).

28 While the facility location looks quite different from a feature based function, it is possible  
 29 to precisely represent any facility location function with a feature based function. Consider  
 30 just  $\max_{a \in A} x_a$  and, without loss of generality, assume that  $0 \leq x_1 \leq x_2 \leq \dots \leq x_n$ . Then  
 31  $\max_{a \in A} x_a = \sum_{i=1}^n y_i \min(|A \cap \{i, i+1, \dots, n\}|, 1)$  where  $y_i = x_i - x_{i-1}$  and we set  $x_0 = 0$ . We note  
 32 that this is a sum of weighted concave composed with modular functions since  $\min(\alpha, 1)$  is concave  
 33 in  $\alpha$ , and  $|A \cap \{i, i+1, \dots, n\}|$  is a modular function in  $A$ . Thus, the facility location function, a  
 34 sum of these, is merely a feature based function.

35 Feature based functions, in fact, are quite expressive, and can be used to represent many different  
 36 submodular functions including set cover and graph-based functions. For example, we can define a *set*  
 37 *cover function*, given a set of sets  $\{U_v\}_{v \in V}$ , via  $f(X) = |\bigcup_{v \in X} U_v|$ . If  $f(X) = |U|$  where  $U = \bigcup_{v \in V} U_v$   
 38 then  $X$  indexes a set that fully covers  $U$ . This can also be represented as  $f(X) = \sum_{u \in U} \min(1, m_u(X))$   
 39 where  $m_u(X)$  is a modular function where  $m_u(v) = 1$  if and only if  $u \in U_v$  and otherwise  $m_u(v) = 0$ .  
 40 We see that this is a feature based submodular function since  $\min(1, x)$  is concave in  $x$ , and  $U$  is a  
 41 set of features.

42 This construct can be used to produce the vertex cover function if we set  $U = V$  to be the set of

43

44

45 5. While we will not be extensively discussing supermodular functions in this section,  $f(A) = \phi(|A|)$  is supermodular  
 46 for any convex function  $\phi$ .

47

vertices in a graph, and set  $m_u(v) = 1$  if and only if vertices  $u$  and  $v$  are adjacent in the graph and otherwise set  $m_u(v) = 0$ . Similarly, the edge cover function can be expressed by setting  $V$  to be the set of edges in a graph,  $U$  to be the set of vertices in the graph, and  $m_u(v) = 1$  if and only edge  $v$  is incident to vertex  $u$ .

A generalization of the set cover function is the *probabilistic coverage* function. Let  $P[B_{u,v} = 1]$  be the probability of the presence of feature (or concept)  $u$  within element  $v$ . Here, we treat  $B_{u,v}$  as a Bernoulli random variable for each element  $v$  and feature  $u$  so that  $P[B_{u,v} = 1] = 1 - P[B_{u,v} = 0]$ . Then we can define the probabilistic coverage function as  $f(X) = \sum_{u \in U} f_u(X)$  where, for feature  $u$ , we have  $f_u(X) = 1 - \prod_{v \in X} (1 - P[B_{u,v} = 1])$  which indicates the degree to which feature  $u$  is “covered” by  $X$ . If we set  $P[B_{u,v} = 1] = 1$  if and only if  $u \in U_v$  and otherwise  $P[B_{u,v} = 1] = 0$ , then  $f_u(X) = \min(1, m_u(X))$  and the set cover function can be represented as  $\sum_{u \in U} f_u(X)$ . We can generalize this in two ways. First, to make it softer and more probabilistic we allow  $P[B_{u,v} = 1]$  to be any number between zero and one. We also allow each feature to have a non-negative weight. This yields the general form of the probabilistic coverage function, which is defined by taking a weighted combination over all features:  $f_u(X) = \sum_{u \in U} \omega_u f_u(X)$  where  $\omega_u \geq 0$  is a weight for feature  $u$ . Observe that  $1 - \prod_{v \in X} (1 - P[B_{u,v} = 1]) = 1 - \exp(-m_u(X)) = \phi(m_u(X))$  where  $m_u$  is a modular function with evaluation  $m_u(X) = \sum_{v \in X} \log(1/(1 - P[B_{u,v} = 1]))$  and for  $z \in \mathbb{R}$ ,  $\phi(z) = 1 - \exp(-z)$  is a concave function. Thus, the probabilistic coverage function (and its set cover specialization) is also feature based function.

Another common submodular function is the graph cut function. Here, we measure the value of a subset of  $V$  by the edges that cross between a set of nodes and all but that set of nodes. We are given an undirected non-negative weighted graph  $\mathcal{G} = (V, E, w)$  where  $V$  is the set of nodes,  $E \subseteq V \times V$  is the set of edges, and  $w \in \mathbb{R}_+^E$  are non-negative edge weights corresponding to symmetric matrix (so  $w_{i,j} = w_{j,i}$ ). For any  $e \in E$ , we have  $e = \{i, j\}$  for some  $i, j \in V$  with  $i \neq j$ , the graph cut function  $f : 2^V \rightarrow \mathbb{R}$  is defined as  $f(X) = \sum_{i \in X, j \in \bar{X}} w_{i,j}$  where  $w_{i,j} \geq 0$  is the weight of edge  $e = \{i, j\}$  ( $w_{i,j} = 0$  if the edge does not exist), and where  $\bar{X} = V \setminus X$  is the complement of set  $X$ . Notice that we can write the graph cut function as follows:

$$f(X) = \sum_{i \in X, j \in \bar{X}} w_{i,j} = \sum_{i, j \in V} w_{i,j} \mathbf{1}\{i \in X, j \in \bar{X}\} \quad (6.263)$$

$$= \frac{1}{2} \sum_{i, j \in V} w_{i,j} \min(|X \cap \{i, j\}|, 1) + \frac{1}{2} \sum_{i, j \in V} w_{i,j} \min(|(V \setminus X) \cap \{i, j\}|, 1) - \frac{1}{2} \sum_{i, j \in V} w_{i,j} \quad (6.264)$$

$$= \tilde{f}(X) + \tilde{f}(V \setminus X) - \tilde{f}(V) \quad (6.265)$$

where  $\tilde{f}(X) = \frac{1}{2} \sum_{i, j \in V} w_{i,j} \min(|X \cap \{i, j\}|, 1)$ . Therefore, since  $\min(\alpha, 1)$  is concave, and since  $m_{i,j}(X) = |X \cap \{i, j\}|$  is modular,  $\tilde{f}(X)$  is submodular for all  $i, j$ . Also, since  $\tilde{f}(X)$  is submodular, so is  $\tilde{f}(V \setminus X)$  (in  $X$ ). Therefore, the graph cut function can be expressed as a sum of non-normalized feature-based functions. Note that here the second modular function is not normalized and is non-increasing, and also we subtract the constant  $\tilde{f}(V)$  to achieve equality.

Another way to view the graph cut function is to consider the non-negative weights as a modular function defined over the edges. That is, we view  $w \in \mathbb{R}_+^E$  as a modular function  $w : 2^E \rightarrow \mathbb{R}_+$  where for every  $A \subseteq E$ ,  $w(A) = \sum_{e \in A} w(e)$  is the weight of the edges  $A$  where  $w(e)$  is the weight of edge  $e$ . Then the graph cut function becomes  $f(X) = w(\{(a, b) \in E : a \in X, b \in X \setminus X\})$ . We view  $\{(a, b) \in E : a \in X, b \in X \setminus X\}$  as a set-to-set mapping function, that maps subsets of nodes to

1 subsets of edges, and the edge weight modular function  $w$  measures the weight of the resulting edges.  
 2 This immediately suggests that other functions can measure the weight of the resulting edges as  
 3 well, including non-modular functions. One example is to use a polymatroid function itself leading  
 4  $h(X) = g(\{(a, b) \in E : a \in X, b \in X \setminus X\})$  where  $g : 2^E \rightarrow \mathbb{R}_+$  is a submodular function defined  
 5 on subsets of edges. The function  $h$  is known as the **cooperative cut** function, and it is neither  
 6 submodular nor supermodular in general but there are many useful and practical algorithms that  
 7 can be used to optimize it [JB16a] thanks to its internal yet exposed and thus available to exploit  
 8 submodular structure.

9 While feature based functions are flexible and powerful, there is a strictly broader class of  
 10 submodular functions, unable to be expressed by feature-based functions, and that are related to deep  
 11 neural networks. Here, we create a recursively nested composition of concave functions with sums  
 12 of compositions of concave functions. An example is  $f(A) = \phi(\sum_{u \in U} \omega_u \phi_u(\sum_{a \in A} m_{u,a}))$ , where  $\phi$   
 13 is an outer concave function composed with a feature based function, with  $m_{u,a} \geq 0$  and  $\omega_u \geq 0$ .  
 14 This is known as a two-layer **deep submodular function** (DSF). A three-layer DSF has the form  
 15  $f(A) = \phi(\sum_{c \in C} \omega_c \phi_c(\sum_{u \in U} \omega_{u,c} \phi_u(\sum_{a \in A} m_{u,a})))$ . DSFs strictly expand the class of submodular  
 16 functions beyond feature based functions, meaning that there are feature based functions that can  
 17 not[BB17] represent deep submodular functions, even simple ones.

18

#### 19 20 6.11.4 Submodular Optimization

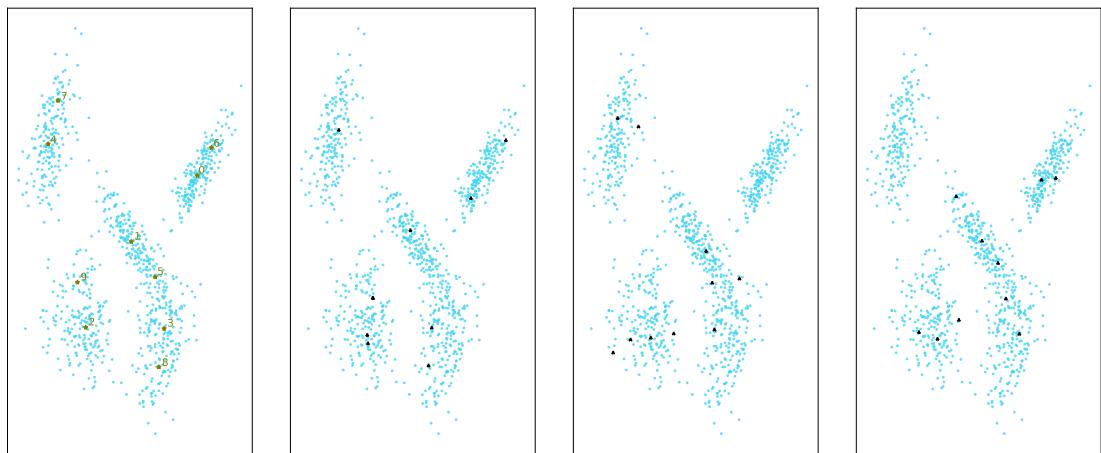
21

22 Submodular functions, while discrete, would not be very useful if it was not possible to optimize  
 23 over them efficiently. There are many natural problems in machine learning that can be cast as  
 24 submodular optimization and that can be addressed relatively efficiently.

25 When one wishes to encourage diversity, information, spread, high complexity, independence,  
 26 coverage, or dispersion, one usually will maximize a submodular function, in the form of  $\max_{A \in \mathcal{C}} f(A)$   
 27 where  $\mathcal{C} \subseteq 2^V$  is a constraint set, a set of subsets we are willing to accept as feasible solutions (more  
 28 on this below).

29 Why is submodularity, in general, a good model for diversity? Submodular functions are such  
 30 that once you have some elements, any other elements not in your possession but that are similar  
 31 to, explained by, or represented by the elements in your possession become less valuable. Thus, in  
 32 order to maximize the function, one must choose other elements that are dissimilar to, or not well  
 33 represented by, the ones you already have. That is, the elements similar to the ones you own are  
 34 diminished in value relative to their original values, while the elements dissimilar to the ones you  
 35 have do not have diminished value relative to their original values. Thus, maximizing a submodular  
 36 function successfully involves choosing elements that are jointly dissimilar amongst each other, which  
 37 is a definition of diversity. Diversity in general is a critically important aspect in machine learning  
 38 and artificial intelligence. For example, bias in data science and machine learning can often be seen  
 39 as some lack of diversity somewhere. Submodular functions have the potential to encourage (and  
 40 even ensure) diversity, enhance balance, and reduce bias in artificial intelligence.

41 Note that in order for a submodular function to appropriately model diversity, it is important  
 42 for it to be instantiated appropriately. Figure 6.15 shows an example in two dimensions. The plot  
 43 compares the ten points chosen according to a facility location instantiated with a Gaussian kernel,  
 44 along with the random samples of size ten. We see that the facility location selected points are more  
 45 diverse and tend to cover the space much better than any of the randomly selected points each of  
 46 which miss large regions of the space and/or show cases where points near each other are jointly  
 47



*Figure 6.15: Far Left: cardinality constrained (to ten) submodular maximization of a facility location function over 1000 points in two dimensions. Similarities are based on a Gaussian kernel  $\text{sim}(a, v) = \exp(-d(a, v))$  where  $d(\cdot, \cdot)$  is a distance. Selected points are green stars and the greedy order is also shown next to each selected point. Right three plots: different uniformly-at-random subsets of size ten.*

selected.

When one wishes for homogeneity, conformity, low complexity, coherence, or cooperation, one will usually minimize a submodular function, in the form of  $\min_{A \in \mathcal{C}} f(A)$ . For example, if  $V$  is a set of pixels in an image, one might wish to choose a subset of pixels corresponding to a particular object over which the properties (i.e., color, luminance, texture) are relatively homogeneous. Finding a set  $X$  of size  $k$ , even if  $k$  is large, need not have a large valuation  $f(X)$ , in fact it could even have the least valuation. Thus, semantic image segmentation could work even if the object being segmented and isolated consists of the majority of image pixels.

#### 6.11.4.1 Submodular Maximization

While the cardinality constrained submodular maximization problem is NP complete [Fei98], it was shown in [NWF78; FNW78] that the very simple and efficient greedy algorithm finds an approximate solution guaranteed to be within  $1 - 1/e \approx 0.63$  of the optimal solution. Moreover, the approximation ratio achieved by the simple greedy algorithm is provably the best achievable in polynomial time, assuming  $P \neq NP$  [Fei98]. The greedy algorithm proceeds as follows: Starting with  $X_0 = \emptyset$ , we repeat the following greedy step for  $i = 0 \dots (k - 1)$ :

$$X_{i+1} = X_i \cup \left( \underset{v \in V \setminus X_i}{\operatorname{argmax}} f(X_i \cup \{v\}) \right) \quad (6.266)$$

What the above approximation result means is that if  $X^* \in \operatorname{argmax}\{f(X) : |X| \leq k\}$ , and if  $\tilde{X}$  is the result of the greedy procedure, then  $f(\tilde{X}) \geq (1 - 1/e)f(X^*)$ .

The  $1 - 1/e$  guarantee is a powerful constant factor approximation result since it holds regardless of the size of the initial set  $V$  and regardless of which polymatroid function  $f$  is being optimized.

1 It is possible to make this algorithm run extremely fast using various acceleration tricks [FNW78;  
2 NWF78; Min78].

3 A minor bit of additional information about a polymatroid function, however, can improve  
4 the approximation guarantee. Define the total curvature if the polymatroid function  $f$  as  $\kappa =$   
5  $1 - \min_{v \in V} f(v|V - v)/f(v)$  where we assume  $f(v) > 0$  for all  $v$  (if not, we may prune them from the  
6 ground set since such elements can never improve a polymatroid function valuation). We thus have  
7  $0 \leq \kappa \leq 1$ , and [CC84] showed that the greedy algorithm gives a guarantee of  $\frac{1}{\kappa}(1 - e^{-\kappa}) \geq 1 - 1/e$ .  
8 In fact, this is an equality (and we get the same bound) when  $\kappa = 1$ , which is the fully curved case.  
9 As  $\kappa$  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  $\ell_i$  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  $\ell_i$  for each  $i$ . I.e., consider  
41

$$\mathcal{I}_p = \{X : \forall i \in [m], |V_i \cap X| \leq \ell_i\}. \quad (6.267)$$

42 Then  $(V, \mathcal{I}_p)$  is a matroid. The corresponding submodular maximization problem is a natural  
43 generalization of the cardinality constraint in that, rather than having a fixed number of elements  
44 beyond which we are uninterested, the set of elements  $V$  is organized into groups, and here we have  
45

46

a fixed per-group limit beyond which we are uninterested. This is useful for fairness applications since the solution must be distributed over the blocks of the matroid. Still, there are many much more powerful types of matroids that one can use [Oxl11; GM12].

Regardless of the matroid, the problem  $\max_{X \in \mathcal{I}} f(X)$  can be solved, with a  $1/2$  approximation factor, using the same greedy algorithm as above [NWF78; FNW78]. Indeed, the greedy algorithm has an intimate relationship with submodularity, a fact that is well studied in some of the seminal works on submodularity [Edm70; Lov83; Sch04]. It is also possible to define constraints consisting of an *intersection of matroids*, meaning that the solution must be simultaneously independent in multiple distinct matroids. Adding on to this, we might wish a set to be independent in multiple matroids and also satisfy a knapsack constraint. Knapsack constraints are not matroid constraints, since there can be multiple maximal cost solutions that are not the same size (as must be the case in a matroid). It is also possible to define discrete constraints using level sets of another completely different submodular function [IB13] — given two submodular functions  $f$  and  $g$ , this leads to optimization problems of the form  $\max_{X \subseteq V: g(X) \leq \alpha} f(X)$  (the submodular cost submodular knapsack, or SCSK, problem) and  $\min_{X \subseteq V: g(X) \geq \alpha} f(X)$  (the submodular cost submodular cover, or SCSC, problem). Other examples include covering constraints [IN09], and cut constraints [JB16a]. Indeed, the type of constraints on submodular maximization for which good and scalable algorithms exist is quite vast, and still growing.

One last note on submodular maximization. In the above, the function  $f$  has been assumed to be a polymatroid function. There are many submodular functions that are not monotone [Buc+12]. One example we saw before, namely the graph cut function. Another example is the log of the determinant (log-determinant) of a submatrix of a positive-definite matrix (which is the Gaussian entropy plus a constant). Suppose that  $\mathbf{M}$  is an  $n \times n$  symmetric positive-definite (SPD) matrix, and that  $\mathbf{M}_X$  is a row-column submatrix (i.e., it is an  $|X| \times |X|$  matrix consisting of the rows and columns of  $\mathbf{M}$  consisting of the elements in  $X$ ). Then the function defined as  $f(X) = \log \det(\mathbf{M}_X)$  is submodular but not necessarily monotone non-decreasing. In fact, the submodularity of the log-determinant function is one of the reasons that *determinantal point processes* (DPPs), which instantiate probability distributions over sets in such a way that high probability is given to those subsets that are diverse according to  $\mathbf{M}$ , are useful for certain tasks where we wish to probabilistically model diversity [KT11a]. Diversity of a set  $X$  here is measured by the volume of the parallelepiped which is known to be computed as the determinant of the submatrix  $\mathbf{M}_X$  and taking the log of this volume makes the function submodular in  $X$ . A DPP in fact is an example of a log-submodular probabilistic model (more in Section 6.11.10).

#### 6.11.4.3 Submodular Function Minimization

In the case of a polymatroid function, unconstrained minimization is again trivial. However, even in the unconstrained case, the minimization of an arbitrary (i.e., not necessarily monotone) submodular function  $\min_{X \subseteq V} f(X)$  might seem hopelessly intractable. Unconstrained submodular maximization is NP-hard (albeit approximable) and this is not surprising given that there are an exponential number of sets needing to be considered. Remarkably, submodular minimization does not require exponential computation, is not NP-hard, and in fact, there are polynomial time algorithms for doing so, something that is not at all obvious. This is one of the important characteristics that submodular functions share with convex functions, their common amenability to minimization. Starting in the very late 1960s, and spearheaded by individuals such as Jack Edmonds [Edm70],

1 there was a concerted effort in the discrete mathematics community in search of either an algorithm  
2 that could minimize a submodular function in polynomial time or a proof that such a problem was  
3 NP-hard. The nut was finally cracked in a classic paper [GLS81] on the ellipsoid algorithm that gave  
4 a polynomial time algorithm for submodular function minimization (SFM). While the algorithm was  
5 polynomial, it was a continuous algorithm, and it was not practical, so the search continued for a  
6 purely combinatorial strongly polynomial time algorithm. Queyranne [Que98] then proved that an  
7 algorithm [NI92] worked for this problem when the set function also satisfies a symmetry condition  
8 (i.e.,  $\forall X \subseteq V, f(X) = f(V \setminus X)$ ), which only requires  $O(n^3)$  time. The result finally came in around  
9 year 2000 using two mostly independent methods [IFF00; Sch00]. These algorithms, however, also  
10 were impractical in that while they are polynomial time, they have unrealistically high polynomial  
11 degree (i.e.,  $\tilde{O}(V^7 * \gamma + V^8)$  for [Sch00] and  $\tilde{O}(V^7 * \gamma)$  for [IFF00]). This led to additional work on  
12 combinatorial algorithms for SFM leading to algorithms that could perform SFM in time  $\tilde{O}(V^5\gamma + V^6)$   
13 in [IO09]. Two practical algorithms for SFM include the Fujishige-Wolfe procedure [Fuj05; Wol76]<sup>6</sup>  
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  $\epsilon$ -approximate minimization with a linear running time [ALS20] of  $\tilde{O}(n/\epsilon^2)$ . There  
23 have been other efforts to utilize parallelism to further improve SFM [BS20].  
24

25

### 26 6.11.5 Applications of Submodularity in Machine Learning and AI

27

28 Submodularity arises naturally in applications in machine learning and artificial intelligence, but its  
29 utility has still not yet been as widely recognized and exploited as other techniques. For example,  
30 while information theoretic concepts like entropy and mutual information are extremely widely used  
31 in machine learning (e.g., the cross entropy loss for classification is ubiquitous), the submodularity  
32 property of entropy is not nearly as widely explored.

33

34 Still, the last several decades, submodularity has been increasingly studied and utilized in the  
35 context of machine learning. The below we begin to provide only a brief survey of some of the major  
36 sub-areas within machine learning that have been touched by submodularity. The list is not meant  
37 to be exhaustive, or even extensive. It is hoped that the below should, at least, offer a reasonable  
38 introduction into how submodularity has been and can continue to be useful in machine learning and  
39 artificial intelligence.

40

### 41 6.11.6 Sketching, CoreSets, Distillation, and Data Subset & Feature Selection

42

43 A summary is a concise representation of a body of data that can be used as an effective and efficient  
44 substitute for that data. There are many types of summary and some are extremely simple. For  
45 example, the mean or median of a list of numbers summarizes some property (the central tendency)  
46 of that list. A random subset is also a form of summary.

47

---

46 6. This is the same Wolfe as the Wolfe in Frank-Wolfe but not the same algorithm.

Any given summary, however, is not guaranteed to do a good job serving all purposes. Moreover, a summary usually involves at least some degree of approximation and fidelity loss relative to the original, and different summaries are faithful to the original in different ways and for different tasks. For these and other reasons, the field of summarization is rich and diverse, and summarization procedures are often very specialized.

Several distinct names for summarization have been used over the past few decades, including “sketches”, “coresets”, (in the field of natural language processing) “summaries”, and “distillation.”

Sketches [Cor17; CY20; Cor+12], arose in the field of computer science and was based on the acknowledgment that data is often too large to fit in memory and too large for an algorithm to run on a given machine, something enabled by a much smaller but still representative, and provably approximate, representation of the data.

Coresets are similar to sketches and there are some properties that are more often associated with coresets than with sketches, but sometimes the distinction is a bit vague. The notion of a coreset [BHP02; AHP+05; BC08] comes from the field of computational geometry where one is interested in solving certain geometric problems based on a set of points in  $\mathbb{R}^d$ . For any geometric problem and a set of points, a coreset problem typically involves finding the smallest weighted subset of points so that when an algorithm is run on the weighted subset, it produces approximately the same answer as when it is run on the original large dataset. For example, given a set of points, one might wish to find the diameter of set, or the radius of the smallest enclosing sphere, or finding the narrowest annulus (ring) containing the points, or a subset of points whose  $k$ -center clustering is approximately the same as the  $k$ -center clustering of the whole [BHP02].

Document summarization became one of the most important problems in natural language processing (NLP) in the 1990s although the idea of computing a summary of a text goes back much further to the 1950s [Luh58; Edm69], also and coincidentally around the same time that the CliffsNotes [Wik21] organization began. There are two main forms of document summarization [YWX17]. With *extractive summarization* [NM12], a set of sentences (or phrases) are extracted from the documents needing to be summarized, and the resulting subset of sentences, perhaps appropriately ordered, comprises the summary.

With abstractive summarization [LN19], on the other hand, the goal is to produce an “abstract” of the documents, where one is not constrained to have any of the sentences in the abstract correspond to any of the sentences in the original documents. With abstractive summarization, therefore, the goal is to synthesize a small set of new pseudo sentences that represent the original documents. CliffsNotes, for example, are abstractive summaries of the literature being represented.

Another form of summarization that has more recently become popular in the machine learning community is *data distillation* [SG06b; Wan+20b; Suc+20; BYH20; NCL20; SS21; Ngu+21] or equivalently *dataset condensation* [ZMB21; ZB21]. With data distillation<sup>7</sup>, 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

<sup>7</sup> 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

use of a deep submodular function and the latter does this in an unsupervised label-free fashion); [SS18a] which is a form of robust submodularity for producing coresets for training CNNs; [Kau+19] which uses a facility location to facilitate diversity selection in active learning; [Bai+15; CTN17] which develops a mixture of submodular functions for document summarization where the mixture coefficients are also included in the hyperparameter set; [Xu+15] uses a symmetrized submodular function for the purposes of video summarization.

The learnability and identifiability of submodular functions has received a good amount of study from a theoretical perspective. Starting with the strictest learning settings, the problem looks pretty dire. For example, [SF08; Goe+09] shows that if one is restricted to making a polynomial number of queries (i.e., training pairs of the form  $(S, f(S))$ ) of a monotone submodular function, then it is not possible to approximate  $f$  with a multiplicative approximation factor better than  $\tilde{\Omega}(\sqrt{n})$ . In [BH11], goodness is judged multiplicatively, meaning for a set  $A \subseteq V$  we wish that  $\tilde{f}(A) \leq f(A) \leq g(n)f(A)$  for some function  $g(n)$ , and this is typically a probabilistic condition (i.e., measured by distribution, or  $\tilde{f}(A) \leq f(A) \leq g(n)f(A)$ , should happen on a fraction at least  $1 - \beta$  of the points). Alternatively, goodness may also be measured by an additive approximation error, say by a norm. I.e., defining  $\text{err}_p(f, \tilde{f}) = \|f - \tilde{f}\|_p = (E_{A \sim \mathbf{P}_f} [|f(A) - \tilde{f}(A)|^p])^{1/p}$ , we may wish  $\text{err}_p(f, \tilde{f}) < \epsilon$  for  $p = 1$  or  $p = 2$ . In the PAC (probably approximately correct) model, we probably ( $\delta > 0$ ) approximately ( $\epsilon > 0$  or  $g(n) > 1$ ) learn ( $\beta = 0$ ) with a sample or algorithmic complexity that depends on  $\delta$  and  $g(n)$ . In the PMAC (probably mostly approximately correct) model [BH11], we also “mostly”  $\beta > 0$  learn. In some cases we wish to learn the best submodular approximation to a non-submodular function. In other cases, we are allowed to deviate from submodularity as long as the error is small. Learning special cases includes coverage functions [FK14; FK13a], and low-degree polynomials [FV15], curvature limited functions [IJB13], functions with a limited “goal” [DHK14; Bac+18], functions that are Fourier sparse [Wen+20a], or that are of a family called “juntas” [FV16], or that come from families other than submodular [DFF21], and still others [BRS17; FKV14; FKV17; FKV20; FKV13; YZ19]. Other results include that one can not minimize a submodular function by learning it first from samples [BS17]. The essential strategy of learning is to attempt to construct a submodular function approximation  $\hat{f}$  from an underlying submodular function  $f$  querying the latter only a small number of times. The overall gist of these results is that it is hard to learn everywhere and accurately.

In the machine learning community, learning can be performed extremely efficiently in practice, although there are not the types of guarantees as one finds above. For example, given a mixture of submodular components of the form  $f(A) = \sum_i \alpha_i f_i(A)$ , if each  $f_i$  is considered fixed, then the learning occurs only over the mixture coefficients  $\alpha_i$ . This can be solved as a linear regression problem where the optimal coefficients can be computed in a linear regression setting. Alternatively, such functions can be learnt in a max-margin setting where the goal is primarily to adjust  $\alpha_i$  to ensure that  $f(A)$  is large on certain subsets [SSJ12; LB12; Tsc+14]. Even here there are practical challenges, however, since it is in general hard in practice to obtain a training set of pairs  $\{(S_i, F(S_i))\}_i$ . Alternatively, one also “learn” a submodular function in a reinforcement learning setting [CKK17] by optimizing the implicit function directly from gain vectors queried from an environment. In general, such practical learning algorithms have been used for image summarization [Tsc+14], document summarization [LB12], and video summarization [GGG15; Vas+17a; Gon+14; SGS16; SLG17]. While none of these learning approaches claim to approximate some true underlying submodular function, in practice, they do perform better than the by-hand crafting of a submodular functions mentioned above.

By a submodularity based coresset, we mean one where the direct optimization of a submodular

1 function offers a theoretical guarantee for some specific problem. This is distinct from above where  
 2 the submodular function is used as a surrogate heuristic objective function and for which, even if the  
 3 submodular function is learnt, optimizing it is only a heuristic for the original problem. In some  
 4 limited cases, it can be shown that the function we wish to approximate is already submodular,  
 5 e.g., in the case of certain naive Bayes and k-NN classifiers [WIB15] where the training accuracy,  
 6 as a function of the training data subset, can be shown to be submodular. Hence, maximizing this  
 7 function offers the same guarantee on the training accuracy as it does on the submodular function.  
 8 Unfortunately, the accuracy function for many models are not submodular, although they do have a  
 9 difference of submodular [NB05; IB12] decomposition.

10 In other cases, it can be shown that certain desirable coresets objectives are inherently submodular.  
 11 For example, in [MBL20], it is shown that the normed difference between the overall gradient  
 12 (from summing over all samples in the training data) and an approximate gradient (from summing  
 13 over only samples in a summary) can be upper bounded with a supermodular function that, when  
 14 converted to a submodular facility location function and maximized, will select a set that reduces  
 15 this difference, and will lead to similar convergence rates to an approximate optimum solution in the  
 16 convex case. A similar example of this in a DPP context is shown in [TBA19]. In other cases, subsets  
 17 of the training data and training occur simultaneously using a continuous-discrete optimization  
 18 framework, where the goal is to minimize the loss on diverse and challenging samples measured by a  
 19 submodular objective [ZB18]. In still other cases, bi-level objectives related to but not guaranteed to  
 20 be submodular can be formed where a set is selected from a training set with the deliberate purpose  
 21 of doing well on a validation set [Kil+20; BMK20].

22 The methods above have focused on reducing the number of samples in a training data. Considering  
 23 the transpose of a design matrix, however, all of the above methods can be used for reducing the  
 24 features of a machine learning procedure as well. Specifically, any of the extractive summarization,  
 25 subset selection, or coresets methods can be seen as feature selection while any of the abstract  
 26 summarization, sketching, or distillation approaches can be seen as dimensionality reduction.

27

### 28 6.11.7 Combinatorial Information Functions

29 The entropy function over a set of random variables  $X_1, X_2, \dots, X_n$  is defined as  $H(X_1, X_2, \dots, X_n) =$   
 30  $-\sum_{x_1, x_2, \dots, x_n} p(x_1, \dots, x_n) \log p(x_1, \dots, x_n)$ . From this we can define three set-argument conditional  
 31 mutual information functions as  $I_H(A; B|C) = I(X_A; X_B|X_C)$  where the latter is the mutual  
 32 information between variables indexed by  $A$  and  $B$  given variables indexed by  $C$ . This mutual  
 33 information expresses the residual information between  $X_A$  and  $X_B$  that is not explained by their  
 34 common information with  $X_C$ .

35 As mentioned above, we may view any polymatroid function as a type of information function over  
 36 subsets of  $V$ . That is,  $f(A)$  is the information in set  $A$  — to the extent that this is true, this property  
 37 justifies  $f$ 's use as a summarization objective as mentioned above. The reason  $f$  may be viewed as an  
 38 information function stems from  $f$  being normalized,  $f$ 's non-negativity,  $f$ 's monotonicity, and the  
 39 property that further conditioning reduces valuation (i.e.,  $f(A|B) \geq f(A|B, C)$  which is identical to  
 40 the submodularity property). These properties were outlined as being essential to the entropy function  
 41 in Shannon's original work [Sha48] but are true of any polymatroid function as well. Hence, given any  
 42 polymatroid function  $f$ , it is possible to define a combinatorial mutual information function [Iye+21]  
 43 in a similar way. Specifically, we can define the combinatorial (submodular) conditional mutual  
 44 information (CCMI) as  $I_f(A; B|C) = f(A+C) + f(B+C) - f(C) - f(A+B+C)$ , which has been  
 45

46

known as the connectivity function [Cun83] amongst other names. If  $f$  is the entropy function then this yields the standard entropic mutual information but here the mutual information can be defined for any submodular information measure  $f$ . For an arbitrary polymatroid  $f$ , therefore,  $I_f(A; B|C)$  can be seen as an  $A, B$  set-pair similarity score that ignores, neglects, or discounts any common similarity between the  $A, B$  pair that is due to  $C$ .

Historical use of a special case of CCMF, i.e.,  $I_f(A; B)$  where  $C = \emptyset$ , occurred in a number of circumstances. For example, in [GKS05] the function  $g(A) = I_f(A; V \setminus A)$  (which, incidentally is both symmetric ( $g(A) = g(V \setminus A)$  for all  $A$ ) and submodular was optimized using the greedy procedure which has a guarantee as long as  $g(A)$  is monotone up  $2k$  elements whenever one wishes for a summary of size  $k$ . This was done for  $f$  being the entropy function, but it can be used for any polymatroid function. In similar work where  $f$  is Shannon entropy, [KG05] demonstrated that  $g_C(A) = I_f(A; C)$  for a fixed set  $C$  is not submodular in  $A$  but if it is the case that the elements of  $V$  are independent given  $C$  then submodularity is preserved. This can be seen quickly easily by the consequence of the assumption which states that  $I_f(A; C) = f(A) - f(A|C) = f(A) - \sum_{a \in A} f(a|C)$  where the second equality is due to the conditional independence property. In this case,  $I_f$  is the difference between a submodular and a modular function which preserves submodularity for any polymatroid  $f$ .

On the other hand, it would be useful for  $g_{B,C}(A) = I_f(A; B|C)$ , where  $B$  and  $C$  are fixed, to be possible to optimize in terms of  $A$ . One can view this function as one that, when it is maximized, chooses  $A$  to be similar to  $B$  in a way that neglects or discounts any common similarity that  $A$  and  $B$  have with  $C$ . One option to optimize this function to utilize difference of submodular [NB05; IB12] optimization as mentioned earlier. A more recent result shows that in some cases  $g_{B,C}(A)$  is still submodular in  $A$ . Define the second order partial derivative of a submodular function  $f$  as follows  $f(i, j|S) \triangleq f(j|S + i) - f(j|S)$ . Then if it is the case that  $f(i, j|S)$  is monotone non-decreasing in  $S$  for  $S \subseteq V \setminus \{i, j\}$  then  $I_f(A; B|C)$  is submodular in  $A$  for fixed  $B$  and  $C$ . It may be thought that only esoteric functions have this property but in fact [Iye+21] shows that this is true for a number of widely used submodular functions in practice, including the facility location function which results in the form  $I_f(A; B|C) = \sum_{v \in V} \max \left( \min \left( \sum_{a \in A} \text{sim}(v, a), \max_{b \in B} \text{sim}(v, b) \right) - \max_{c \in C} \text{sim}(v, c), 0 \right)$ . This function was used [Kot+22] to produce summaries  $A$  that were particularly relevant to a query given by  $B$  but that should neglect information in  $C$  that can be considered “private” information to avoid.

### 6.11.8 Clustering, Data Partitioning, and Parallel Machine Learning

There are an almost limited number of clustering algorithms and a plethora of reviews on their variants. Any given submodular function can also instantiate a clustering procedure as well, and there are several ways to do this. Here we offer only a brief outline of the approach. In the last section, we defined  $I_f(A; V \setminus A)$  as the CCMF between  $A$  and everything but  $A$ . When we view this as a function of  $A$ , then  $g(A) = I_f(A; V \setminus A)$  and  $g(A)$  is a symmetric submodular function that can be minimized using Queyranne’s algorithm [Que98; NI92]. Once this is done, the resulting  $A$  is such that it is least similar to  $V \setminus A$ , according to  $I_f(A; V \setminus A)$  and hence forms a 2-clustering. This process can then be recursively applied where we form two new functions  $g_A(B) = I_f(B; A \setminus B)$  for  $B \subseteq A$  and  $g_{V \setminus A}(B) = I_f(B; (V \setminus A) \setminus B)$  for  $B \subseteq V \setminus A$ . These are two symmetric submodular functions on different ground sets that also can be minimized using Queyranne’s algorithm. This

1 recursive bisection algorithm then repeats until the desired number of clusters is formed. Hence, the  
 2 CCMI function can be used as a top-down recursive bisection clustering procedure and has been  
 3 called Q-clustering [NJB05; NB06]. It should be noted that such forms of clustering often generalizes  
 4 forming a multi-way cut in an undirected graph in which case the objective becomes the graph-cut  
 5 function that, as we saw above, is also submodular. In some cases, the number of clusters need  
 6 not be specified in advance [NKI10]. Another submodular approach to clustering can be found  
 7 in [Wei+15b] where the goal is to minimize the maximum valued block in a partitioning which can  
 8 lead to submodular load balancing or minimum makespan scheduling [HS88; LST90].

10 Yet another form of clustering can be seen via the simple cardinality constrained submodular  
 11 maximization process itself which can be compared to a  $k$ -medoids process whenever the objective  $f$   
 12 is the facility location function. Hence, any such submodular function can be seen as a submodular-  
 13 function-parameterized form of finding the  $k$  “centers” among a set of data items. There have been  
 14 numerous applications of submodular clustering. For example, using these techniques it is possible to  
 15 identify parcellations of the human brain [Sal+17a]. Other applications include partitioning data for  
 16 more effective and accurate and lower variance distributed machine learning training [Wei+15a] and  
 17 also for more ideal mini-batch construction for training deep neural networks [Wan+19b].

18

### 19 6.11.9 Active and Semi-Supervised Learning

20

21 We are given data set  $\{x_i, y_i\}_{i \in V}$  consisting of  $|V| = n$  samples of  $x, y$  pairs but where the labels  
 22 are unknown. Samples are labeled one at a time or one mini-batch at a time, and after each  
 23 labeling step  $t$  each remaining unlabeled sample is given a score  $s_t(x_i)$  that indicates the potential  
 24 benefit of acquiring a label for that sample. Examples include the entropy of the model’s output  
 25 distribution on  $x_i$ , or a margin based score consisting of the difference between the top and the  
 26 second-from-the-top posterior probability (what is known as margin sampling [Set09]). This produces  
 27 a modular function on the unlabeled samples,  $m_t(A) = \sum_{a \in A} s_t(a)$  where  $A \subseteq V$ . It is simple to use  
 28 this modular function to produce a mini-batch active learning procedure where at each stage we form  
 29  $A_t \in \operatorname{argmax}_{A \subseteq U_t: |A|=k} m_t(A)$  where  $U_t$  is the set of labeled samples at stage  $t$ . Then  $A_t$  is a set of  
 30 size  $k$  that getgs labeled, we form  $U_t = U_t \setminus A_t$ , update  $s_t(a)$  for  $a \in U_t$  and repeat. The reason for  
 31 using active learning with mini-batches of size greater than one is that it is often inefficient to ask for  
 32 single label at a time. The problem which such a minibatch strategy, however, is that the set  $A_t$   
 33 can be redundant. The reason is that the uncertainty about every sample in  $A_t$  could be owing to  
 34 the same underlying cause — even though the model is most uncertain about samples in  $A_t$ , once  
 35 one sample in  $A_t$  is labeled, it may not be optimal to label the remaining samples in  $A_t$  due to this  
 36 redundancy. Utilizing submodularity, therefore, can help reduce this redundancy. Suppose  $f_t(A)$  is  
 37 a submodular diversity model over samples at step  $t$ . At each stage, choosing the set of samples  
 38 to label becomes  $A_t \in \operatorname{argmax}_{A \subseteq U_t: |A|=k} m_t(A) + f_t(A)$  —  $A_t$  is selected based on a combination  
 39 of both uncertainty (via  $m_t(A)$ ) and diversity (via  $f_t(A)$ ). This is precisely the submodular active  
 40 learning approach taken in [WIB15; Kau+19].

41 Another quite different approach to a form of submodular “batch” active learning setting where a  
 42 batch  $L$  of labeled samples are selected all at once and then used to label the rest of the unlabeled  
 43 samples. This also allows the remaining unlabeled samples to be utilized in a semi-supervised  
 44 framework [GB09; GB11]. In this setting, we start with a graph  $G = (V, E)$  where the nodes  $V$   
 45 need to be given a binary  $\{0, 1\}$ -valued label,  $y \in \{0, 1\}^V$ . For any  $A \subseteq V$  let  $y_A \in \{0, 1\}^A$  be  
 46 the labels just for node set  $A$ . We also define  $V(y) \subseteq V$  as  $V(y) = \{v \in V : y_v = 1\}$ . Hence  
 47

$V(y)$  are the graph nodes labeled 1 by  $y$  and  $V \setminus V(y)$  are the nodes labeled 0. Given submodular objective  $f$ , we form its symmetric CCMII variant  $I_f(A) \triangleq I_f(A; V \setminus A)$  — note that  $I_f(A)$  is always submodular in  $A$ . This allows  $I_f(V(y))$  to determine the “smoothness” of a given candidate labeling  $y$ . For example, if  $I_f$  is the weighted graph cut function where each weight corresponds to an affinity between the corresponding two nodes, then  $I_f(V(y))$  would be small if  $V(y)$  (the 1-labeled nodes) do not have strong affinity with  $V \setminus V(y)$  (the 0-labeled nodes). In general, however,  $I_f$  can be any symmetric submodular function. Let  $L \subseteq V$  be any candidate set of nodes to be labeled, and define  $\Psi(L) \triangleq \min_{T \subseteq (V \setminus L): T \neq \emptyset} I_f(T)/|T|$ . Then  $\Psi(L)$  measures the “strength” of  $L$  in that if  $\Psi(L)$  is small, an adversary can label nodes other than  $L$  without being too unsmooth according to  $I_f$ , while if  $\Psi(L)$  is large, an adversary can do no such thing. Then [GB11] showed that given a node set  $L$  to be queried, and the corresponding correct labels  $y_L$  that are completed (in a semi-supervised fashion) according to the following  $y' = \operatorname{argmin}_{\hat{y} \in \{0,1\}^V: \hat{y}_L = y_L} I_f(V(\hat{y}))$ , then this results in the following bound on the true labeling  $\|y - y'\|^2 \leq 2I_f(V(y))/\Psi(L)$  suggesting that we can find a good set to query by maximizing  $L$  in  $\Psi(L)$ , and this holds for any submodular function. Of course it is necessary to find an underlying submodular function  $f$  that fits a given problem, and this is discussed in Section 6.11.6.

### 6.11.10 Probabilistic Modeling

Graphical models are often used to describe factorization requirements on families of probability distributions. Factorization is not the only way, however, to describe restrictions on such families. In a graphical model, graphs describe only which random variable may directly interact with other random variable. An entirely different strategy for producing families of often-tractable probabilistic models can be produced without requiring any factorization property at all. Considering an energy function  $E(x)$  where  $p(x) \propto \exp(-E(x))$ , factorizations correspond to there being cliques in the graph such that the graph’s tree-width often is limited. On the other hand, finding  $\max_x p(x)$  is the same as finding  $\min_x E(x)$ , something that can be done if  $E(x) = f(V(x))$  is a submodular function (using the earlier used notation  $V(x)$  to map from binary vectors to subsets of  $V$ ). Even a submodular function as simple as  $f(A) = \sqrt{|A|} - m(A)$  where  $m$  is modular has tree-width of  $n - 1$ , and this leads to an energy function  $E(x)$  that allows  $\max_x p(x)$  to be solved in polynomial time using submodular function minimization (see Section 6.11.4.3). Such restrictions to  $E(x)$  therefore are not of the form *amongst the random variables, who is allowed to directly interact with whom*, but rather *amongst the random variables, what is the manner that they interact*. Such potential function restrictions can also combine with direct interaction restrictions as well and this has been widely used in computer vision, leading to cases where graph-cut and graph-cut like “move making” algorithms (such as *alpha-beta* swap and *alpha*-expansion algorithms) used in attractive models [BVZ99; BK01; BVZ01; SWW08]. In fact, the culmination of these efforts [KZ02] lead to a rediscovery of the submodularity (or the “regular” property) as being the essential ingredient for when Markov random fields can be solved using graph cut minimization, which is a special case of submodular function minimization.

The above model can be seen as log-supermodular since  $\log p(x) = -E(x) + \log 1/Z$  is a supermodular function. These are all distributions that put high probability on configurations that yield small valuation by a submodular function. Therefore, these distributions have high probability when  $x$  consists of a homogeneous and for this reason they are useful for computer vision segmentation problems (e.g., in a segment of an image, the nearby pixels should roughly be homogeneous as that is often what defines an object). The DPPs we saw above, however, are an example of a

1 log-submodular probability distribution since  $f(X) = \log \det(\mathbf{M}_X)$  is submodular. These models  
2 have high probability for diverse sets.

3 More generally,  $E(x)$  being either a submodular or supermodular function can produce log-  
4 submodular or log-supermodular distributions, covering both cases above where the partition function  
5 takes the form  $Z = \sum_{A \subseteq V} \exp(f(A))$  for objective  $f$ . Moreover, we often wish to perform tasks much  
6 more than just finding the most probable random variable assignments. This includes marginalization,  
7 computing the partition function, constrained maximization, and so on. Unfortunately, many of these  
8 more general probabilistic inference problems do not have polynomial time solutions even though  
9 the objectives are submodular or supermodular. On the other hand, such structure has opened the  
10 doors to an assortment of new probabilistic inference procedures that exploit this structure [DK14;  
11 DK15a; DTK16; ZDK15; DJK18]. Most of these methods were of the variational sort and offered  
12 bounds on the partition function  $Z$ , sometimes making use of the fact that submodular functions  
13 have easily computable semi-gradients [IB15; Fuj05] which are modular upper and lower bounds on a  
14 submodular or supermodular function that are tight at one or more subsets. Given a submodular (or  
15 supermodular) function  $f$  and a set  $A$ , it is possible to easily construct (in linear time) a modular  
16 function upperbound  $m^A : 2^V \rightarrow \mathbb{R}$  and a modular function lower bound  $m_A : 2^V \rightarrow \mathbb{R}$  having  
17 the properties that  $m_A(X) \leq f(X) \leq m^A(X)$  for all  $X \subseteq V$  and that is tight at  $X = A$  meaning  
18  $m_A(A) = f(A) = m^A(A)$  [IB15]. For any modular function  $m$ , the probability function for a  
19 characteristic vector  $x = \mathbf{1}_A$  becomes  $p(\mathbf{1}_A) = 1/Z \exp(E(\mathbf{1}_A)) = \prod_{a \in A} \sigma(m(a)) \prod_{a \notin A} \sigma(-m(a))$   
20 where  $\sigma$  is the logistic function. Thus, a modular approximation of a submodular function is like a  
21 mean-field approximation of the distribution, and makes the assumption that all random variables  
22 are independent. Such an approximation can then be used to compute quantities such as upper and  
23 lower bounds on the partition function, and much else.

25

### 26 6.11.11 Structured Norms and Loss Functions

27

28 Convex norms are used ubiquitously in machine learning, often as complexity penalizing regularizers  
29 (e.g., the ubiquitous  $p$ -norms for  $p \geq 1$ ) and also sometimes as losses (e.g., squared error). Identifying  
30 new useful structured and possibly learnable sparse norms is an interesting and useful endeavor,  
31 and submodularity can help here as well. Firstly, recall the  $\ell_0$  or counting norm  $\|x\|_0$  simply counts  
32 the number of nonzero entries in  $x$ . When we wish for a sparse solution, we may wish to regularize  
33 using  $\|x\|_0$  but it both leads to an intractable combinatorial optimization problem and it leads to an  
34 object that is not differentiable. The usual approach is to find the closest convex relaxation of this  
35 norm and that is the one norm or  $\|x\|_1$ . This is convex in  $x$  and has a sub-gradient structure and  
36 hence can be combined with a loss function to produce an optimizable machine learning objective,  
37 for example the lasso. On the other hand  $\|x\|_1$  has no structure, as each element of  $x$  is penalized  
38 based on its absolute value irrespective of the state of any of the other elements. There have thus  
39 been efforts to develop group norms that penalize groups or subsets of elements of  $x$  together, such  
40 as group lasso [HTW15].

41 It turns out that there is a way to utilize a submodular function as the regularizer. Penalizing  
42  $x$  via  $\|x\|_0$  is identical to penalizing it via  $|V(x)|$  and note that  $m(A) = |A|$  is a modular function.  
43 Instead, we could penalize  $x$  via  $f(V(x))$  for a submodular function  $f$ . Here, any element of  $x$   
44 being non-zero would allow for a diminishing penalty of other elements of  $x$  being zero all according  
45 to the submodular function, and such cooperative penalties can be obtained via a submodular  
46 parameterization. Like when using the zero-norm  $\|x\|_0$ , this leads to the same combinatorial problem

47

due to continuous optimization of  $x$  with a penalty term of the form  $f(V(x))$ . To address this, we can use the Lovász extension  $\check{f}(x)$  on a vector  $x$ . This function is convex but it is not a norm, but if we consider the construct defined as  $\|x\|_f = \check{f}(|x|)$ , it can be shown that this satisfies all the properties of a norm for all non-trivial submodular functions [PG98; Bac+13] (i.e., those normalized submodular functions for which  $f(v) > 0$  for all  $v$ ). In fact, the group lasso mentioned above is a special case for a particularly simple feature-based submodular function (a sum of min-truncated cardinality functions). But in principle, the same submodular design strategies mentioned in Section 6.11.6 can be used to produce a submodular function to instantiate an appropriate convex structured norm for a given machine learning problem.

### 6.11.12 Conclusions

We have only barely touched the surface of submodularity and how it applies to and can benefit machine learning. For more details, see [Bil22] and the many references contained therein. Considering once again the innocuous looking submodular inequality, then very much like the definition of convexity, we observe something that belies much of its complexity while opening the gates to wide and worthwhile avenues for machine learning exploration.

## 6.12 Derivative free optimization

**Derivative free optimization** or **DFO** refers to a class of techniques for optimizing functions without using derivatives. This is useful for blackbox function optimization as well as discrete optimization. If the function is expensive to evaluate, we can use Bayesian optimization (Section 6.9). If the function is cheap to evaluate, we can use **stochastic local search** methods or **evolutionary search** methods. To save space, we discuss these in the supplementary material.

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47



PART II

## Inference



# 7

## Inference algorithms: an overview

### 7.1 Introduction

In the probabilistic approach to machine learning, all unknown quantities — be they predictions about the future, hidden states of a system, or parameters of a model — are treated as random variables, and endowed with probability distributions. The process of **inference** corresponds to computing the posterior distribution over these quantities, conditioning on whatever data is available.

Let  $\mathbf{h}$  represent the unknown variables, and  $\mathcal{D}$  represent the known variables. Given a likelihood  $p(\mathcal{D}|\mathbf{h})$  and a prior  $p(\mathbf{h})$ , we can compute the posterior  $p(\mathbf{h}|\mathcal{D})$  using Bayes' rule:

$$p(\mathbf{h}|\mathcal{D}) = \frac{p(\mathbf{h})p(\mathcal{D}|\mathbf{h})}{p(\mathcal{D})} \quad (7.1)$$

The main computational bottleneck is computing the normalization constant in the denominator, which requires solving the following high dimensional integral:

$$p(\mathcal{D}) = \int p(\mathcal{D}|\mathbf{h})p(\mathbf{h})d\mathbf{h} \quad (7.2)$$

This is needed to convert the unnormalized joint probability of some parameter value,  $p(\mathbf{h}, \mathcal{D})$ , to a normalized probability,  $p(\mathbf{h}|\mathcal{D})$ , which takes into account all the other plausible values that  $\mathbf{h}$  could have. Similarly, computing posterior marginals also requires computing integrals:

$$p(h_i|\mathcal{D}) = \int p(h_i, \mathbf{h}_{-i}|\mathcal{D})d\mathbf{h}_{-i} \quad (7.3)$$

Thus integration is at the heart of Bayesian inference, whereas differentiation is at the heart of optimization.

In this chapter, we give a high level summary of algorithmic techniques for computing (approximate) posteriors. We will give more details in the following chapters. Note that most of these methods are independent of the specific model. This allows problem solvers to focus on creating the best model possible for the task, and then relying on some inference engine to do the rest of the work — this latter process is sometimes called “**turning the Bayesian crank**”.

### 7.2 Common inference patterns

There are kinds of posterior we may want to compute, but we can identify 3 main patterns, as we discuss below. These give rise to different types of inference algorithm, as we will see in later chapters.

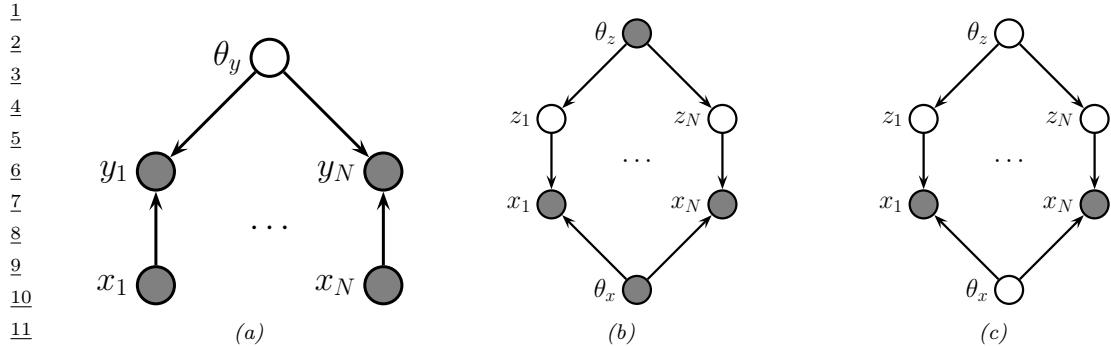


Figure 7.1: Graphical models with (a) Global hidden variables for representing the Bayesian discriminative model  $p(\mathbf{y}_{1:N}, \boldsymbol{\theta}_y | \mathbf{x}_{1:N}) = p(\boldsymbol{\theta}_y) \prod_{n=1}^N p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta}_y)$ ; (b) Local hidden variables for representing the generative model  $p(\mathbf{x}_{1:N}, \mathbf{z}_{1:N} | \boldsymbol{\theta}) = \prod_{n=1}^N p(\mathbf{z}_n | \boldsymbol{\theta}_z) p(\mathbf{x}_n | \mathbf{z}_n, \boldsymbol{\theta}_x)$ ; (c) Local and global hidden variables for representing the Bayesian generative model  $p(\mathbf{x}_{1:N}, \mathbf{z}_{1:N}, \boldsymbol{\theta}) = p(\boldsymbol{\theta}_z) p(\boldsymbol{\theta}_x) \prod_{n=1}^N p(\mathbf{z}_n | \boldsymbol{\theta}_z) p(\mathbf{x}_n | \mathbf{z}_n, \boldsymbol{\theta}_x)$ . Shaded nodes are assumed to be known (observed), unshaded nodes are hidden.

### 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  $\boldsymbol{\theta}$ , 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  $\mathbf{z}_{1:N}$ ; 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  $\mathbf{z}_n$  cannot be done exactly, we can use variational inference, a combination known as variational EM (Section 6.7.6.1).

Alternatively, we can use a minibatch approximation to the likelihood, marginalizing out  $\mathbf{z}_n$  for each example in the minibatch to get

$$\log p(\mathcal{D}_t | \boldsymbol{\theta}_t) = \sum_{n \in \mathcal{D}_t} \log \left[ \sum_{\mathbf{z}_n} p(\mathbf{x}_n, \mathbf{z}_n | \boldsymbol{\theta}_t) \right] \quad (7.6)$$

where  $\mathcal{D}_t$  is the minibatch at step  $t$ . If the marginalization cannot be done exactly, we can use variational inference, a combination known as stochastic variational inference or SVI (Section 10.3.2). We can also learn an inference network  $q_\phi(\mathbf{z} | \mathbf{x}; \boldsymbol{\theta})$  to perform the inference for us, rather than running an inference engine for each example  $n$  in each batch  $t$ ; the cost of learning  $\phi$  can be amortized across the batches. This is called amortized SVI, and is commonly used to train deep latent variable models such as VAEs (Section 22.2).

### 7.2.3 Global and local latents

The third pattern arises when we need to perform inference in models which have **local and global latent variables**. This is shown in Figure 7.1c, and corresponds to the following joint distribution:

$$p(\mathbf{x}_{1:N}, \mathbf{z}_{1:N}, \boldsymbol{\theta}) = p(\boldsymbol{\theta}_x)p(\boldsymbol{\theta}_z) \left[ \prod_{n=1}^N p(\mathbf{x}_n | \mathbf{z}_n, \boldsymbol{\theta}_x)p(\mathbf{z}_n | \boldsymbol{\theta}_z) \right] \quad (7.7)$$

This is essentially a Bayesian version of the latent variable model in Figure 7.1b, where now we model uncertainty in both the local variables  $\mathbf{z}_n$  and the shared global variables  $\boldsymbol{\theta}$ . This approach is less common in the ML community, since it is often assumed that the uncertainty in the parameters  $\boldsymbol{\theta}$  is negligible compared to the uncertainty in the local variables  $\mathbf{z}_n$ . The reason for this is that the parameters are “informed” by all  $N$  data cases, whereas each local latent  $\mathbf{z}_n$  is only informed by a single data point, namely  $\mathbf{x}_n$ . Nevertheless, there are advantages to being “fully Bayesian”, and modeling uncertainty in both local and global variables. We will see some examples of this later in the book.

## 7.3 Exact inference algorithms

In some cases, we can perform example posterior inference in a tractable manner. In particular, if the prior is **conjugate** to the likelihood, the posterior will be analytically tractable. In general, this will be the case when the prior and likelihood are from the same exponential family (Section 2.5). In particular, if the unknown variables are represented by  $\boldsymbol{\theta}$ , then we assume

$$p(\boldsymbol{\theta}) \propto \exp(\boldsymbol{\lambda}_0^\top \mathcal{T}(\boldsymbol{\theta})) \quad (7.8)$$

$$p(\mathbf{y}_i | \boldsymbol{\theta}) \propto \exp(\tilde{\boldsymbol{\lambda}}_i(\mathbf{y}_i)^\top \mathcal{T}(\boldsymbol{\theta})) \quad (7.9)$$

where  $\mathcal{T}(\boldsymbol{\theta})$  are the sufficient statistics, and  $\boldsymbol{\lambda}$  are the natural parameters. We can then compute the posterior by just adding the natural parameters:

$$p(\boldsymbol{\theta}|\mathbf{y}_{1:N}) = \exp(\boldsymbol{\lambda}_*^\top \mathcal{T}(\boldsymbol{\theta})) \quad (7.10)$$

$$\boldsymbol{\lambda}_* = \boldsymbol{\lambda}_0 + \sum_{n=1}^N \tilde{\boldsymbol{\lambda}}_n(\mathbf{y}_n) \quad (7.11)$$

See Section 3.2 for details.

Another setting where we can compute the posterior exactly arises when the  $D$  unknown variables are all discrete, each with  $K$  states; in this case, the integral for the normalizing constant becomes a sum with  $K^D$  terms. In many cases,  $K^D$  will be too large to be tractable. However, if the distribution satisfies certain conditional independence properties, as expressed by a probabilistic graphical model (PGM), then we can write the joint as a product of local terms (see Chapter 4 and Chapter 4). This lets us use dynamic programming to make the computation tractable. We discuss this idea for chain-structured PGMs in Chapter 8, and for general graphs in Chapter 9.

## 7.4 Approximate inference algorithms

For most probability models, we will not be able to compute marginals or posteriors exactly, so we must resort to using **approximate inference**. There are many different algorithms, which trade off speed, accuracy, simplicity, and generality. We briefly discuss some of these algorithms below. We give more detail in the following chapters.

### 7.4.1 MAP estimation

The simplest approximate inference method is to compute the MAP estimate

$$\hat{\boldsymbol{\theta}} = \operatorname{argmax} p(\boldsymbol{\theta}|\mathcal{D}) = \operatorname{argmax} \log p(\boldsymbol{\theta}) + \log p(\mathcal{D}|\boldsymbol{\theta}) \quad (7.12)$$

and then to assume that the posterior puts 100% of its probability on this single value:

$$p(\boldsymbol{\theta}|\mathcal{D}) \approx \delta(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) \quad (7.13)$$

The advantage of this approach is that we can compute the MAP estimate using a variety of optimization algorithms, which we discuss in Chapter 6. The disadvantages of the MAP approximation are discussed in Section 3.1.5.

### 7.4.2 Grid approximation

If we want to capture uncertainty, we need to allow for the fact that  $\boldsymbol{\theta}$  may have a range of possible values, each with non-zero probability. The simplest way to capture this property is to partition the space of possible values into a finite set of regions, call them  $\mathbf{r}_1, \dots, \mathbf{r}_K$ , each representing a region of parameter space of volume  $\Delta$  centered on  $\boldsymbol{\theta}_k$ . This is called a **grid approximation**. The probability of being in each region is given by  $p(\boldsymbol{\theta} \in \mathbf{r}_k | \mathcal{D}) \approx p_k \Delta$ , where

$$p_k = \frac{\tilde{p}_k}{\sum_{k'=1}^K \tilde{p}_{k'}} \quad (7.14)$$

$$\tilde{p}_k = p(\mathcal{D}|\boldsymbol{\theta}_k)p(\boldsymbol{\theta}_k) \quad (7.15)$$

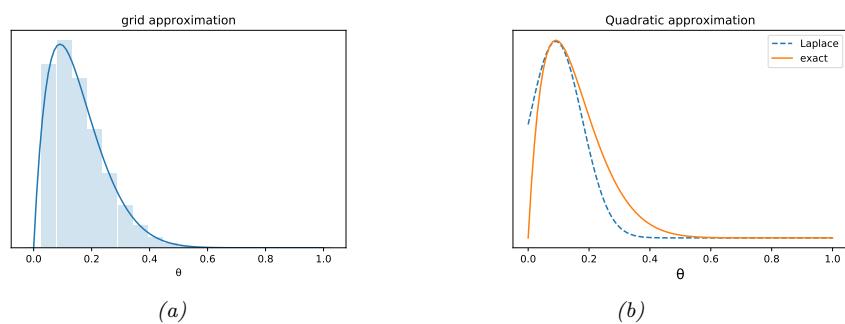


Figure 7.2: Approximating the posterior of a beta-Bernoulli model. (a) Grid approximation using 20 grid points. (b) Laplace approximation. Generated by `beta_binom_approx_post_pymc.ipynb`.

As  $K$  increases, we decrease the size of each grid cell. Thus the denominator is just a simple numerical approximation of the integral

$$p(\mathcal{D}) = \int p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta} \approx \sum_{k=1}^K \Delta \tilde{p}_k \quad (7.16)$$

As a simple example, we will use the problem of approximating the posterior of a beta-Bernoulli model. Specifically, the goal is to approximate

$$p(\theta|\mathcal{D}) \propto \left[ \prod_{n=1}^N \text{Ber}(y_n|\theta) \right] \text{Beta}(1, 1) \quad (7.17)$$

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:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{1}{Z} e^{-\mathcal{E}(\boldsymbol{\theta})} \quad (7.18)$$

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

$$\mathcal{E}(\boldsymbol{\theta}) \approx \mathcal{E}(\hat{\boldsymbol{\theta}}) + (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T \mathbf{g} + \frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T \mathbf{H}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) \quad (7.19)$$

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

$$\hat{p}(\boldsymbol{\theta}, \mathcal{D}) = e^{-\mathcal{E}(\hat{\boldsymbol{\theta}})} \exp \left[ -\frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T \mathbf{H}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) \right] \quad (7.20)$$

$$\hat{p}(\boldsymbol{\theta} | \mathcal{D}) = \frac{1}{Z} \hat{p}(\boldsymbol{\theta}, \mathcal{D}) = \mathcal{N}(\boldsymbol{\theta} | \hat{\boldsymbol{\theta}}, \mathbf{H}^{-1}) \quad (7.21)$$

$$Z = e^{-\mathcal{E}(\hat{\boldsymbol{\theta}})} (2\pi)^{D/2} |\mathbf{H}|^{-\frac{1}{2}} \quad (7.22)$$

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,  $\boldsymbol{\theta} \in \mathbb{R}^D$ . 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:

$$q^* = \underset{q \in \mathcal{Q}}{\operatorname{argmin}} D(q, p) \quad (7.23)$$

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  $\psi$ .

It is common to use the KL divergence (Section 5.1) as the discrepancy measure, which is given by

$$D(q, p) = D_{\text{KL}}(q(\boldsymbol{\theta} | \psi) \| p(\boldsymbol{\theta} | \mathcal{D})) = \int q(\boldsymbol{\theta} | \psi) \log \frac{q(\boldsymbol{\theta} | \psi)}{p(\boldsymbol{\theta} | \mathcal{D})} d\boldsymbol{\theta} \quad (7.24)$$

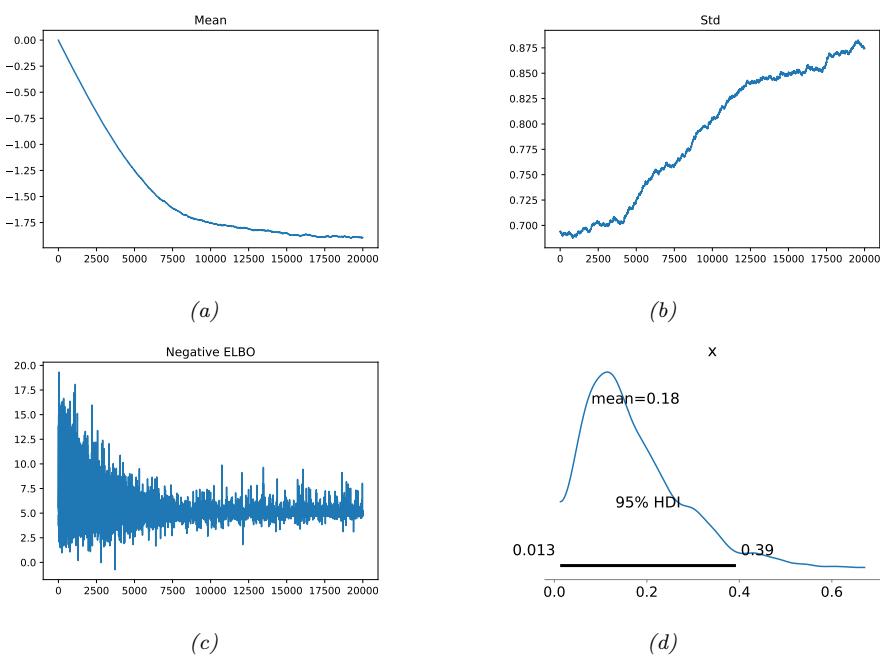


Figure 7.3: Approximating the posterior of a beta-Bernoulli model using Gaussian approximation to the logits,  $q(\alpha) = \mathcal{N}(\mu, \sigma)$ , where  $\alpha = \text{logit}(\theta)$ . (a) Estimate of variational mean over time. (b) Estimate of variational standard deviation over time. (c) ELBO 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_pymc.ipynb`.

where  $p(\boldsymbol{\theta}|\mathcal{D}) = p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})/p(\mathcal{D})$ . The inference problem then reduces to the following optimization problem:

$$\psi^* = \underset{\psi}{\operatorname{argmin}} D_{\text{KL}}(q(\boldsymbol{\theta}|\psi) \| p(\boldsymbol{\theta}|\mathcal{D})) \quad (7.25)$$

$$= \underset{\psi}{\operatorname{argmin}} \mathbb{E}_{q(\boldsymbol{\theta}|\psi)} \left[ \log q(\boldsymbol{\theta}|\psi) - \log \left( \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})} \right) \right] \quad (7.26)$$

$$= \underset{\psi}{\operatorname{argmin}} \mathbb{E}_{q(\boldsymbol{\theta}|\psi)} \underbrace{[-\log p(\mathcal{D}|\boldsymbol{\theta}) - \log p(\boldsymbol{\theta}) + \log q(\boldsymbol{\theta}|\psi)]}_{-\mathcal{L}(\psi)} + \log p(\mathcal{D}) \quad (7.27)$$

Note that  $\log p(\mathcal{D})$  is independent of  $\psi$ , so we can ignore it when fitting the approximate posterior, and just focus on maximizing the term

$$\mathcal{L}(\psi) \triangleq \mathbb{E}_{q(\boldsymbol{\theta}|\psi)} [\log p(\mathcal{D}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) - \log q(\boldsymbol{\theta}|\psi)] \quad (7.28)$$

Since we have  $D_{\text{KL}}(q||p) \geq 0$ , we have  $\mathcal{L}(\psi) \leq \log p(\mathcal{D})$ . The quantity  $\log p(\mathcal{D})$ , which is the log marginal likelihood, is also called the **evidence**. Hence  $\mathcal{L}(\psi)$  is known as the **evidence lower bound**.

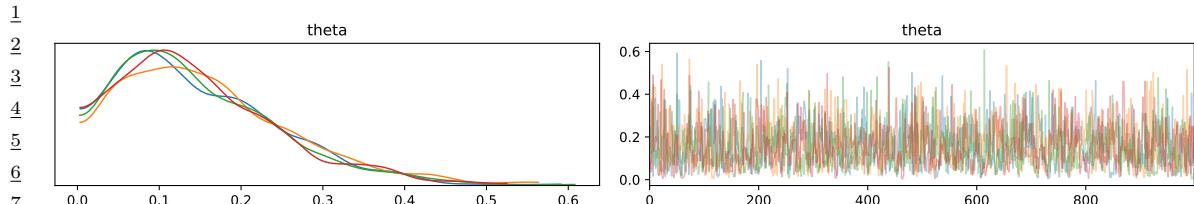


Figure 7.4: Approximating the posterior of a beta-Bernoulli model using MCMC. (a) Kernel density estimate derived from samples from 4 independent chains. (b) Trace plot of the chains as they generate posterior samples. Generated by `beta_binom_approx_post_pymc.ipynb`.

**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,  $q(\boldsymbol{\theta}|\psi) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ . 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  $q(\theta|\psi) = \text{Beta}(\theta|a, b)$ , where  $\psi = (a, b)$ . However choosing a suitable form of variational distribution requires some level of expertise. To create a more easily applicable, or “turn-key”, method, that works on a wide range of models, we can use a method called **automatic differentiation variational inference** or **ADVI** [Kuc+16]. This uses the change of variables method to convert the parameters to an unconstrained form, and then computes a Gaussian variational approximation. The method also uses automatic differentiation to derive the Jacobian term needed to compute the density of the transformed variables. See Section 10.3.5 for details.

We apply ADVI to our 1d beta-Bernoulli model in Figure 7.3. Let  $\alpha = \text{logit}(\theta)$ . We will use the approximation  $p(\alpha|\mathcal{D}) \approx q(\alpha|\psi) = \mathcal{N}(\alpha|\mu, \sigma)$ , where  $\psi = (\mu, \sigma)$ . We optimize the ELBO using SGD (this is known as **stochastic variational inference**). In panels a-b, we plot  $\mu$  and  $\sigma$  as a function of the number of steps of optimization. We see that  $\mu$  has converged, but  $\sigma$  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 that the method has (approximately) converged. Finally, in panel d, we draw samples from  $q(\alpha)$ , convert to  $\theta = \sigma(\alpha)$ , and then plot the KDE approximation to  $p(\theta|\mathcal{D})$ . We see that this is a fairly good approximation to the true beta posterior shown in Figure 7.2.

37

### 38 7.4.5 Markov Chain Monte Carlo (MCMC) 39

40 Although VI is fast, it can give a biased approximation to the posterior, since it is restricted to a  
41 specific function form  $q \in \mathcal{Q}$ . A more flexible approach is to use a non-parametric approximation in  
42 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**.  
43 The key issue is how to create the posterior samples  $\boldsymbol{\theta}^s \sim p(\boldsymbol{\theta}|\mathcal{D})$  efficiently, without having to  
44 evaluate the normalization constant  $p(\mathcal{D}) = \int p(\boldsymbol{\theta}, \mathcal{D}) d\boldsymbol{\theta}$ .

45 For low dimensional problems, we can use methods such as **importance sampling**, which we  
46 discuss in Section 11.5. However, for high dimensional problems, it is more common to use **Markov**  
47

**chain Monte Carlo** or **MCMC**. We give the details in Chapter 12, but give a brief introduction here.

The most common kind of MCMC is known as the **Metropolis Hastings algorithm**. The basic idea behind MH is as follows: we start at a random point in parameter space, and then perform a random walk, by sampling new states (parameters) from a **proposal distribution**  $q(\boldsymbol{\theta}'|\boldsymbol{\theta})$ . If  $q$  is chosen carefully, the resulting Markov chain distribution will satisfy the property that the fraction of time we visit each point in space is proportional to the posterior probability. The key point is that to decide whether to move to a newly proposed point  $\boldsymbol{\theta}'$  or to stay in the current point  $\boldsymbol{\theta}$ , we only need to evaluate the unnormalized density ratio

$$\frac{p(\boldsymbol{\theta}|\mathcal{D})}{p(\boldsymbol{\theta}'|\mathcal{D})} = \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})/p(\mathcal{D})}{p(\mathcal{D}|\boldsymbol{\theta}')p(\boldsymbol{\theta}')/p(\mathcal{D})} = \frac{p(\mathcal{D}, \boldsymbol{\theta})}{p(\mathcal{D}, \boldsymbol{\theta}')} \quad (7.29)$$

This avoids the need to compute the normalization constant  $p(\mathcal{D})$ . (In practice we usually work with log probabilities, instead of joint probabilities, to avoid numerical issues.)

We see that the input to the algorithm is just a function that computes the log joint density,  $\log p(\boldsymbol{\theta}, \mathcal{D})$ , as well as a proposal distribution  $q(\boldsymbol{\theta}'|\boldsymbol{\theta})$  for deciding which states to visit next. It is 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 **random walk Metropolis** algorithm. However, this can be very inefficient, since it is blindly walking through the space, in the hopes of finding higher probability regions.

In models that have conditional independence structure, it is often easy to compute the **full 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 like a stochastic analog of coordinate ascent, and is called **Gibbs sampling** (see Section 12.3 for details).

For models where all unknown variables are continuous, we can often compute the gradient of the log joint,  $\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}, \mathcal{D})$ . We can use this gradient information to guide the proposals into regions of space with higher probability. This approach is called **Hamiltonian Monte Carlo** or **HMC**, and is one of the most widely used MCMC algorithms due to its speed. For details, see Section 12.5.

We apply HMC to our beta-Bernoulli model in Figure 7.4. (We use a logit transformation for the parameter.) In panel b, we show samples generated by the algorithm from 4 parallel Markov chains. We see that they oscillate around the true posterior, as desired. In panel a, we compute a kernel density estimate from the posterior samples from each chain; we see that the result is a good approximation to the true posterior in Figure 7.2.

#### 7.4.6 Sequential Monte Carlo

MCMC is like a stochastic local search algorithm, in that it makes moves through the state space of the posterior distribution, comparing the current value to proposed neighboring values. An alternative approach is to use perform inference using a sequence of different distributions, from simpler to more complex, with the final distribution being equal to the target posterior. This is called **sequential Monte Carlo** or **SMC**. This approach, which is more similar to tree search than local search, has various advantages over MCMC, which we discuss in Chapter 13.

A common application of SMC is to **sequential Bayesian inference**, in which we recursively compute (i.e., in an online fashion) the posterior  $p(\boldsymbol{\theta}_t|\mathcal{D}_{1:t})$ , where  $\mathcal{D}_{1:t} = \{(\mathbf{x}_n, y_n) : n = 1 : t\}$  is all the data we have seen so far. This sequence of distributions converges to the full batch posterior  $p(\boldsymbol{\theta}|\mathcal{D})$  once all the data has been seen. However, the approach can also be used when the data is

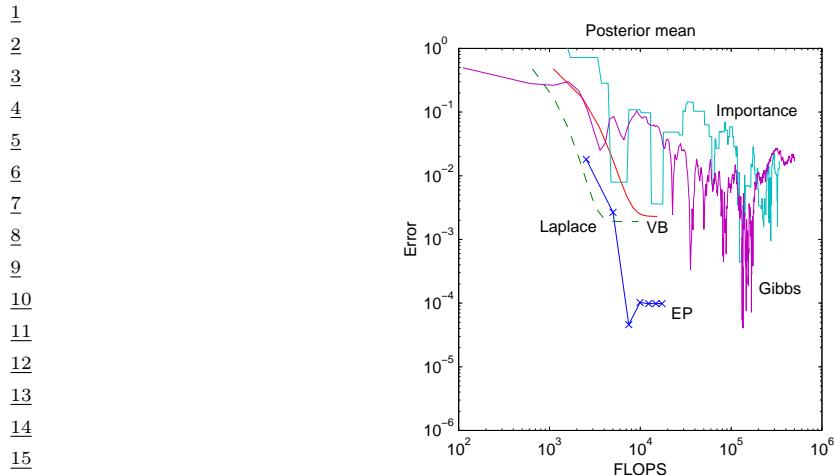


Figure 7.5: Accuracy vs compute time for different inference methods. Code source: <https://github.com/tminka/ep-clutter-example>. From Figure 1 of [Min01b]. Used with kind permission of Tom Minka.

arriving in a continual, unending stream, as in state-space models (see Chapter 31). The application of SMC to such dynamical models is known as **particle filtering**. See Section 13.2 for details.

## 7.5 Evaluating approximate inference algorithms

There are many different inference algorithms each of which make different tradeoffs between speed, accuracy, generality, simplicity, etc. This makes it hard to compare them on an equal footing. However, a common approach is to evaluate the accuracy of the approximation as a function of compute time.

We give an example of this in Figure 7.5, where we plot performance for several different inference algorithms applied to the following simple 1d model:

$$p(z) = \mathcal{N}(z|0, 100) \quad (7.30)$$

$$p(x_i|z) = 0.5\mathcal{N}(x_i|z, 1) + 0.5\mathcal{N}(x_i|0, 10) \quad (7.31)$$

That is, each measurement  $x_i$  is either a noisy copy of the hidden quantity of interest,  $z$ , or is an outlier coming from a uniform background noise model, approximated by a Gaussian with a large variance,  $\mathcal{N}(0, 10)$ . The goal is to compute  $p(z|\mathbf{x}_{1:N})$ . (Tom Minka (who created this example) calls this the **clutter problem**.)

Since this is a 1d problem, we can compute the exact answer using numerical integration. In Figure 7.5, we plot the accuracy of the posterior mean estimate using various approximate inference methods, namely: the Laplace approximation (Section 7.4.3), variational Bayes (Section 10.2.3), expectation propagation (Section 10.7), importance sampling (Section 11.5), and Gibbs sampling (Section 12.3). We see that the error smoothly decreases for the 3 deterministic methods (Laplace, BP and EP). For the 2 stochastic methods (IS and Gibbs), the error decreases on average, but the performance is quite noisy.

47

In principle, the Monte Carlo methods will converge to a zero overall error, since they are unbiased estimators, but it is clear that their variance is quite high. The deterministic methods, by comparison, converge to a finite error, so they are biased, but their variance is much lower. We can trade off bias and variance by creating hybrid algorithms, that combine different techniques. We will see examples of this in later chapters.

When we cannot compute the true target posterior distribution to compare to, evaluation is harder, but there are some approaches than can be used in certain cases. For some methods for assessing variational inference, see e.g., [Yao+18b; Hug+20]. For some methods for assessing Monte Carlo methods, see [CGR06; CTM17; GAR16]. For assessing posterior predictive distributions, see Section 14.2.3.

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



# 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.<sup>1</sup>

### 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](https://en.wikipedia.org/wiki/Kalman_filter)) to use  $\mathbf{x}_t$  to represent the hidden states, and  $\mathbf{y}_t$  to represent the observations. However, this convention is inconsistent with the rest of this book (and the rest of the ML literature), where  $\mathbf{x}$  is used to represent observed features and  $\mathbf{z}$  to represent hidden states. As a compromise, we use  $\mathbf{z}$  to represent hidden states,  $\mathbf{y}$  to represent observations, and  $\mathbf{u}$  for optional inputs, representing control signals or other covariates that we condition on.

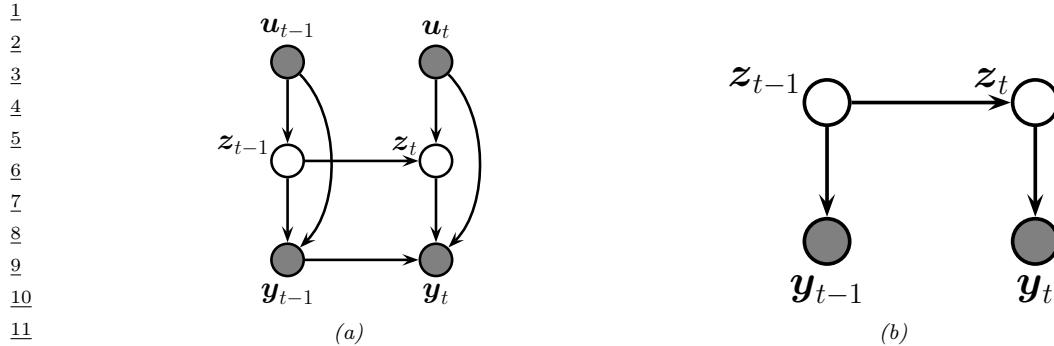


Figure 8.1: State-space model represented as a graphical model. (a) Generic form, with inputs  $\mathbf{u}_t$ , hidden state  $\mathbf{z}_t$ , and observations  $\mathbf{y}_t$ . We assume the observation likelihood is first-order auto-regressive. (b) Simplified form, with no inputs, and Markovian observations.

hidden state:

$$p(\mathbf{y}_{1:T}|\mathbf{z}_{1:T}) = \prod_{t=1}^T p(\mathbf{y}_t|\mathbf{z}_t) \quad (8.3)$$

However, often the hidden state is not sufficient to explain all the observations. We can generalize this by letting the current observation also depend on past observations, as in an auto-regressive model:

$$p(\mathbf{y}_{1:T}|\mathbf{z}_{1:T}) = \prod_{t=1}^T p(\mathbf{y}_t|\mathbf{z}_t, \mathbf{y}_{1:t-1}) \quad (8.4)$$

We can also let the model also depend on exogeneous inputs or covariates that are assumed to be known a priori (i.e., they are not explained or generated by the model); we denote these by  $\mathbf{u}_t$ . This gives rise to the following conditional joint distribution:

$$p(\mathbf{z}_{1:T}, \mathbf{y}_{1:T}|\mathbf{u}_{1:T}) = \prod_{t=1}^T p(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t)p(\mathbf{y}_t|\mathbf{z}_t, \mathbf{y}_{1:t-1}, \mathbf{u}_t) \quad (8.5)$$

See Figure 8.1(a) for an illustration of this model, assuming first-order dependencies between the observations.

To simplify the notation, we will usually ignore the inputs  $\mathbf{u}_t$ , and assume Markovian (non-autoregressive) likelihoods, giving rise to the simplified form in Figure 8.1(b). This corresponds to the model

$$p(\mathbf{z}_{1:T}, \mathbf{y}_{1:T}) = \prod_{t=1}^T p(\mathbf{z}_t|\mathbf{z}_{t-1})p(\mathbf{y}_t|\mathbf{z}_t, \boldsymbol{\theta}) \quad (8.6)$$

This is called a **state space model** or **SSM**. We generally assume  $\mathbf{z}_t \in \mathbb{R}^n$ . However, in the special case that the hidden state is discrete, so  $z_t \in \{1, \dots, K\}$ , the model is called a **hidden Markov model**. We give examples of these models below, and discuss them in more detail in Chapter 30 and Chapter 31.

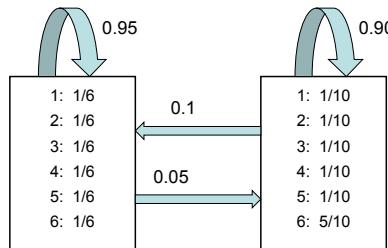


Figure 8.2: An HMM for the occasionally dishonest casino. The blue arrows represent the state transition diagram **A**. The list of numbers in each rectangle are the parameters of the observation matrix **B**. Adapted from [Dur+98, p54].

### 8.1.2 Example: casino HMM

In this section, we give a simple example of an HMM, which we will use to illustrate various algorithms. Suppose we are in a casino and observe a series of dice rolls,  $y_t \in \{1, 2, \dots, 6\}$ . Being a keen-eyed statistician, we notice that the distribution of values is not what we expect from a fair die: it seems that there are occasional “streaks”, in which 6s seem to show up more often than other values. We would like to **segment** the time series into regimes corresponding to the use of a fair die ( $z = 1$ ) and a loaded die ( $z = 2$ ). We have no labeled data (since the casino does not want to admit they are cheating), but we can still hope to infer  $z_{1:T}$  from  $y_{1:T}$  by creating a model and using posterior inference, as we show below. (This example is from [Dur+98], who call this setup the **occasionally dishonest casino**.)

Let us model this with an HMM. (In this example, there are no inputs  $u_t$ .) Let  $A_{jk} = p(z_t = k | z_{t-1} = j)$  be the state transition matrix, and  $B_{kl} = p(y_t = l | z_t = k)$  be the observation matrix corresponding to a categorical distribution over values of the dice face. Most of the time the casino uses a fair dice,  $z = 1$ , but occasionally it switches to a loaded dice,  $z = 2$ , for a short period. If  $z = 1$  the observation distribution is a uniform categorical distribution over the symbols  $\{1, \dots, 6\}$ . If  $z = 2$ , the observation distribution is skewed towards face 6. That is,

$$p(y_t | z_t = 1) = \text{Cat}(y_t | [1/6, \dots, 1/6]) \quad (8.7)$$

$$p(y_t | z_t = 2) = \text{Cat}(y_t | [1/10, 1/10, 1/10, 1/10, 1/10, 5/10]) \quad (8.8)$$

See Figure 8.2 for an illustration of the state transition diagram **A** and the emission distributions **B**. If we sample from this model, we may generate data such as the following:

$y: 664153216162115234653214356634261655234232315142464156663246$   
 $z: 2222222222222111112222222222211111111111122222222$

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, the next step is perform posterior inference. This can mean computing the marginal distribution over the current hidden state given the data up until now,

$\underline{1}$   $p(\mathbf{z}_t | \mathbf{y}_{1:t})$ ; or the distribution over the current hidden state given all the data, or  $p(\mathbf{z}_t | \mathbf{y}_{1:T})$ ; or the  
 $\underline{2}$  most probable sequence given all the data,  $\text{argmax}_{\mathbf{z}_{1:T}} p(\mathbf{z}_{1:T} | \mathbf{y}_{1:T})$ ; or a sample from the posterior  
 $\underline{3}$  over latent sequences,  $\mathbf{z}_{1:T} \sim p(\mathbf{z}_{1:T} | \mathbf{y}_{1:T})$ ; and so on. We discuss these inference tasks in more detail  
 $\underline{4}$  in Section 8.1.4.  
 $\underline{5}$

$\underline{6}$

### $\underline{7}$ 8.1.3 Example: linear-Gaussian SSM for tracking in 2d

$\underline{8}$  In this section, we give an example of a commonly used kind of SSM known as a **linear-Gaussian**  
 $\underline{9}$  **state space model**, also called **linear dynamical system**. These models are described in detail  
 $\underline{10}$  in Section 31.2, but basically they correspond to the following generative model:  
 $\underline{11}$

$$\underline{12} \quad p(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{u}_t) = \mathcal{N}(\mathbf{z}_t | \mathbf{A}_t \mathbf{z}_{t-1} + \mathbf{B}_t \mathbf{u}_t, \mathbf{Q}_t) \quad (8.9)$$

$$\underline{13} \quad p(\mathbf{y}_t | \mathbf{z}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{y}_t | \mathbf{C}_t \mathbf{z}_t + \mathbf{D}_t \mathbf{u}_t, \mathbf{R}_t) \quad (8.10)$$

$\underline{14}$  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  
 $\underline{15}$  model is stationary. See Figure 8.1 for the PGM-D.<sup>2</sup>

$\underline{16}$  A common application of LG-SSMs is for **tracking** objects, such as airplanes or animals, from  
 $\underline{17}$  noisy measurements, such as radar or cameras. We discuss a specific 2d example in Section 31.2.2.  
 $\underline{18}$  Here we give a brief summary of this model, so we can use this as a running example. The hidden  
 $\underline{19}$  state  $\mathbf{z}_t$  encodes the location,  $(x, y)$ , and the velocity,  $(\dot{x}, \dot{y})$ , of the moving object. The observation  
 $\underline{20}$   $\mathbf{y}_t$  is a noisy version of the location. (The velocity is not observed but can be inferred from the  
 $\underline{21}$  change in location.) We assume that we obtain measurements with a sampling frequency of  $\Delta$ . The  
 $\underline{22}$  new location is the old location plus  $\Delta$  times the velocity, plus noise added to all terms:  
 $\underline{23}$

$$\underline{24} \quad \mathbf{z}_t = \underbrace{\begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{A}} \mathbf{z}_{t-1} + \mathbf{q}_t \quad (8.11)$$

$\underline{25}$  where  $\mathbf{q}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_t)$ . The observation extracts the location and adds noise: In matrix notation, the  
 $\underline{26}$  observation model becomes

$$\underline{27} \quad \mathbf{y}_t = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}}_{\mathbf{C}} \mathbf{z}_t + \mathbf{r}_t \quad (8.12)$$

$\underline{28}$  where  $\mathbf{r}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_t)$ .

$\underline{29}$  Our goal is to use this model to estimate the unknown location (and velocity) of the object given  
 $\underline{30}$  the noisy observations. We discuss this in more detail in Section 8.1.4.

$\underline{31}$

### $\underline{32}$ 8.1.4 Inferential goals

$\underline{33}$  When faced with an SSM, there are various forms of inference we may be interested in performing. We  
 $\underline{34}$  give a visual summary in Figure 8.3, and give more details below. (See also [Sar13] for detailed coverage  
 $\underline{35}$  of inference in SSMs, as well as <https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python>,  
 $\underline{36}$  which has lots of excellent tutorial material.)

$\underline{37}$

$\underline{38}$  2. Some authors write  $\mathbf{F}_t$  instead of  $\mathbf{A}_t$  and  $\mathbf{H}_t$  instead of  $\mathbf{C}_t$ .

$\underline{39}$

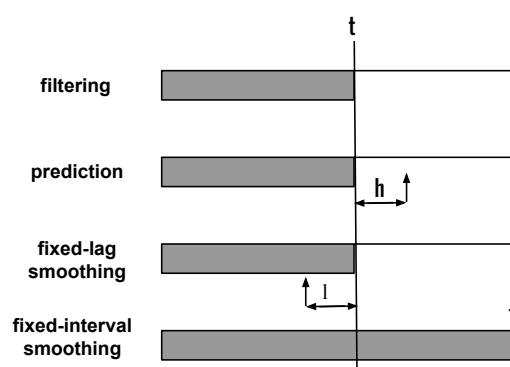


Figure 8.3: The main kinds of inference for state-space models. The shaded region is the interval for which we have data. The arrow represents the time step at which we want to perform inference.  $t$  is the current time,  $T$  is the sequence length,  $\ell$  is the lag and  $h$  is the prediction horizon. See text for details.

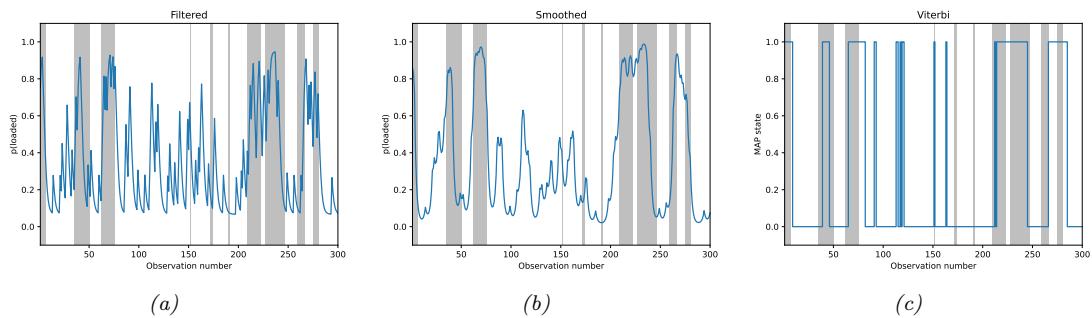


Figure 8.4: Inference in the dishonest casino. Vertical gray bars denote times when the hidden state corresponded to the loaded die. Blue lines represent the posterior probability of being in that state given different subsets of observed data. If we recover the true state exactly, the blue curve will transition at the same time as the gray bars. (a) Filtered estimate. (b) Smoothed estimates. (c) MAP trajectory. Generated by [hmm\\_casino.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

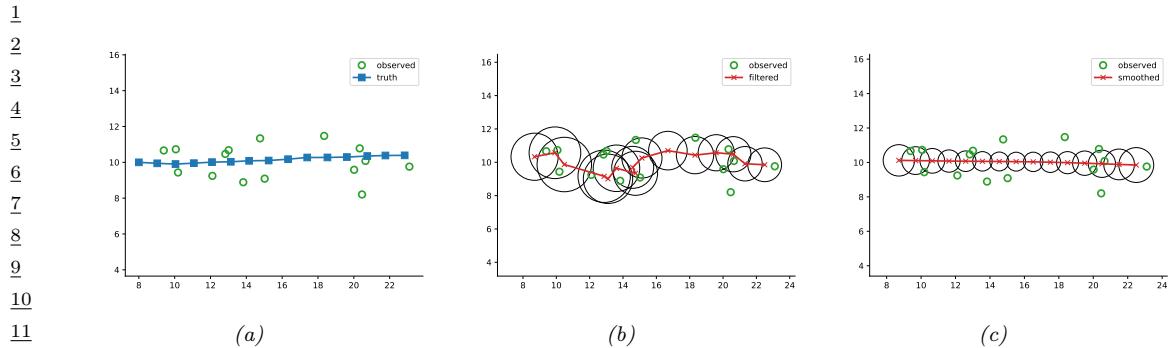


Figure 8.5: Illustration of Kalman filtering and smoothing for a linear dynamical system. (a) Observations (green circles) are generated by an object moving to the right (true location denoted by blue squares). (b) Results of online Kalman filtering. Red cross is the posterior mean, circles are 95% confidence 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.py`.

also has many false alarms. We will resolve this issue below.

Figure 8.5(b) illustrates filtering for the linear-Gaussian SSM, applied to the noisy tracking data in Figure 8.5(a) (shown by the green dots). The filtered estimates are computed using the Kalman filter algorithm described in Section 8.4.1. The red line shows the posterior mean estimate of the location, and the black circles show the posterior covariance. We see that the estimated trajectory is less noisy than the raw data, since it incorporates prior knowledge about the dynamics.

#### 8.1.4.2 Smoothing

If we are in the offline setting, we can compute the posterior over hidden states conditional on *all* the data,  $p(\mathbf{z}_t | \mathbf{y}_{1:T})$ . This is known as **(fixed interval) smoothing**. Figure 8.4(b) illustrates smoothing for the casino HMM, implemented using the forwards-backwards algorithm described in Section 8.3.3. We see that the resulting estimate is much smoother than the filtered (online) estimate.

Figure 8.5(c) illustrates smoothing for the LG-SSM, implemented using the Kalman smoothing algorithm described in Section 8.4.4. We see that the resulting estimate is smoother, and that the posterior uncertainty is reduced (as visualized by the smaller confidence ellipses).

To understand this behavior intuitively, consider a detective trying to figure out who committed a crime. As he moves through the crime scene, his uncertainty is high until he finds the key clue; then he has an “aha” moment, his uncertainty is reduced, and all the previously confusing observations are, in **hindsight**, easy to explain. Thus we see that, given all the data (including finding the clue), it is much easier to infer the state of the world.

The disadvantage of the above smoothing method is that we have to wait until all the data has been observed before we start inference. **Fixed lag smoothing** is a useful compromise between online and offline estimation; it involves computing  $p(\mathbf{z}_{t-\ell} | \mathbf{y}_{1:t})$ , where  $\ell > 0$  is called the lag. This gives better performance than filtering, but incurs a slight delay. By changing the size of the lag, we can trade off accuracy vs delay.

47

1

### 8.1.4.3 MAP state estimation

2 The **MAP estimate** is the most probable sequence of states, i.e.,

3

$$\underline{z}_{1:T}^* = \underset{\underline{z}_{1:T}}{\operatorname{argmax}} p(\underline{z}_{1:T} | \mathbf{y}_{1:T}) \quad (8.13)$$

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  $p(\underline{z}_{t+h} | \mathbf{y}_{1:t})$ , 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,  $p(\underline{z}_t | \mathbf{y}_{1:t})$ ,  
17 and passing it through the transition model  $h$  times:

18

$$p(\underline{z}_{t+1} | \mathbf{y}_{1:t}) = \int p(\underline{z}_{t+1} | \underline{z}_t) p(\underline{z}_t | \mathbf{y}_{1:t}) d\underline{z}_t \quad (8.14)$$

19

$$p(\underline{z}_{t+2} | \mathbf{y}_{1:t}) = \int p(\underline{z}_{t+2} | \underline{z}_{t+1}) p(\underline{z}_{t+1} | \mathbf{y}_{1:t}) d\underline{z}_{t+1} \quad (8.15)$$

20      :

21

$$p(\underline{z}_{t+h} | \mathbf{y}_{1:t}) = \int p(\underline{z}_{t+h} | \underline{z}_{t+h-1}) p(\underline{z}_{t+h-1} | \mathbf{y}_{1:t}) d\underline{z}_{t+h-1} \quad (8.16)$$

22 The quantity  $p(\underline{z}_{t+h} | \mathbf{y}_{1:t})$  is a prediction about future hidden states. We can convert this into a  
23 prediction about future observations using

24

$$p(\mathbf{y}_{t+h} | \mathbf{y}_{1:t}) = \int p(\mathbf{y}_{t+h} | \underline{z}_{t+h}) p(\underline{z}_{t+h} | \mathbf{y}_{1:t}) d\underline{z}_{t+h} \quad (8.17)$$

25 This is the  $h$ -step-ahead predictive distribution for observations, and can be used for time-series  
26 forecasting (see Section 19.3).

27

## 8.2 Bayesian filtering and smoothing

28 In this section, we derive the optimal form for the filtered and smoothed posteriors for the simplified  
29 SSM in Equation (8.6). (The equations can easily be extended to the more general model in  
30 Equation (8.5).) This will require integrating (or summing) over the latent states. In the following  
31 sections, we discuss how to compute these integrals in practice, depending on the form of the model,  
32 and any approximations we choose to make.

1 **8.2.1 The filtering equations**

3 The recursive **Bayes filter** starts with a prior distribution  $p(\mathbf{z}_0)$ , and then repeats the following two  
4 steps for each time step  $t$ .

6 **8.2.1.1 Prediction step**

8 The **prediction step** is just the **Chapman-Kolmogorov equation**:

$$\underline{10} \quad p(\mathbf{z}_t | \mathbf{y}_{1:t-1}) = \int p(\mathbf{z}_t | \mathbf{z}_{t-1}) p(\mathbf{z}_{t-1} | \mathbf{y}_{1:t-1}) d\mathbf{z}_{t-1} \quad (8.18)$$

12 The prediction step computes the one-step-ahead predictive distribution for the latent state, which  
13 updates the posterior from the previous time step into the prior for the current step.

15 **8.2.1.2 Update step**

16 The **update step**, is just Bayes rule:

$$\underline{18} \quad p(\mathbf{z}_t | \mathbf{y}_{1:t}) = \frac{1}{Z_t} p(\mathbf{y}_t | \mathbf{z}_t) p(\mathbf{z}_t | \mathbf{y}_{1:t-1}) \quad (8.19)$$

20 where the normalization constant is

$$\underline{22} \quad Z_t = \int p(\mathbf{y}_t | \mathbf{z}_t) p(\mathbf{z}_t | \mathbf{y}_{1:t-1}) d\mathbf{z}_t = p(\mathbf{y}_t | \mathbf{y}_{1:t-1}) \quad (8.20)$$

24 We will give concrete implementations of these equations in later sections.

26 **8.2.2 The smoothing equations**

27 In the offline setting, we want to compute  $p(\mathbf{z}_t | \mathbf{y}_{1:T})$ , as discussed in Section 8.1.4.2. We can do this  
28 recursively as follows:

$$\underline{30} \quad p(\mathbf{z}_t | \mathbf{y}_{1:T}) = p(\mathbf{z}_t | \mathbf{y}_{1:t}) \int \left[ \frac{p(\mathbf{z}_{t+1} | \mathbf{z}_t) p(\mathbf{z}_{t+1} | \mathbf{y}_{1:T})}{p(\mathbf{z}_{t+1} | \mathbf{y}_{1:t})} \right] d\mathbf{z}_{t+1} \quad (8.21)$$

32 This can be written in words as follows:

$$\underline{34} \quad \text{posterior}(\mathbf{z}_t) = \text{filtered}(\mathbf{z}_t) \int \left[ \frac{\text{dynamics}(\mathbf{z}_{t+1}) \times \text{smoothed}(\mathbf{z}_{t+1})}{\text{predictive}(\mathbf{z}_{t+1})} \right] d\mathbf{z}_{t+1} \quad (8.22)$$

36 To see why this equation is true, note that from the Markov properties of the model, and Bayes rule,  
37 we have

$$\underline{39} \quad p(\mathbf{z}_t | \mathbf{z}_{t+1}, \mathbf{y}_{1:T}) = p(\mathbf{z}_t | \mathbf{z}_{t+1}, \mathbf{y}_{1:t}) \quad (8.23)$$

$$\underline{40} \quad = \frac{p(\mathbf{z}_t, \mathbf{z}_{t+1} | \mathbf{y}_{1:t})}{p(\mathbf{z}_{t+1} | \mathbf{y}_{1:t})} \quad (8.24)$$

$$\underline{42} \quad = \frac{p(\mathbf{z}_{t+1} | \mathbf{z}_t, \mathbf{y}_{1:t}) p(\mathbf{z}_t | \mathbf{y}_{1:t})}{p(\mathbf{z}_{t+1} | \mathbf{y}_{1:t})} \quad (8.25)$$

$$\underline{45} \quad = \frac{p(\mathbf{z}_{t+1} | \mathbf{z}_t) p(\mathbf{z}_t | \mathbf{y}_{1:t})}{p(\mathbf{z}_{t+1} | \mathbf{y}_{1:t})} \quad (8.26)$$

Hence the joint distribution over two consecutive time steps is given by

$$p(\mathbf{z}_t, \mathbf{z}_{t+1} | \mathbf{y}_{1:T}) = p(\mathbf{z}_t | \mathbf{z}_{t+1}, \mathbf{y}_{1:T}) p(\mathbf{z}_{t+1} | \mathbf{y}_{1:T}) \quad (8.27)$$

$$= p(\mathbf{z}_t | \mathbf{z}_{t+1}, \mathbf{y}_{1:t}) p(\mathbf{z}_{t+1} | \mathbf{y}_{1:T}) \quad (8.28)$$

$$= \frac{p(\mathbf{z}_{t+1} | \mathbf{z}_t) p(\mathbf{z}_t | \mathbf{y}_{1:t}) p(\mathbf{z}_{t+1} | \mathbf{y}_{1:T})}{p(\mathbf{z}_{t+1} | \mathbf{y}_{1:t})} \quad (8.29)$$

Marginalizing out  $\mathbf{z}_{t+1}$  gives Equation (8.21).

## 8.3 Inference for discrete SSMs

In this section, we consider inference for chain-structured graphical models, where all the hidden variables are discrete. This includes HMMs (Section 30.1), linear-chain CRFs (Section 19.2), etc. We represent the hidden variables at time  $t$  by  $\mathbf{z}_t \in \{1, \dots, K\}$ , and the visible variables by  $\mathbf{y}_t \in \mathbb{R}^D$ .

As we discussed in Section 8.1.4, there are several kinds of inference we may be interested in when working with temporal or sequential models, namely filtering (i.e., computing  $p(\mathbf{z}_t | \mathbf{y}_{1:t})$ ), smoothing (i.e., computing  $p(\mathbf{z}_t | \mathbf{y}_{1:T})$ ), and MAP estimation (i.e., computing  $\text{argmax}_{\mathbf{z}_{1:T}} p(\mathbf{z}_{1:T} | \mathbf{y}_{1:T})$ ). In this section, we discuss how to solve all three of these problems in  $O(TK^2)$  time, where  $T$  is the length of the chain. The basic idea is to exploit the Markov structure of the model so that we can recursively solve smaller inference problems for the left and right half of the model, and then combine the solutions to these subproblems in an optimal way. We give the details below.

### 8.3.1 Forwards filtering

In this section, we discuss the **forwards algorithm** for HMMs, which is a way to recursively compute the **belief state**  $\alpha_t = p(\mathbf{z}_t | \mathbf{y}_{1:t})$ , where  $\alpha_t(j) = p(\mathbf{z}_t = j | \mathbf{y}_{1:t})$  is the probability the hidden state has value  $j$  given the evidence seen so far. We can do this using **sequential Bayesian updating** as follows:

$$p(\mathbf{z}_t = j | \mathbf{y}_{1:t}) = p(\mathbf{z}_t = j | \mathbf{y}_t, \mathbf{y}_{1:t-1}) \propto p(\mathbf{y}_t | \mathbf{z}_t = j, \mathbf{y}_{1:t-1}) p(\mathbf{z}_t = j | \mathbf{y}_{1:t-1}) \quad (8.30)$$

$$= p(\mathbf{y}_t | \mathbf{z}_t = j) \left[ \sum_i p(\mathbf{z}_t = j | \mathbf{z}_{t-1} = i) p(\mathbf{z}_{t-1} = i | \mathbf{y}_{1:t-1}) \right] \quad (8.31)$$

where we have crossed out  $\mathbf{y}_{1:t-1}$  since  $\mathbf{y}_t$  is conditionally independent of past observations given  $\mathbf{z}_t$ . (This assumption is not actually necessary for the algorithm to work: the only requirement is that the hidden states be first-order Markov.)

The term in the brackets is known as the **one-step ahead prediction distribution**, and is given by

$$\alpha_{t|t-1}(j) \triangleq p(\mathbf{z}_t = j | \mathbf{y}_{1:t-1}) = \sum_i \alpha_{t-1}(i) A(i, j) \quad (8.32)$$

where  $A(i, j) = p(\mathbf{z}_t = j | \mathbf{z}_{t-1} = i)$  is the state transition probability. We then multiply this by the **local evidence**

$$\lambda_t(j) \triangleq p(\mathbf{y}_t | \mathbf{z}_t = j) \quad (8.33)$$

1 which is the result of evaluating the observation model using  $\mathbf{y}_t$  for each possible state. Combining  
2 these gives  
3

$$\underline{4} \quad \alpha_t(j) = \frac{1}{Z_t} \lambda_t(j) \left[ \sum_i \alpha_{t-1}(i) A(i, j) \right] \quad (8.34)$$

5 where the normalization constant for each time step is given by  
6

$$\underline{7} \quad Z_t \triangleq p(\mathbf{y}_t | \mathbf{y}_{1:t-1}) = \sum_{j=1}^K p(\mathbf{y}_t | z_t = j) p(z_t = j | \mathbf{y}_{1:t-1}) = \sum_{j=1}^K \lambda_t(j) \left[ \sum_i \alpha_{t-1}(i) A(i, j) \right] \quad (8.35)$$

8 We can therefore view Equation (8.31) in terms of a **predict-update cycle**: in the first step, we  
9 predict the distribution over states given the past (i.e., we compute  $\alpha_{t|t-1}$ ), and then we update our  
10 predictions using the likelihood  $p(\mathbf{y}_t | z_t) = \lambda_t$  to compute the new posterior  $\alpha_t$ .  
11

12 Since all the quantities are finite length vectors and matrices, we can write the update equation in  
13 matrix-vector notation as follows:

$$\underline{14} \quad \alpha_t = \text{normalize} (\lambda_t \odot (\mathbf{A}^\top \alpha_{t-1})) \quad (8.36)$$

15 where  $\odot$  represents elementwise vector multiplication, and the normalize function just ensures its  
16 argument sums to one.

17 It is useful to keep track of the normalization constants, since they can be used to compute the log  
18 likelihood of the sequence as follows:

$$\underline{19} \quad \log p(\mathbf{y}_{1:T}) = \sum_{t=1}^T \log p(\mathbf{y}_t | \mathbf{y}_{1:t-1}) = \sum_{t=1}^T \log Z_t \quad (8.37)$$

### 20 8.3.1.1 An aside on normalization

21 In most publications on HMMs,  $\alpha_t(j)$  is defined as the unnormalized *joint probability*  $p(z_t = j, \mathbf{y}_{1:t})$ ,  
22 perhaps due to the influential tutorial [Rab89]. That is, the standard definition is to use

$$\underline{23} \quad \alpha'_t(j) = p(z_t = j, \mathbf{y}_{1:t}) = \lambda_t(j) \left[ \sum_i \alpha'_{t-1}(i) A(i, j) \right] \quad (8.38)$$

24 without the  $Z_t$  term. We instead define  $\alpha_t(j)$  as the normalized *conditional probability*  $p(z_t = j | \mathbf{y}_{1:t})$ .  
25

26 The unnormalized form has several problems. First, it rapidly suffers from numerical underflow,  
27 since the probability of the joint event that  $(z_t = k, \mathbf{y}_{1:t})$  is vanishingly small.<sup>3</sup> Second, it is less  
28 interpretable, since it is not a distribution over states. Third, it precludes the use of methods which  
29 are designed to approximate state distributions (we will see such methods later).

30 Of course, the two definitions only differ by a multiplicative constant, since  $p(z_t = j | \mathbf{y}_{1:t}) = p(z_t = j, \mathbf{y}_{1:t}) / p(\mathbf{y}_{1:t})$  [Dev85]. So the *algorithmic* difference is just one line of code (namely the presence  
31 or absence of a call to the `normalize` function). Nevertheless, we feel it is better to present the  
32 normalized version, since it will encourage readers to implement the method properly (normalizing  
33 after each step to avoid underflow), and to think about it in terms of posterior filtering.  
34

35 3. For example, if the observations are independent of the states, we have  $p(z_t = j, \mathbf{y}_{1:t}) = p(z_t = j) \prod_{i=1}^t p(\mathbf{y}_i)$ , which  
36 becomes exponentially small with  $t$ .

### 8.3.2 Backwards smoothing

In this section, we show how to apply the generic Bayesian smoothing algorithm from Section 8.2.2 to an HMM. We assume, by induction, that we have already computed the smoothed posterior marginal for  $t + 1$ :

$$\gamma_{t+1}(j) \triangleq p(z_{t+1} = j | \mathbf{y}_{1:T}) \quad (8.39)$$

Then Equation (8.21) becomes

$$p(z_t = i | \mathbf{y}_{1:T}) = p(z_t = i | \mathbf{y}_{1:t}) \sum_j \left[ \frac{p(z_{t+1} = j | z_t = i)p(z_{t+1} = j | \mathbf{y}_{1:T})}{p(z_{t+1} = j | \mathbf{y}_{1:t})} \right] \quad (8.40)$$

$$= \alpha_t(i) \sum_j \left[ \frac{A(i, j)\gamma_{t+1}(j)}{\sum_{i'} \alpha_t(i')A(i', j)} \right] \quad (8.41)$$

We initialize the recursion using  $\gamma_T(j) = \alpha_T(j) = p(z_t = j | \mathbf{y}_{1:T})$ . Since we first have to perform filtering, and then smoothing, this two-pass algorithm is sometimes called **forwards filtering, backwards smoothing**.

### 8.3.3 The forwards-backwards algorithm

In the “traditional” approach to inference in HMMs, known as the **forwards-backwards algorithm**, the backwards step has a different form. It leverages the fact that we can break the chain into two parts, the past and the future, by conditioning on  $z_t$ :

$$\gamma_t(j) \propto p(z_t = j, \mathbf{y}_{t+1:T} | \mathbf{y}_{1:t}) \propto p(z_t = j | \mathbf{y}_{1:t})p(\mathbf{y}_{t+1:T} | z_t = j, \mathbf{y}_{1:t}) \quad (8.42)$$

The first term is just  $\alpha_t(j) \triangleq p(z_t = j | \mathbf{y}_{1:t})$ , computed in the forwards filtering step. The second term is the conditional likelihood of future evidence given that the hidden state at time  $t$  is  $j$ . (We assume the hidden state is sufficient to explain future evidence for notational simplicity, but the algorithm still works if future observations depend on past ones.) which we denote as follows:

$$\beta_t(j) \triangleq p(\mathbf{y}_{t+1:T} | z_t = j) \quad (8.43)$$

(Note that  $\beta_t$  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  $\alpha_t$  as a prior and  $\beta_t$  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  $\alpha_t$  in a left-to-right fashion, as in Section 8.3.1. We now describe how to recursively compute the  $\beta$ 's in a right-to-left fashion. If we have already computed  $\beta_t$ , we can

1 compute  $\beta_{t-1}$  as follows:

2

$$\beta_{t-1}(i) = p(\mathbf{y}_{t:T} | z_{t-1} = i) \quad (8.46)$$

3

$$= \sum_j p(z_t = j, \mathbf{y}_t, \mathbf{y}_{t+1:T} | z_{t-1} = i) \quad (8.47)$$

4

$$= \sum_j p(\mathbf{y}_{t+1:T} | z_t = j, \mathbf{y}_t, \cancel{z_{t-1} = i}) p(z_t = j, \mathbf{y}_t | z_{t-1} = i) \quad (8.48)$$

5

$$= \sum_j p(\mathbf{y}_{t+1:T} | z_t = j) p(\mathbf{y}_t | z_t = j, \cancel{z_{t-1} = i}) p(z_t = j | z_{t-1} = i) \quad (8.49)$$

6

$$= \sum_j \beta_t(j) \lambda_t(j) A(i, j) \quad (8.50)$$

7 We can write the resulting equation in matrix-vector form as

8

$$\beta_{t-1} = \mathbf{A}(\boldsymbol{\lambda}_t \odot \beta_t) \quad (8.51)$$

9 The base case is

10

$$\beta_T(i) = p(\mathbf{y}_{T+1:T} | z_T = i) = p(\emptyset | z_T = i) = 1 \quad (8.52)$$

11 which is the probability of a non-event.

### 12 8.3.3.1 Another aside on normalization

13 Note that  $\beta_t(i)$  may underflow, since it is the conditional likelihood of an event (namely the particular sequence of observations  $\mathbf{y}_{t:T}$ ) from a large joint event space. Since only relative likelihoods matter, (since normalization constants will cancel out when we compute the posterior belief in Equation (8.44)), we can rescale the backwards messages by using

14

$$\beta_{t-1}(i) = \frac{1}{Z'_t} \sum_j \beta_t(j) \lambda_t(j) A(i, j) \quad (8.53)$$

15 where

16

$$Z'_t = \sum_i \sum_j \beta_t(j) \lambda_t(j) A(i, j) \quad (8.54)$$

17 Note that, unlike the case of the forwards filtering pass, where the normalization constants have some statistical meaning (see Equation (8.37)), in the backwards pass, they are just used for numerical stability.

18

### 19 8.3.3.2 Two-filter smoothing

20 Since the backwards pass needs access to the local evidence,  $\lambda_t$ , at each step (unlike the backwards smoothing step in Section 8.3.2), we can think of it as applying a filtering algorithm to the observations in the reverse-time direction. In the SSM literature, this approach is called **two-filter**

21

**smoothing** [Kit04], although in the HMM literature, it is called the forwards-backwards algorithm [Rab89]. The disadvantage of this approach compared to backwards smoothing is that the backwards messages are not probability distributions; this makes it harder to employ approximationg techniques for distributions, which is often necessary when working with continuous latent variables (see Section 8.4.4).

### 8.3.4 Two-slice smoothed marginals

When we estimate the parameters of the transition matrix (e.g., using EM, as described in Section 30.4.1), we need to compute the expected number of transitions from state  $i$  to state  $j$ :

$$N_{ij} = \sum_{t=1}^{T-1} \mathbb{E}[\mathbb{I}(z_t = i, z_{t+1} = j) | \mathbf{y}_{1:T}] = \sum_{t=1}^{T-1} p(z_t = i, z_{t+1} = j | \mathbf{y}_{1:T}) \quad (8.55)$$

The term  $p(z_t, z_{t+1} | \mathbf{y}_{1:T})$  is called a (smoothed) **two-slice marginal**. We can compute this from Equation (8.29) as follows:

$$p(z_t = i, z_{t+1} = j | \mathbf{y}_{1:T}) = p(z_t = i | \mathbf{y}_{1:t}) \sum_j \left[ \frac{p(z_{t+1} = j | z_t = i)p(z_{t+1} | \mathbf{y}_{1:T})}{p(z_{t+1} = j | \mathbf{y}_{1:t})} \right] \quad (8.56)$$

If we define  $\xi_{t,t+1}(i, j) \triangleq p(z_t = i, z_{t+1} = j | \mathbf{y}_{1:T})$ , we can write this as follows:

$$\xi_{t,t+1}(i, j) = \alpha_t(i) \sum_j A(i, j) \frac{\gamma_{t+1}(j)}{\alpha_{t+1|t}(j)} \quad (8.57)$$

where

$$\alpha_{t+1|t}(j) = p(z_{t+1} = j | \mathbf{y}_{1:t}) = \sum_i p(z_{t+1} = j | z_t = i)p(z_t = i | \mathbf{y}_{1:t}) \quad (8.58)$$

is the one-step-ahead predictive distribution. We can interpret the ratio in Equation (8.57) as dividing out the old estimate of  $z_{t+1}$  given  $\mathbf{y}_{1:t}$ , namely  $\alpha_{t+1|t}$ , and multiplying in the new estimate given  $\mathbf{y}_{1:T}$ , namely  $\gamma_{t+1}$ . See [SHJ97] for further insight into these equations.

The more traditional approach to deriving the two-slice marginals uses the output of the forwards backwards algorithm as follows:

$$p(z_t, z_{t+1} | \mathbf{y}_{1:T}) = p(z_t, z_{t+1} | \mathbf{y}_{1:t}, \mathbf{y}_{t+1:T}) \quad (8.59)$$

$$\propto p(\mathbf{y}_{t+1:T} | z_t, z_{t+1}, \mathbf{y}_{1:t}) p(z_t, z_{t+1} | \mathbf{y}_{1:t}) \quad (8.60)$$

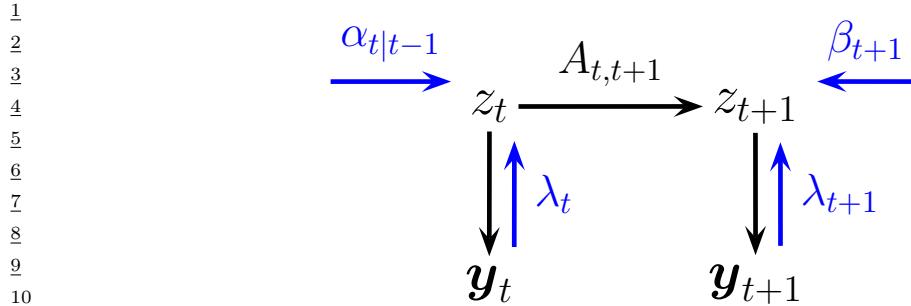
$$= p(\mathbf{y}_{t+1:T} | z_{t+1}) p(z_t, z_{t+1} | \mathbf{y}_{1:t}) \quad (8.61)$$

$$= p(\mathbf{y}_{t+1:T} | z_{t+1}) p(z_t | \mathbf{y}_{1:t}) p(z_{t+1} | z_t) \quad (8.62)$$

$$= p(\mathbf{y}_{t+1}, \mathbf{y}_{t+2:T} | z_{t+1}) p(z_t | \mathbf{y}_{1:t}) p(z_{t+1} | z_t) \quad (8.63)$$

$$= p(\mathbf{y}_{t+1} | z_{t+1}) p(\mathbf{y}_{t+2:T} | z_{t+1}, \mathbf{y}_{t+1}) p(z_t | \mathbf{y}_{1:t}) p(z_{t+1} | z_t) \quad (8.64)$$

$$= p(\mathbf{y}_{t+1} | z_{t+1}) p(\mathbf{y}_{t+2:T} | z_{t+1}) p(z_t | \mathbf{y}_{1:t}) p(z_{t+1} | z_t) \quad (8.65)$$



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:

$$\xi_{t,t+1}(i,j) \propto \lambda_{t+1}(j) \beta_{t+1}(j) \alpha_t(i) A(i,j) \quad (8.66)$$

19 Or in matrix-vector form:

$$\xi_{t,t+1} \propto \mathbf{A} \odot [\boldsymbol{\alpha}_t (\boldsymbol{\lambda}_{t+1} \odot \boldsymbol{\beta}_{t+1})^\top] \quad (8.67)$$

22 Since  $\boldsymbol{\alpha}_t \propto \boldsymbol{\lambda}_t \odot \boldsymbol{\alpha}_{t|t-1}$ , we can also write the above equation as follows:

$$\xi_{t,t+1} \propto (\boldsymbol{\lambda}_t \odot \boldsymbol{\alpha}_{t|t-1}) \odot \mathbf{A} \odot (\boldsymbol{\lambda}_{t+1} \odot \boldsymbol{\beta}_{t+1})^\top \quad (8.68)$$

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  $\boldsymbol{\alpha}_{t|t-1} = p(z_t | \mathbf{y}_{1:t-1})$ ,  $\mathbf{A} = p(z_t | z_{t-1})$ ,  $\boldsymbol{\lambda}_t \propto p(\mathbf{y}_t | z_t)$ ,  $\boldsymbol{\lambda}_{t+1} \propto p(\mathbf{y}_{t+1} | z_{t+1})$ , and  $\boldsymbol{\beta}_{t+1} \propto p(\mathbf{y}_{t+2:T} | z_{t+1})$   
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 It is clear that a straightforward implementation of the forwards-backwards algorithm takes  $O(K^2T)$   
33 time, since we must perform a  $K \times K$  matrix multiplication at each step. For some applications,  
34 such as speech recognition,  $K$  is very large, so the  $O(K^2)$  term becomes prohibitive. Fortunately,  
35 if the transition matrix is sparse, we can reduce this substantially. For example, in a left-to-right  
36 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

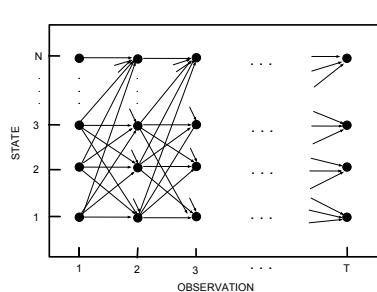


Figure 8.7: The trellis of states vs time for a Markov chain. Adapted from [Rab89].

In some cases, the bottleneck is memory, not time. In particular, to compute the posteriors  $\gamma_t(i)$ , we must store  $\alpha_t$  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  $\alpha_t$  and  $\beta_t$  vectors at a logarithmic number of intermediate checkpoints, and then recompute the missing messages on demand from these checkpoints. See [BMR97a; ZP00] for details.

### 8.3.6 The Viterbi algorithm

The MAP estimate is the most probable hidden sequence, given all the evidence:

$$\mathbf{z}_{1:T}^* = \underset{\mathbf{z}_{1:T}}{\operatorname{argmax}} p(\mathbf{z}_{1:T} | \mathbf{y}_{1:T}) \quad (8.69)$$

$$= \underset{\mathbf{z}_{1:T}}{\operatorname{argmax}} \log p(\mathbf{z}_{1:T} | \mathbf{y}_{1:T}) \quad (8.70)$$

$$= \underset{\mathbf{z}_{1:T}}{\operatorname{argmax}} \log \pi_1(z_1) + \log \lambda_1(z_1) + \sum_{t=2}^T [\log A(z_{t-1}, z_t) + \log \lambda_t(z_t)] \quad (8.71)$$

This is equivalent to computing a shortest path through the **trellis diagram** in Figure 8.7, where the nodes are possible states at each time step, and the node and edge weights are log probabilities. This can be computed in  $O(TK^2)$  time using the **Viterbi algorithm** [Vit67], as we explain below.

#### 8.3.6.1 Forwards pass

Recall the (unnormalized) forwards equation

$$\alpha'_t(j) = p(z_t = j, \mathbf{y}_{1:t}) = \sum_{z_1, \dots, z_{t-1}} p(\mathbf{z}_{1:t-1}, z_t = j, \mathbf{y}_{1:t}) \quad (8.72)$$

Now suppose we replace sum with max to get

$$\delta_t(j) \triangleq \max_{z_1, \dots, z_{t-1}} p(\mathbf{z}_{1:t-1}, z_t = j, \mathbf{y}_{1:t}) \quad (8.73)$$

1 This is the maximum probability we can assign to the data so far if we end up in state  $j$ . The key  
2 insight is that the most probable path to state  $j$  at time  $t$  must consist of the most probable path to  
3 some other state  $i$  at time  $t - 1$ , followed by a transition from  $i$  to  $j$ . Hence  
4

$$\underline{5} \quad \delta_t(j) = \max_i \delta_{t-1}(i) A(i, j) \lambda_t(j) \quad (8.74)$$

6 We initialize by setting  $\delta_1(j) = \pi_j \lambda_1(j)$ .

7 We also need keep track of the most likely previous (ancestor) state, for each possible state that  
8 we end up in:

$$\underline{9} \quad a_t(j) = \operatorname{argmax}_i \delta_{t-1}(i) A(i, j) \lambda_t(j) \quad (8.75)$$

10 That is,  $a_t(j)$  stores the identity of the previous state on the most probable path to  $z_t = j$ . We will  
11 see why we need this in Section 8.3.6.2.

### 12 8.3.6.2 Backwards pass

13 In the backwards pass, we compute the most probable sequence of states using a **traceback** procedure,  
14 as follows:  $z_t^* = a_{t+1}(z_{t+1}^*)$ , where we initialize using  $z_T^* = \arg \max_i \delta_T(i)$ . This is just following the  
15 chain of ancestors along the MAP path.

16 If there is a unique MAP estimate, the above procedure will give the same result as picking  
17  $\hat{z}_t = \operatorname{argmax}_j \gamma_t(j)$ , computed by forwards-backwards, as shown in [WF01b]. However, if there are  
18 multiple posterior modes, the latter approach may not find any of them, since it chooses each state  
19 independently, and hence may break ties in a manner that is inconsistent with its neighbors. The  
20 traceback procedure avoids this problem, since once  $z_t$  picks its most probable state, the previous  
21 nodes condition on this event, and therefore they will break ties consistently.

22 We are free to normalize the  $\delta_t$  terms at each step to avoid numerical underflow; this will not  
23 affect the maximum. However, since  $\log \max = \max \log$ , we can also do all computation in the log  
24 domain, which is often faster. Hence we can use

$$\underline{25} \quad \log \delta_t(j) \triangleq \max_{\mathbf{z}_{1:t-1}} \log p(\mathbf{z}_{1:t-1}, z_t = j | \mathbf{y}_{1:t}) \quad (8.76)$$

$$\underline{26} \quad = \max_i \log \delta_{t-1}(i) + \log A(i, j) + \log \lambda_t(j) \quad (8.77)$$

### 27 8.3.6.3 Example

28 Figure 8.8 gives a worked example of the Viterbi algorithm, based on [Rus+95]. Suppose we observe  
29 the sequence of discrete observations  $\mathbf{y}_{1:4} = (C_1, C_3, C_4, C_6)$ , representing codebook entries in a  
30 vector-quantized version of a speech signal. The model starts in state  $z_1 = S_1$ . The probability of  
31 generating  $x_1 = C_1$  in  $z_1$  is 0.5, so we have  $\delta_1(1) = 0.5$ , and  $\delta_1(i) = 0$  for all other states. Next we  
32 can self-transition to  $S_1$  with probability 0.3, or transition to  $S_2$  with probability 0.7. If we end up  
33 in  $S_1$ , the probability of generating  $x_2 = C_3$  is 0.3; if we end up in  $S_2$ , the probability of generating  
34  $x_2 = C_3$  is 0.2. Hence we have

$$\underline{35} \quad \delta_2(1) = \delta_1(1)A(1, 1)\lambda_2(1) = 0.5 \cdot 0.3 \cdot 0.3 = 0.045 \quad (8.78)$$

$$\underline{36} \quad \delta_2(2) = \delta_1(1)A(1, 2)\lambda_2(2) = 0.5 \cdot 0.7 \cdot 0.2 = 0.07 \quad (8.79)$$

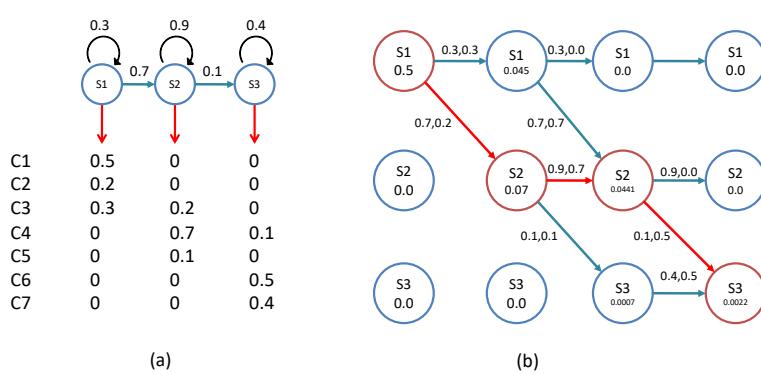


Figure 8.8: Illustration of Viterbi decoding in a simple HMM for speech recognition. (a) A 3-state HMM for a single phone. We are visualizing the state transition diagram. We assume the observations have been vector quantized into 7 possible symbols,  $C_1, \dots, C_7$ . Each state  $s_1, s_2, s_3$  has a different distribution over these symbols. Adapted from Figure 15.20 of [RN02]. (b) Illustration of the Viterbi algorithm applied to this model, with data sequence  $C_1, C_3, C_4, C_6$ . The columns represent time, and the rows represent states. The numbers inside the circles represent the  $\delta_t(j)$  value for that state. An arrow from state  $i$  at  $t - 1$  to state  $j$  at  $t$  is annotated with two numbers: the first is the probability of the  $i \rightarrow j$  transition, and the second is the probability of generating observation  $y_t$  from state  $j$ . The red lines/ circles represent the most probable sequence of states. Adapted from Figure 24.27 of [RN95].

Thus state 2 is more probable at  $t = 2$ ; see the second column of Figure 8.8(b). The algorithm continues in this way until we have reached the end of the sequence. Once we have reached the end, we can follow the red arrows back to recover the MAP path (which is 1,2,2,3).

For more details on HMMs for automatic speech recognition (ASR) see e.g., [JM08].

#### 8.3.6.4 Time and space complexity

The time complexity of Viterbi is clearly  $O(K^2T)$  in general, and the space complexity is  $O(KT)$ , both the same as forwards-backwards. If the transition matrix has the form  $A(i, j) \propto \exp(-\sigma^2 \|\mathbf{z}_i - \mathbf{z}_j\|^2)$ , where  $\mathbf{z}_i$  is the continuous vector represented by state  $i$ , we can implement Viterbi in  $O(TK)$  time, instead of  $O(TK \log K)$  needed by forwards-backwards. See [FHK03] for details.

#### 8.3.6.5 N-best list

There are often multiple paths which have the same likelihood. The Viterbi algorithm returns one of them, but can be extended to return the top  $N$  paths [SC90; NG01]. This is called the **N-best list**. Computing such a list,  $\mathcal{L}_N$ , can provide a better summary of the posterior uncertainty.

In addition, we can perform **discriminative reranking** [CK05] of all the sequences in  $\mathcal{L}_N$ , based on global features derived from  $(\mathbf{y}_{1:T}, \mathbf{z}_{1:T})$ . This technique is widely used in speech recognition. For example, consider the sentence “recognize speech”. It is possible that the most probable interpretation by the system of this acoustic signal is “wreck a nice speech”, or maybe “wreck a nice beach”. Maybe the correct interpretation is much lower down on the list. However, by using a re-ranking system, we

1 may be able to improve the score of the correct interpretation based on a more global context.  
2

3 One problem with the  $N$ -best list is that often the top  $N$  paths are very similar to each other,  
4 rather than representing qualitatively different interpretations of the data. Instead we might want to  
5 generate a more diverse set of paths to more accurately represent posterior uncertainty. One way  
6 to do this is to sample paths from the posterior, as we discuss in Section 8.3.7. Another way is to  
7 use a determinantal point process [KT11b; ZA12], which encourages points to be diverse (see also  
8 [YBS11]).  
9

### 10 8.3.7 Forwards filtering, backwards sampling

11 Rather than computing the single most probable path, it is often useful to sample multiple paths from  
12 the posterior:  $\mathbf{z}_{1:T}^s \sim p(\mathbf{z}_{1:T} | \mathbf{y}_{1:T})$ . We can do this by modifying the forwards filtering backwards  
13 smoothing algorithm from Section 8.3.2, so that we draw samples on the backwards pass, rather  
14 than computing marginals. This is called **forwards filtering backwards sampling** (FFBS). In  
15 particular, note that we can write the joint from right to left using  
16

$$\underline{17} \quad p(\mathbf{z}_{1:T} | \mathbf{y}) = p(z_T | \mathbf{y}) p(z_{T-1} | z_T, \mathbf{y}) p(z_{T-2} | z_{T-1}, \mathbf{y}) \cdots p(z_1 | z_2, z_{3:T}, \mathbf{y}) \quad (8.80)$$

$$\underline{18} \quad = p(z_T | \mathbf{y}) \prod_{t=T-1}^1 p(z_t | z_{t+1}, \mathbf{y}) \quad (8.81)$$

22 where we write  $\mathbf{y} = \mathbf{y}_{1:T}$  for brevity. Thus we can sample trajectories backwards. At step  $t$ , we  
23 sample from the backwards posterior conditional

$$\underline{24} \quad p(z_t = i | z_{t+1} = j, \mathbf{y}_{1:T}) = p(z_t = i | z_{t+1} = j, \mathbf{y}_{1:t}, \mathbf{y}_{t+1:T}) \quad (8.82)$$

$$\underline{25} \quad = \frac{p(z_t = i, z_{t+1} = j | \mathbf{y}_{1:t})}{p(z_{t+1} = j | \mathbf{y}_{1:t})} \quad (8.83)$$

$$\underline{26} \quad = \frac{p(z_{t+1} = j | z_t = i) p(z_t = i | \mathbf{y}_{1:t})}{p(z_{t+1} = j | \mathbf{y}_{1:t})} \quad (8.84)$$

$$\underline{27} \quad = \frac{A(i, j) \alpha_t(i)}{\sum_{i'} A(i', j) \alpha_t(i')} \quad (8.85)$$

34 The base case is  $p(z_T = i | \mathbf{y}_{1:T}) = \alpha_T(i)$ .  
35

### 36 8.3.8 Application to discretized state spaces

38 Exact inference in SSMS with continuous latent states is in general intractable, except for the special  
39 case of linear Gaussian models (which we discuss in Section 8.4.1). For low dimensional problems, we  
40 can discretize the latent variables into a grid, and then run the HMM filter and smoother. This can  
41 be a useful debugging tool. See [RG17] for some examples in the 1d case.

42 In 2d, the number of states is typically too large to be practical, since the HMM filter takes  
43  $O(K^2)$  time per step, where  $K = G_1 G_2$  is the number of states, and  $G_1$  and  $G_2$  are the grid sizes in  
44 dimensions 1 and 2. However, for certain problems, it is possible to leverage sparse or convolutional  
45 structure in the transition matrix to perform the computation in  $O(K)$  time per step [MHS03;  
46 FHK03].  
47

In higher dimensions, this discretization strategy is usually intractable, so other approximations are needed, as we discuss later on.

## 8.4 Inference for linear-Gaussian SSMs

In this section, we discuss inference in linear-Gaussian state space models, which we introduced in Section 8.1.3. This corresponds to the following model:

$$p(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{u}_t) = \mathcal{N}(\mathbf{z}_t | \mathbf{A}_t \mathbf{z}_{t-1} + \mathbf{B}_t \mathbf{u}_t, \mathbf{Q}_t) \quad (8.86)$$

$$p(\mathbf{y}_t | \mathbf{z}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{y}_t | \mathbf{C}_t \mathbf{z}_t + \mathbf{D}_t \mathbf{u}_t, \mathbf{R}_t) \quad (8.87)$$

An LG-SSM is just a special case of a Gaussian Bayes net (Section 4.2.2.3), so the entire joint distribution  $p(\mathbf{y}_{1:T}, \mathbf{z}_{1:T} | \mathbf{u}_{1:T})$  is a large multivariate Gaussian with  $DKT$  dimensions. However, it has special structure that makes it computationally tractable to use, as we show below. (To simplify the notation, we drop the conditioning on the inputs  $\mathbf{u}_t$ , and we assume the parameters are known.)

### 8.4.1 The Kalman filter

The **Kalman filter** is an algorithm for exact Bayesian filtering for linear-Gaussian state space models. The resulting algorithm is the Gaussian analog of the HMM filter in Section 8.3.1, except the belief state at time  $t$  is now given by  $p(\mathbf{z}_t | \mathbf{y}_{1:t}) = \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ . Since everything is Gaussian, we can perform the prediction and update steps in closed form, as we explain below.<sup>4</sup>

#### 8.4.1.1 Predict step

The one-step-ahead prediction for the hidden state, also called the **time update step**, is given by the following:<sup>5</sup>

$$p(\mathbf{z}_t | \mathbf{y}_{1:t-1}) = \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}) \quad (8.88)$$

$$\boldsymbol{\mu}_{t|t-1} = \mathbf{A}_t \boldsymbol{\mu}_{t-1} \quad (8.89)$$

$$\boldsymbol{\Sigma}_{t|t-1} = \mathbf{A}_t \boldsymbol{\Sigma}_{t-1} \mathbf{A}_t^\top + \mathbf{Q}_t \quad (8.90)$$

4. Note on notation: some authors represent the posterior mean and covariance by  $\mathbf{m}_t$  and  $\mathbf{P}_t$ , but we use  $\boldsymbol{\mu}_t$  and  $\boldsymbol{\Sigma}_t$ , to be consistent with the rest of the book.

5. Some authors write  $\boldsymbol{\mu}_t^-$  and  $\boldsymbol{\Sigma}_t^-$  to represent the one-step-ahead posterior predictive distribution,  $p(\mathbf{z}_t | \mathbf{y}_{1:t-1})$ . We use the notation  $\boldsymbol{\mu}_{t|t-1}$  and  $\boldsymbol{\Sigma}_{t|t-1}$ , to indicate that we are conditioning on observations up to step  $t-1$ . This notation will generalize nicely to the smoothing case. Note, however, that for shorthand, we write  $\boldsymbol{\mu}_t$  and  $\boldsymbol{\Sigma}_t$  rather than  $\boldsymbol{\mu}_{t|t}$  and  $\boldsymbol{\Sigma}_{t|t}$ .

### 8.4.1.2 Update step

The update step (also called the **measurement step**) can be computed using Bayes rule, as follows:

$$p(\mathbf{z}_t | \mathbf{y}_{1:t}) = \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t) \quad (8.91)$$

$$\boldsymbol{e}_t = \boldsymbol{y}_t - \mathbf{C}_t \boldsymbol{\mu}_{t|t-1} \quad (8.92)$$

$$\mathbf{S}_t = \mathbf{C}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{C}_t^\top + \mathbf{R}_t \quad (8.93)$$

$$\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1} \mathbf{C}_t^\top \mathbf{S}_t^{-1} \quad (8.94)$$

$$\mu_t = \mu_{t|t-1} + \mathbf{K}_t e_t \quad (8.95)$$

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t \mathbf{C}_t \boldsymbol{\Sigma}_{t|t-1} \quad (8.96)$$

$$= \Sigma_{t|t-1} - \mathbf{K}_t \mathbf{S}_t \mathbf{K}_t^T \quad (8.97)$$

(Proving the equivalence of Equation (8.96) and Equation (8.97) is left as an exercise.)

The normalization constant of the new posterior can be computed as follows:

$$p(\mathbf{y}_t | \mathbf{y}_{1:t-1}) = \int p(\mathbf{y}_t, \mathbf{z}_t | \mathbf{y}_{1:t-1}) d\mathbf{z}_t = \mathcal{N}(\mathbf{y}_t | \mathbf{C}_t \boldsymbol{\mu}_{t|t-1}, \mathbf{S}_t) \quad (8.98)$$

#### 8.4.1.3 Intuition

22 To understand these equations intuitively, note that  $e_t = y_t - \hat{y}_{t|t-1}$  is the difference between our predicted observation  $\hat{y}_{t|t-1}$  and the actual observation  $y_t$ ; this is called the **residual** or **innovation**.  
23 The update for the mean is given by  $\mu_t = \mu_{t|t-1} + K_t e_t$ , which is the predicted new mean plus a  
24 correction factor, which is  $K_t$  times the error signal  $e_t$ .

26 The amount of weight placed on the error signal depends on the **Kalman gain matrix**,  $\mathbf{K}_t$ . By  
 27 using the matrix inversion lemma, the Kalman gain matrix can also be written as

$$\mathbf{K}_t = \Sigma_{t|t-1} \mathbf{C}_t^\top (\mathbf{C}_t \Sigma_{t|t-1} \mathbf{C}_t^\top + \mathbf{R}_t)^{-1} = (\Sigma_{t|t-1}^{-1} + \mathbf{C}_t^\top \mathbf{R}_t^{-1} \mathbf{C}_t)^{-1} \mathbf{C}_t^\top \mathbf{R}_t^{-1} \quad (8.99)$$

If  $\mathbf{C}_t = \mathbf{I}$ , then  $\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1} \mathbf{S}_t^{-1}$ , which is the ratio between the covariance of the prior (from the dynamic model) and the covariance of the measurement,  $\mathbf{S}_t$ . If we have a strong prior and/or very noisy sensors,  $|\mathbf{K}_t|$  will be small, and we will place little weight on the correction term. Conversely, if we have a weak prior and/or high precision sensors, then  $|\mathbf{K}_t|$  will be large, and we will place a lot of weight on the correction term. Similarly, the new covariance is the old covariance minus a positive definite matrix, which depends on how informative the measurement is.

#### 8.4.1.4 Derivation

<sup>39</sup> In this section we derive the Kalman filter equations, following [Sar13, p57]. From Equation (2.56),  
<sup>40</sup> the joint predictive distribution for states is given by

$$p(\tilde{z}_{t-1}, z_t | \mathbf{u}_{1:t-1}) = p(z_t | \tilde{z}_{t-1}) p(\tilde{z}_{t-1} | \mathbf{u}_{1:t-1}) \quad (8.100)$$

$$= \mathcal{N}(\mathbf{z}_t | \mathbf{A}_t \mathbf{z}_{t-1}, \mathbf{Q}_{t-1}) \mathcal{N}(\mathbf{z}_{t-1} | \boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}) \quad (8.101)$$

$$= \mathcal{N} \left( \begin{pmatrix} z_{t-1} \\ z_t \end{pmatrix} | \boldsymbol{\mu}', \boldsymbol{\Sigma}' \right) \quad (8.102)$$

1  
2 where

3  
4  $\boldsymbol{\mu}' = \begin{pmatrix} \boldsymbol{\mu}_{t-1} \\ \mathbf{C}_{t-1}\boldsymbol{\mu}_{t-1} \end{pmatrix}, \boldsymbol{\Sigma}' = \begin{pmatrix} \boldsymbol{\Sigma}_{t-1} & \boldsymbol{\Sigma}_{t-1}\mathbf{A}_t^\top \\ \mathbf{A}_t\boldsymbol{\Sigma}_{t-1} & \mathbf{A}_t\boldsymbol{\Sigma}_{t-1}\mathbf{A}_t^\top + \mathbf{Q}_{t-1} \end{pmatrix}$  (8.103)

5  
6 From Equation (2.62), the marginal distribution is

7  
8  $p(\mathbf{z}_t | \mathbf{y}_{1:t-1}) = \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1})$  (8.104)

9  
10 This gives us the predict step.

11 Now for the measurement update step. From Equation (2.56), the joint distribution for state and  
12 observation is given by

13  
14  $p(\mathbf{z}_t, \mathbf{y}_t) = p(\mathbf{y}_t | \mathbf{z}_t)p(\mathbf{z}_t | \mathbf{y}_{1:t-1})$  (8.105)

15  
16  $= \mathcal{N}(\mathbf{y}_t | \mathbf{C}_t \mathbf{z}_t, \mathbf{R}_t) \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1})$  (8.106)

17  
18  $= \mathcal{N} \left( \begin{pmatrix} \mathbf{z}_t \\ \mathbf{y}_t \end{pmatrix} | \boldsymbol{\mu}'', \boldsymbol{\Sigma}'' \right)$  (8.107)

19  
20 where

21  
22  $\boldsymbol{\mu}'' = \begin{pmatrix} \boldsymbol{\mu}_{t|t-1} \\ \mathbf{C}_t \boldsymbol{\mu}_{t|t-1} \end{pmatrix}, \boldsymbol{\Sigma}'' = \begin{pmatrix} \boldsymbol{\Sigma}_{t|t-1} & \boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^\top \\ \mathbf{C}_t\boldsymbol{\Sigma}_{t|t-1} & \mathbf{C}_t\boldsymbol{\Sigma}_{t|t-1}^{-1}\mathbf{C}_t^\top + \mathbf{R}_t \end{pmatrix}$  (8.108)

23  
24 Finally, we convert this joint into a conditional using Equation (2.50) as follows:

25  
26  $p(\mathbf{z}_t | \mathbf{y}_t, \mathbf{y}_{1:t-1}) = \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$  (8.109)

27  
28  $\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^\top (\mathbf{C}_t\boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^\top + \mathbf{R}_t)^{-1}[\mathbf{y}_t - \mathbf{C}_t\boldsymbol{\mu}_{t|t-1}]$  (8.110)

29  
30  $\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t|t-1} - \boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^\top (\mathbf{C}_t\boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^\top + \mathbf{R}_t)^{-1}\mathbf{C}_t\boldsymbol{\Sigma}_{t|t-1}$  (8.111)

31  
32 which can be shown to equal Equation (8.97).

#### 33 34 8.4.1.5 Steady state solution

35  
36 One surprising thing about linear-Gaussian systems is that the posterior covariance is independent of  
37 the observations, as we see from Equation (8.97). We can therefore precompute  $\boldsymbol{\Sigma}_t$  for all  $t$ . The  
38 iterative equations for updating  $\boldsymbol{\Sigma}_t$  are called the **Riccati equations**, and for time invariant systems,  
39 they converge to a fixed point, namely  $\boldsymbol{\Sigma} = \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^\top + \mathbf{Q}$ . The corresponding steady state Kalman  
40 gain matrix is therefore

41  
42  $\mathbf{K} \triangleq \boldsymbol{\Sigma}\mathbf{C}^\top(\mathbf{C}\boldsymbol{\Sigma}\mathbf{C}^\top + \mathbf{R})$  (8.112)

43  
44 This is often precomputed, to save time. The corresponding update for the posterior mean becomes

45  
46  $\boldsymbol{\mu}_t = (\mathbf{A} - \mathbf{K}\mathbf{C}\mathbf{A})\boldsymbol{\mu}_{t-1} + \mathbf{K}\mathbf{y}_t$  (8.113)

1    **8.4.1.6 Posterior predictive**

3    The one-step-ahead posterior predictive density for the observations can be computed as follows.  
4    First we compute the one-step-ahead predictive density for latent states:

$$\underline{6} \quad p(\mathbf{z}_t | \mathbf{y}_{1:t-1}) = \mathcal{N}(\mathbf{z}_t | \mathbf{A}_t \boldsymbol{\mu}_{t-1}, \mathbf{A}_t \boldsymbol{\Sigma}_{t-1} \mathbf{A}_t^\top + \mathbf{Q}_t) \quad (8.114)$$

7    Then we convert this to a prediction about observations by marginalizing out  $\mathbf{z}_t$ :

$$\underline{9} \quad p(\mathbf{y}_t | \mathbf{y}_{1:t-1}) = \mathcal{N}(\mathbf{y}_t | \mathbf{C} \boldsymbol{\mu}_{t|t-1}, \mathbf{C} \boldsymbol{\Sigma}_{t|t-1} \mathbf{C}^\top + \mathbf{R}_t) \quad (8.115)$$

11    We can generalize this to predict observations  $K$  steps into the future by first forecasting  $K$  stpes  
12 in latent space, and then “grounding” the final state into predicted observations. (This is in contrast  
13 to an RNN (Section 16.3.3), which requires generating observations at each step, in order to update  
14 future hidden states.) For notational simplicity, we just show how to do this for  $K = 2$ , i.e., we want  
15 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  
16 to know future inputs if the model is conditional.

17    We start with the prior, which is the previous posterior:

$$\underline{19} \quad p(\mathbf{z}_{t-1} | \mathbf{y}_{1:t-1}) = \mathcal{N}(\mathbf{z}_{t-1} | \boldsymbol{\mu}_{t-1|t-1}, \boldsymbol{\Sigma}_{t-1|t-1}) \quad (8.116)$$

20    We predict the current hidden state as follows:

$$\underline{22} \quad p(\mathbf{z}_t | \mathbf{y}_{1:t-1}) = \int \mathcal{N}(\mathbf{z}_t | \mathbf{A} \mathbf{z}_{t-1}, \mathbf{Q}) \mathcal{N}(\mathbf{z}_{t-1} | \boldsymbol{\mu}_{t-1|t-1}, \boldsymbol{\Sigma}_{t-1|t-1}) d\mathbf{z}_{t-1} \quad (8.117)$$

$$\underline{24} \quad = \mathcal{N}(\mathbf{z}_t | \mathbf{A} \boldsymbol{\mu}_{t-1|t-1}, \mathbf{A} \boldsymbol{\Sigma}_{t-1|t-1} \mathbf{A}^\top + \mathbf{Q}) \quad (8.118)$$

$$\underline{26} \quad \triangleq \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}) \quad (8.119)$$

27    We predict the next hidden state as follows:

$$\underline{29} \quad p(\mathbf{z}_{t+1} | \mathbf{y}_{1:t-1}) = \int \mathcal{N}(\mathbf{z}_{t+1} | \mathbf{A} \mathbf{z}_t, \mathbf{Q}) \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}) d\mathbf{z}_t \quad (8.120)$$

$$\underline{31} \quad = \mathcal{N}(\mathbf{z}_{t+1} | \mathbf{A} \boldsymbol{\mu}_{t|t-1}, \mathbf{A} \boldsymbol{\Sigma}_{t|t-1} \mathbf{A}^\top + \mathbf{Q}) \quad (8.121)$$

$$\underline{33} \quad \triangleq \mathcal{N}(\mathbf{z}_{t+1} | \boldsymbol{\mu}_{t+1|t-1}, \boldsymbol{\Sigma}_{t+1|t-1}) \quad (8.122)$$

34    Finally we predict the next observation as follows:

$$\underline{36} \quad p(\mathbf{y}_{t+1} | \mathbf{y}_{1:t-1}) = \int \mathcal{N}(\mathbf{y}_{t+1} | \mathbf{C} \mathbf{z}_{t+1}, \mathbf{R}) \mathcal{N}(\mathbf{z}_{t+1} | \boldsymbol{\mu}_{t+1|t-1}, \boldsymbol{\Sigma}_{t+1|t-1}) d\mathbf{z}_{t+1} \quad (8.123)$$

$$\underline{38} \quad = \mathcal{N}(\mathbf{y}_{t+1} | \mathbf{C} \boldsymbol{\mu}_{t+1|t-1}, \mathbf{C} \boldsymbol{\Sigma}_{t+1|t-1} \mathbf{C}^\top + \mathbf{R}) \quad (8.124)$$

40    **8.4.1.7 Numerical issues**

42    In practice, the Kalman filter often encounters numerical issues, primarily related to the need to  
43 compute and invert the posterior covariance matrix. Recall from Equation (8.97) that the update  
44 step involves computing

$$\underline{46} \quad \boldsymbol{\Sigma}_t = (\mathbf{I} - \mathbf{K}_t \mathbf{C}_t) \boldsymbol{\Sigma}_{t|t-1} \quad (8.125)$$

47

However, the subtraction  $\mathbf{I} - \mathbf{K}_t \mathbf{C}_t$  can lead to nonsymmetric matrices, due to floating point errors. In [BH12; Lab18], they propose to replace this with the mathematically equivalent expression, which is more numerically stable:

$$\Sigma_t = (\mathbf{I} - \mathbf{K}_t \mathbf{C}_t) \Sigma_{t|t-1} (\mathbf{I} - \mathbf{K}_t \mathbf{C}_t)^T + \mathbf{K}_t \mathbf{R}_t \mathbf{K}_t^T \quad (8.126)$$

To see why this equation is true, first recall that the posterior mean estimate is given by

$$\mu_{t|t} = \mu_{t|t-1} + \mathbf{K}_t (\mathbf{y}_t - \mathbf{h}_t \mu_{t|t-1}) \quad (8.127)$$

We have that  $\mathbf{y}_t = \mathbf{C}_t \mathbf{z}_t + \epsilon_t$ , where  $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_t)$  is the observation noise. Hence we have the following, for any matrix  $\mathbf{K}$ :

$$\mathbf{z}_t - \mu_{t|t} = \mathbf{z}_t - (\mu_{t|t-1} + \mathbf{K}(\mathbf{y}_t - \mathbf{C}_t \mu_{t|t-1})) \quad (8.128)$$

$$= \mathbf{z}_t - (\mu_{t|t-1} + \mathbf{K}(\mathbf{C}_t \mathbf{z}_t + \epsilon_t - \mathbf{C}_t \mu_{t|t-1})) \quad (8.129)$$

$$= (\mathbf{z}_t - \mu_{t|t-1}) - \mathbf{K} \mathbf{C}_t (\mathbf{z}_t - \mu_{t|t-1}) - \mathbf{K} \epsilon_t \quad (8.130)$$

$$= (\mathbf{I} - \mathbf{K} \mathbf{C}_t) (\mathbf{z}_t - \mu_{t|t-1}) - \mathbf{K} \epsilon_t \quad (8.131)$$

Hence the posterior covariance of  $\mathbf{z}_t$  is given by

$$\Sigma_{t|t} = \mathbb{E}[(\mathbf{z}_t - \mu_{t|t})(\mathbf{z}_t - \mu_{t|t})^T] \quad (8.132)$$

$$= \mathbb{E}[[(\mathbf{I} - \mathbf{K} \mathbf{C}_t)(\mathbf{z}_t - \mu_{t|t-1}) - \mathbf{K} \epsilon_t][(\mathbf{I} - \mathbf{K} \mathbf{C}_t)(\mathbf{z}_t - \mu_{t|t-1}) - \mathbf{K} \epsilon_t]^T] \quad (8.133)$$

$$= (\mathbf{I} - \mathbf{K} \mathbf{C}_t) \Sigma_{t|t-1} (\mathbf{I} - \mathbf{K} \mathbf{C}_t)^T + \mathbf{K} \mathbf{R}_t \mathbf{K}^T \quad (8.134)$$

There are other solutions that can be used. One approach is the **information filter**, which recursively updates the natural parameters of the Gaussian,  $\Lambda_t = \Sigma_t^{-1}$  and  $\eta_t = \Lambda_t \mu_t$ , instead of the mean and covariance. Another approach is the **square root filter**, which works with the Cholesky or QR decomposition of  $\Sigma_t$ , which is much more numerically stable than directly updating  $\Sigma_t$ . These techniques can be combined to create the **square root information filter** (SRIF) [May79]. According to [Bie06], the SRIF was developed in 1969 for use in JPL's Mariner 10 mission to Venus.

#### 8.4.1.8 Handling unknown observation noise

In the case of scalar observations (as often arises in time series forecasting), we can extend the Kalman filter to handle the common situation in which the observation noise variance  $V = \sigma^2$  is unknown, as described in [WH97, Sec 4.6]. The model is defined as follows:

$$p(\mathbf{z}_t | \mathbf{z}_{t-1}) = \mathcal{N}(\mathbf{z}_t | \mathbf{A}_t \mathbf{z}_{t-1}, V \mathbf{Q}_t^*) \quad (8.135)$$

$$p(y_t | \mathbf{z}_t) = \mathcal{N}(y_t | \mathbf{h}_t^T \mathbf{z}_t, V) \quad (8.136)$$

where  $\mathbf{Q}_t^*$  is the unscaled system noise, and we define  $\mathbf{C}_t = \mathbf{h}_t^T$  to be the vector that maps the hidden state vector to the scalar observation. Let  $\lambda = 1/V$  be the observation precision. To start the algorithm, we use the following prior:

$$p_0(\lambda) = \text{Ga}\left(\frac{\nu_0}{2}, \frac{\nu_0 \tau_0}{2}\right) \quad (8.137)$$

$$p_0(\mathbf{z} | \lambda) = \mathcal{N}(\mu_0, V \Sigma_0^*) \quad (8.138)$$

1 where  $\tau_0$  is the prior mean for  $\sigma^2$ , and  $\nu_0 > 0$  is the strength of this prior.  
 2

3 We now discuss the belief updating step. We assume that the prior belief state at time  $t - 1$  is  
 4

$$5 \quad \mathcal{N}(\mathbf{z}_{t-1}, \lambda | \mathcal{D}_{1:t-1}) = \mathcal{N}(\mathbf{z}_{t-1} | \boldsymbol{\mu}_{t-1}, V \boldsymbol{\Sigma}_{t-1}^*) \text{Ga}(\lambda | \frac{\nu_{t-1}}{2}, \frac{\nu_{t-1} \tau_{t-1}}{2}) \quad (8.139)$$

6 The posterior is given by  
 7

$$8 \quad \mathcal{N}(\mathbf{z}_t, \lambda | \mathcal{D}_{1:t}) = \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_t, V \boldsymbol{\Sigma}_t^*) \text{Ga}(\lambda | \frac{\nu_t}{2}, \frac{\nu_t \tau_t}{2}) \quad (8.140)$$

10 where  
 11

$$12 \quad \boldsymbol{\mu}_{t|t-1} = \mathbf{A}_t \boldsymbol{\mu}_{t-1} \quad (8.141)$$

$$13 \quad \boldsymbol{\Sigma}_{t|t-1}^* = \mathbf{A}_t \boldsymbol{\Sigma}_{t-1}^* \mathbf{A}_t + \mathbf{Q}_t^* \quad (8.142)$$

$$14 \quad e_t = y_t - \mathbf{h}_t^\top \boldsymbol{\mu}_{t|t-1} \quad (8.143)$$

$$15 \quad s_t^* = \mathbf{h}_t^\top \boldsymbol{\Sigma}_{t|t-1}^* \mathbf{h}_t + 1 \quad (8.144)$$

$$16 \quad \mathbf{k}_t = \boldsymbol{\Sigma}_{t|t-1}^* \mathbf{h}_t / s_t^* \quad (8.145)$$

$$17 \quad \boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + \mathbf{k}_t e_t \quad (8.146)$$

$$18 \quad \boldsymbol{\Sigma}_t^* = \boldsymbol{\Sigma}_{t|t-1}^* - \mathbf{k}_t \mathbf{k}_t^\top s_t^* \quad (8.147)$$

$$19 \quad \nu_t = \nu_{t-1} + 1 \quad (8.148)$$

$$20 \quad \nu_t \tau_t = \nu_{t-1} \tau_{t-1} + e_t^2 / s_t^* \quad (8.149)$$

24 If we marginalize out  $V$ , the marginal distribution for  $\mathbf{z}_t$  is a Student distribution:  
 25

$$26 \quad p(\mathbf{z}_t | \mathcal{D}_{1:t}) = \mathcal{T}_{\nu_t}(\mathbf{z}_t | \boldsymbol{\mu}_t, \tau_t \boldsymbol{\Sigma}_t^*) \quad (8.150)$$

### 28 8.4.2 Kalman filtering for linear regression (recursive least squares)

29 In Section 15.2.1, we discuss how to compute  $p(\mathbf{w} | \sigma^2, \mathcal{D})$  for a linear regression model in batch mode,  
 30 using a Gaussian prior of the form  $p(\mathbf{w}) = \mathcal{N}(\mathbf{w} | \boldsymbol{\mu}, \boldsymbol{\Sigma})$ . In this section, we discuss how to compute  
 31 this posterior online, by repeatedly performing the following update:  
 32

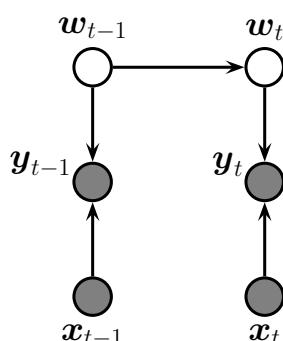
$$33 \quad p(\mathbf{w} | \mathcal{D}_{1:t}) \propto p(\mathcal{D}_t | \mathbf{w}) p(\mathbf{w} | \mathcal{D}_{1:t-1}) \quad (8.151)$$

$$34 \quad \propto p(\mathcal{D}_t | \mathbf{w}) p(\mathcal{D}_{t-1} | \mathbf{w}) \dots p(\mathcal{D}_1 | \mathbf{w}) p(\mathbf{w}) \quad (8.152)$$

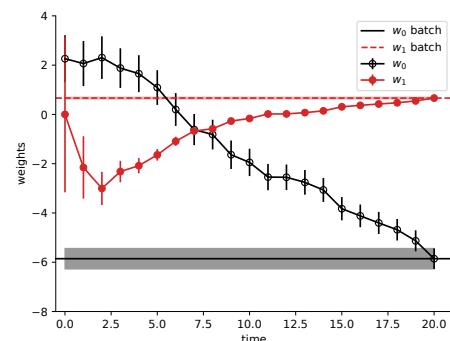
36 where  $\mathcal{D}_t = (\mathbf{x}_t, y_t)$  is the  $t$ 'th labeled example, and  $\mathcal{D}_{1:t-1}$  are the first  $t - 1$  examples. (For brevity,  
 37 we drop the conditioning on  $\sigma^2$ .) We see that the previous posterior,  $p(\mathbf{w} | \mathcal{D}_{1:t-1})$ , becomes the  
 38 current prior, which gets updated by  $\mathcal{D}_t$  to become the new posterior,  $p(\mathbf{w} | \mathcal{D}_{1:t})$ . This is an example  
 39 of sequential Bayesian updating or online Bayesian inference. In the case of linear regression, this  
 40 process is known as the **recursive least squares** or **RLS** algorithm.

41 We can implement this method by using a linear Gaussian state space model (Section 31.2). The  
 42 basic idea is to let the hidden state represent the regression parameters, and to let the (time-varying)  
 43 observation model represent the current data vector. If we assume the regression parameters do not  
 44 change, the dynamics model becomes  
 45

$$46 \quad p(\mathbf{w}_t | \mathbf{w}_{t-1}) = \mathcal{N}(\mathbf{w}_t | \mathbf{w}_{t-1}, 0) = \delta(\mathbf{w}_t - \mathbf{w}_{t-1}) \quad (8.153)$$



(a)



(b)

16 *Figure 8.9: (a) A dynamic generalization of linear regression. (b) Illustration of the recursive least squares*  
 17 *algorithm applied to the model  $p(y|x, \mathbf{w}) = \mathcal{N}(y|w_0 + w_1 x, \sigma^2)$ . We plot the marginal posterior of  $w_0$  and  $w_1$*   
 18 *vs number of data points. (Error bars represent  $\mathbb{E}[w_j|y_{1:t}, \mathbf{x}_{1:t}] \pm \sqrt{\text{Var}[w_j|y_{1:t}, \mathbf{x}_{1:t}]}$ .) 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.py](#).*

22 (If we do let the parameters change over time, we get a so-called **dynamic linear model** [Har90;  
 23 WH97; PPC09].) The (non-stationary) observation model has the form

$$25 \quad p(y_t|\mathbf{w}_t, \mathbf{x}_t) = \mathcal{N}(y_t|\mathbf{C}_t z_t, \mathbf{R}_t) = \mathcal{N}(y_t|\mathbf{x}_t^\top \mathbf{w}_t, \sigma^2) \quad (8.154)$$

27 See Figure 8.9a for the model.

28 Recall from Section 8.4.1 that the Kalman filter equations are as follows:

$$29 \quad \Sigma_{t|t-1} = \mathbf{A}_t \Sigma_{t-1} \mathbf{A}_t^\top + \mathbf{Q}_t \quad (8.155)$$

$$31 \quad \mathbf{S}_t = \mathbf{C}_t \Sigma_{t|t-1} \mathbf{C}_t^\top + \mathbf{R}_t \quad (8.156)$$

$$32 \quad \mathbf{K}_t = \Sigma_{t|t-1} \mathbf{C}_t^\top \mathbf{S}_t^{-1} \quad (8.157)$$

$$34 \quad \boldsymbol{\mu}_t = \mathbf{A}_{t-1} \boldsymbol{\mu}_{t-1} + \mathbf{K}_t (y_t - \mathbf{C}_t \mathbf{A}_{t-1} \boldsymbol{\mu}_{t-1}) \quad (8.158)$$

$$35 \quad \Sigma_t = (\mathbf{I} - \mathbf{K}_t \mathbf{C}_t) \Sigma_{t|t-1} \quad (8.159)$$

36 In the case of RLS we have  $\mathbf{C}_t = \mathbf{x}_t^\top$ ,  $\mathbf{A}_t = \mathbf{I}$ ,  $\mathbf{Q}_t = 0$  and  $\mathbf{R}_t = \sigma^2$ . Thus  $\Sigma_{t|t-1} = \Sigma_{t-1}$ , and the  
 37 remaining equations simplify as follows:

$$39 \quad s_t = \mathbf{x}_t^\top \Sigma_{t-1} \mathbf{x}_t + \sigma^2 \quad (8.160)$$

$$41 \quad k_t = \frac{1}{s_t} \Sigma_{t-1} \mathbf{x}_t \quad (8.161)$$

$$43 \quad \boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + k_t (y_t - \mathbf{x}_t^\top \boldsymbol{\mu}_{t-1}) = \boldsymbol{\mu}_{t-1} + \frac{1}{s_t} \Sigma_{t-1} \mathbf{x}_t (y_t - \mathbf{x}_t^\top \boldsymbol{\mu}_{t-1}) \quad (8.162)$$

$$45 \quad \Sigma_t = (\mathbf{I} - k_t \mathbf{x}_t^\top) \Sigma_{t-1} = \Sigma_{t-1} - \frac{1}{s_t} \Sigma_{t-1} \mathbf{x}_t \mathbf{x}_t^\top \Sigma_{t-1} \quad (8.163)$$

Note that from Equation (8.99), we can also write the Kalman gain as

$$\mathbf{k}_t = \frac{1}{\sigma^2} (\boldsymbol{\Sigma}_{t-1}^{-1} + \frac{1}{\sigma^2} \mathbf{x}_t \mathbf{x}_t^\top)^{-1} \mathbf{x}_t \quad (8.164)$$

Also, from Equation (8.97), we can also write the posterior covariance as

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t-1} - s_t \mathbf{k}_t \mathbf{k}_t^\top \quad (8.165)$$

If we let  $\mathbf{V}_t = \boldsymbol{\Sigma}_t / \sigma^2$ , we can further simplify the equations, as follows [Bor16].

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + \frac{\sigma^2 \mathbf{V}_{t-1} \mathbf{x}_t (y_t - \mathbf{x}_t^\top \boldsymbol{\mu}_{t-1})}{\sigma^2 (\mathbf{x}_t^\top \mathbf{V}_{t-1} \mathbf{x}_t + 1)} = \boldsymbol{\mu}_{t-1} + \frac{\mathbf{V}_{t-1} \mathbf{x}_t (y_t - \mathbf{x}_t^\top \boldsymbol{\mu}_{t-1})}{\mathbf{x}_t^\top \mathbf{V}_{t-1} \mathbf{x}_t + 1} \quad (8.166)$$

$$\mathbf{V}_t = \mathbf{V}_{t-1} - \frac{\mathbf{V}_{t-1} \mathbf{x}_t \mathbf{x}_t^\top \mathbf{V}_{t-1}}{\mathbf{x}_t^\top \mathbf{V}_{t-1} \mathbf{x}_t + 1} \quad (8.167)$$

We can initialize these recursions using a vague prior,  $\boldsymbol{\mu}_0 = \mathbf{0}$ ,  $\boldsymbol{\Sigma}_0 = \infty \mathbf{I}$ . In this case, the posterior mean will converge to the MLE, and the posterior standard deviations will converge to the standard error of the mean.

The quantity  $\mathbf{K}_t$  is called the **Kalman gain matrix**, and determines how much we should trust the current observation relative to our current prior. If we make the approximation  $\mathbf{K}_t \approx \eta_t \mathbf{1}$ , we recover the **least mean squares** or **LMS** algorithm, where  $\eta_t$  is the learning rate. In LMS, we need to adapt the learning rate to ensure convergence to the MLE. Furthermore, the algorithm may require multiple passes through the data to converge to this global optimum. By contrast, the RLS algorithm automatically performs step-size adaptation, and converges to the optimal posterior in a single pass over the data. See Figure 8.9b for an example.

### 8.4.3 Predictive coding as Kalman filtering

In the field of neuroscience, a popular theoretical model for how the brain works is known as **predictive coding** (see e.g., [Rao99; Fri03; Spr17; MSB21; Mar21]). This posits that the core function of the brain is simply to minimize prediction error at each layer of a hierarchical model, and at each moment in time (c.f., Figure 28.1). There is considerable biological evidence for this (see above references). Furthermore, it turns out that the predictive coding algorithm, when applied to a linear Gaussian state-space model, is equivalent to the Kalman filter, as shown in [Mil21, Sec 3.4.2].

To see this, we adopt the framework of inference as optimization, as used in variational inference (Chapter 10). The joint distribution is given by  $p(\mathbf{y}_{1:T}, \mathbf{z}_{1:T}) = p(\mathbf{y}_1 | \mathbf{z}_1) p(\mathbf{z}_1) \prod_{t=2}^T p(\mathbf{y}_t | \mathbf{z}_t) p(\mathbf{z}_t | \mathbf{z}_{t-1})$ . Our goal is to approximate the filtering distribution,  $p(\mathbf{z}_t | \mathbf{y}_{1:t})$ . We will use a fully factorized approximation of the form  $q(\mathbf{z}_{1:T}) = \prod_{t=1}^T q(\mathbf{z}_t)$ . Following Section 10.1.1, the variational free energy is given by  $\mathcal{F}(\boldsymbol{\psi}) = \sum_{t=1}^T \mathcal{F}_t(\boldsymbol{\psi}_t)$ , where

$$\mathcal{F}_t(\boldsymbol{\psi}_t) = \mathbb{E}_{q(\mathbf{z}_{t-1})} [D_{\text{KL}}(q(\mathbf{z}_t) \| p(\mathbf{z}_t, \mathbf{y}_t | \mathbf{z}_{t-1}))] \quad (8.168)$$

We will use a Gaussian approximation for  $q$  at each step. Furthermore, we will use the Laplace approximation, which derives the covariance from the Hessian at the mean. Thus we have  $q(\mathbf{z}_t) = \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_t, \boldsymbol{\Sigma}(\boldsymbol{\mu}_t))$ , where  $\boldsymbol{\psi}_t = \boldsymbol{\mu}_t$  is the variational parameter which we need to compute. (Once we have computed  $\boldsymbol{\mu}_t$ , we can derive  $\boldsymbol{\Sigma}$ .)

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,  $\boldsymbol{\epsilon}_y = \mathbf{y}_t - \mathbf{C}\boldsymbol{\mu}_t$ :

$$\nabla \mathcal{F}_t(\boldsymbol{\mu}_t) = 2\mathbf{C}^\top \Sigma_y^{-1} \mathbf{y}_t - (\mathbf{C}^\top \Sigma_y^{-1} \mathbf{C} + \mathbf{C}^\top \Sigma_y^{-\top} \mathbf{C})\boldsymbol{\mu}_t \quad (8.178)$$

$$+ (\Sigma_z^{-1} + \Sigma_z - \mathbf{T})\boldsymbol{\mu}_t - 2\Sigma_z^{-1} \mathbf{A}\boldsymbol{\mu}_{t-1} - 2\Sigma_z^{-1} \mathbf{B}\mathbf{u}_{t-1} \quad (8.179)$$

$$= 2\mathbf{C}^\top \Sigma_y^{-1} \mathbf{C}\boldsymbol{\mu}_t - 2\mathbf{C}^\top \Sigma_y^{-1} \mathbf{C}\boldsymbol{\mu}_t + 2\Sigma_z^{-1} \boldsymbol{\mu}_t - 2\Sigma_z^{-1} \mathbf{A}\boldsymbol{\mu}_{t-1} - 2\Sigma_z^{-1} \mathbf{B}\mathbf{u}_{t-1} \quad (8.180)$$

$$= -\mathbf{C}^\top \Sigma_y^{-1} [\mathbf{y}_t - \mathbf{C}\boldsymbol{\mu}_t] + \Sigma_z^{-1} [\boldsymbol{\mu}_t - \mathbf{A}\boldsymbol{\mu}_{t-1} - \mathbf{B}\mathbf{u}_{t-1}] \quad (8.181)$$

$$= -\mathbf{C}^\top \Sigma_y^{-1} \boldsymbol{\epsilon}_y + \Sigma_z^{-1} \boldsymbol{\epsilon}_z \quad (8.182)$$

Thus minimizing (precision weighted) prediction errors is equivalent to minimizing the VFE.<sup>6</sup> In this case the objective is convex, so we can find the global optimum. Furthermore, the resulting Gaussian posterior is exact for this model class, and thus predictive coding gives the same results as Kalman filtering. However, the advantage of predictive coding is that it is easy to extend to hierarchical and nonlinear models: we just have to minimize the VFE using gradient descent (see e.g., [HM20]).

6. Scaling the error terms by the inverse variance can be seen as a form of normalization. To see this, consider the standardization operator:  $\text{standardize}(x) = (x - \mathbb{E}[x])/\sqrt{\mathbb{V}[x]}$ . It has been argued that that the widespread presence of neural circuitry for performing normalization, together with the upwards and downwards connections between brain regions, adds support for the claim that the brain implements predictive coding (see e.g., [RB99; Fri03; Spr17; MSB21; Mar21]).

Furthermore, we can also optimize the VFE with respect to the model parameters, as in variational EM. In the case of linear Gaussian state-space models, [Mil21, Sec 3.4.2] show that for the dynamics matrix the gradient is  $\nabla_{\mathbf{A}} \mathcal{F}_t = -\boldsymbol{\Sigma}_z \boldsymbol{\epsilon}_y \boldsymbol{\mu}_{t-1}^\top$ , for the control matrix the gradient is  $\nabla_{\mathbf{B}} \mathcal{F}_t = -\boldsymbol{\Sigma}_z \boldsymbol{\epsilon}_y \mathbf{u}_{t-1}^\top$ , and for the observation matrix the gradient is  $\nabla_{\mathbf{C}} \mathcal{F}_t = -\boldsymbol{\Sigma}_y \boldsymbol{\epsilon}_y \boldsymbol{\mu}_t^\top$ . These expressions can be generalized to nonlinear models. Indeed, predictive coding can in fact approximate backpropagation for many kinds of model [MTB20].

Gradient descent using these predicting coding takes the form of a **Hebbian update rule**, in which we set the new parameter to the old one plus a term that is a multiplication of the two quantities available at each end of the synaptic connection, namely the prediction error  $\epsilon$  as input, and the value  $\mu$  (or  $\theta$ ) of the neuron as output. However, there are still several aspects of this model that are biologically implausible, such as assuming symmetric weights (since both  $\mathbf{C}$  and  $\mathbf{C}^\top$  are needed, the former to compute  $\boldsymbol{\epsilon}_y$  and the latter to compute  $\nabla_{\boldsymbol{\mu}_t} \mathcal{F}_t$ ), the need for one-to-one alignment of error signals and parameter values, and the need (in the nonlinear case) for computing the derivative of the activation function. In [Mil+21] they develop an approximate, more biologically plausible version of predictive coding that relaxes these requirements, and which does not seem to hurt empirical performance too much.

18

19

#### 20 8.4.4 The Kalman (RTS) smoother

21 In Section 8.4.1, we described the Kalman filter, which sequentially computes  $p(\mathbf{z}_t | \mathbf{y}_{1:t})$  for each  $t$ .  
 22 This is useful for online inference problems, such as tracking. However, in an offline setting, we can  
 23 wait until all the data has arrived, and then compute  $p(\mathbf{z}_t | \mathbf{y}_{1:T})$ . By conditioning on past and future  
 24 data, our uncertainty will be significantly reduced. This is illustrated in Figure 8.5(c), where we see  
 25 that the posterior covariance ellipsoids are smaller for the smoothed trajectory than for the filtered  
 26 trajectory.  
 27

28 We now explain how to compute the smoothed estimates, using an algorithm called the **RTS**  
 29 **smoother**, named after its inventors, Rauch, Tung and Striebel [RTS65]. It is also known as the  
 30 **Kalman smoothing** algorithm. The algorithm is the linear-Gaussian analog to the the forwards-  
 31 filtering backwards-smoothing algorithm for HMMs.  
 32

##### 33 8.4.4.1 Algorithm

34 One can show (see below) that the backwards smoothing pass has the following recursive form:  
 35

$$36 p(\mathbf{z}_t | \mathbf{y}_{1:T}) = \mathcal{N}(\boldsymbol{\mu}_{t|T}, \boldsymbol{\Sigma}_{t|T}) \quad (8.183)$$

$$37 \boldsymbol{\mu}_{t|T} = \boldsymbol{\mu}_{t|t} + \mathbf{G}_t (\boldsymbol{\mu}_{t+1|T} - \boldsymbol{\mu}_{t+1|t}) \quad (8.184)$$

$$38 \boldsymbol{\Sigma}_{t|T} = \boldsymbol{\Sigma}_{t|t} + \mathbf{G}_t (\boldsymbol{\Sigma}_{t+1|T} - \boldsymbol{\Sigma}_{t+1|t}) \mathbf{G}_t^\top \quad (8.185)$$

$$39 \mathbf{G}_t = \boldsymbol{\Sigma}_{t|t} \mathbf{A}_{t+1}^\top \boldsymbol{\Sigma}_{t+1|t}^{-1} \quad (8.186)$$

40 where  $\mathbf{G}_t$  is the backwards Kalman gain matrix, and  $\boldsymbol{\mu}_{t|t} = \boldsymbol{\mu}_t$  and  $\boldsymbol{\Sigma}_{t|t} = \boldsymbol{\Sigma}_t$  are the outputs from  
 41 the filtering step. The algorithm can be initialized from  $\boldsymbol{\mu}_{T|T}$  and  $\boldsymbol{\Sigma}_{T|T}$  from the Kalman filter. See  
 42 e.g., [Sar13, p136] for the derivation (where they use the notation  $\boldsymbol{\mu}_t^s$  and  $\boldsymbol{\Sigma}_t^s$  for  $\boldsymbol{\mu}_{t|T}$  and  $\boldsymbol{\Sigma}_{t|T}$ ).  
 43

47

---

#### 8.4.4.2 Two-filter smoothing

Note that the backwards pass of the Kalman smoother does not need access to the observations,  $\mathbf{y}_{1:T}$ , but does need access to the filtered belief states from the forwards pass,  $p(\mathbf{z}_t|\mathbf{y}_{1:t}) = \mathcal{N}(\mathbf{z}_t|\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t})$ . There is an alternative version of the algorithm, known as **two-filter smoothing** [FP69; Kit04], in which we compute the forwards pass as usual, and then separately compute backwards messages  $p(\mathbf{y}_{t+1:T}|\mathbf{z}_t) \propto \mathcal{N}(\mathbf{z}_t|\boldsymbol{\mu}_t^b, \boldsymbol{\Sigma}_t^b)$ , similar to the backwards filtering algorithm in HMMs (Section 8.3.3).

However, these backwards messages are not posteriors, but instead are conditional likelihoods. This causes numerical problems. For example, consider  $t = T$ ; in this case, we need to set the initial covariance matrix to be  $\boldsymbol{\Sigma}_T^b = \infty \mathbf{I}$ , so the the backwards message has no effect on the filtered posterior (since there is no evidence beyond step  $T$ ). This problem can be resolved by working in information form. Unfortunately this does not work in the non-linear and/or non-Gaussian case.

An alternative approach is to generalize the two-filter smoothing equations to ensure the likelihoods are normalizable by multiplying them by artificial distributions  $\{\gamma_t(\mathbf{z}_t)\}$ . See [BDM10] for details.

#### 8.4.4.3 Time and space complexity

In general, the Kalman smoothing algorithm takes  $O(N_y N_z^2 T)$  time where  $\mathbf{z}_t \in \mathbb{R}^{N_z}$  is the hidden state, and  $\mathbf{y}_t \in \mathbb{R}^{N_y}$  is the observation. This can be slow when applied to long sequences. In [SGF21], they describe how to reduce the linear dependence on  $T$  to  $\log(T)$  time using a **parallel prefix scan** operator, that can be run efficiently on GPUs. In addition, we can reduce the space from  $O(N_z^2 T)$ ,  $O(N_z^2 \log T)$  using the same island algorithm as in Section 8.3.5.

### 8.5 Inference based on local linearization

In this section, we extend the Kalman filter and smoother to the case where the system dynamics and/or the observation model are nonlinear. However, we continue to assume Gaussian noise distributions. Thus we assume the model has the following form:

$$\mathbf{z}_t = \mathbf{f}_t(\mathbf{z}_{t-1}) + \mathcal{N}(\mathbf{0}, \mathbf{Q}_t) \quad (8.187)$$

$$\mathbf{y}_t = \mathbf{h}_t(\mathbf{z}_t) + \mathcal{N}(\mathbf{0}, \mathbf{R}_t) \quad (8.188)$$

where  $\mathbf{z}_t \in \mathbb{R}^{N_z}$  is the hidden state,  $\mathbf{y}_t \in \mathbb{R}^{N_y}$  is the observation,  $\mathbf{f}_t : \mathbb{R}^{N_z} \rightarrow \mathbb{R}^{N_z}$  is the dynamics model, and  $\mathbf{h}_t : \mathbb{R}^{N_z} \rightarrow \mathbb{R}^{N_y}$  is the observation model.

Exact posterior inference in this model family is generally computationally intractable, so in this section, we create a locally linear approximation to the model, following the presentation of [Sar13, Ch. 5]. For other reviews of algorithms for approximate inference in nonlinear state space models, see e.g., [Fan+17; Li+17d; Koy+10].

#### 8.5.1 Taylor series expansion

Suppose  $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$  and  $\mathbf{y} = \mathbf{f}(\mathbf{z})$ , where  $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$  is a differentiable and invertible function. The pdf for  $\mathbf{y}$  is given by

$$p(\mathbf{y}) = |\det \mathbf{J}(\mathbf{y})| \mathcal{N}(\mathbf{f}^{-1}(\mathbf{y})|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \quad (8.189)$$

1 where  $\mathbf{J}$  is the Jacobian of  $\mathbf{f}^{-1}$  evaluated at  $\mathbf{y}$ :

2

$$[\mathbf{J}]_{jj'} = \frac{\partial f_j^{-1}(\mathbf{u})}{\partial u_{j'}}|_{\mathbf{u}=\mathbf{y}} \quad (8.190)$$

3

4 In general this is intractable to compute, so we seek an approximation.

5 Suppose  $\mathbf{z} = \boldsymbol{\mu} + \delta\mathbf{z}$ , where  $\delta\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ . Then we can form a Taylor series expansion of the

6 function  $\mathbf{f}$  as follows:

7

$$\mathbf{f}(\mathbf{z}) = \mathbf{f}(\boldsymbol{\mu} + \delta\mathbf{z}) \approx \mathbf{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\delta\mathbf{z} + \sum_i \frac{1}{2} \delta\mathbf{z}^\top \mathbf{F}^i(\boldsymbol{\mu}) \delta\mathbf{z} \mathbf{e}_i + \dots \quad (8.191)$$

8

9 where  $\mathbf{e}_i = (0, \dots, 1, 0, \dots, 0)$  is the  $i$ 'th unit vector,  $\mathbf{F}(\boldsymbol{\mu})$  is the Jacobian of  $\mathbf{f}$ ,

10

$$[\mathbf{F}(\boldsymbol{\mu})]_{jj'} = \frac{\partial f_j(\mathbf{z})}{\partial x_{j'}}|_{\mathbf{z}=\boldsymbol{\mu}} \quad (8.192)$$

11

12 and  $\mathbf{F}^i(\boldsymbol{\mu})$  is the Hessian of  $f_i(\cdot)$  computed at  $\boldsymbol{\mu}$ :

13

$$[\mathbf{F}^i(\boldsymbol{\mu})]_{jj'} = \frac{\partial^2 f_i(\mathbf{z})}{\partial x_j \partial x_{j'}}|_{\mathbf{z}=\boldsymbol{\mu}} \quad (8.193)$$

14

15 We can create a linear approximation by just using the first two terms:

16

$$\hat{\mathbf{f}}(\mathbf{z}) = \mathbf{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\delta\mathbf{z} \quad (8.194)$$

17

18 We now derive the induced Gaussian approximation to  $\mathbf{y} = \mathbf{f}(\mathbf{z})$ . The mean is given by

19

$$\mathbb{E}[\mathbf{y}] \approx \mathbb{E}[\mathbf{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\delta\mathbf{z}] \quad (8.195)$$

20

$$= \mathbf{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\mathbb{E}[\delta\mathbf{z}] \quad (8.196)$$

21

$$= \mathbf{f}(\boldsymbol{\mu}) \quad (8.197)$$

22

23 The covariance is given by

24

$$\text{Cov}[\mathbf{y}] = \mathbb{E}[(\mathbf{f}(\mathbf{z}) - \mathbb{E}[\mathbf{f}(\mathbf{z})])(\mathbf{f}(\mathbf{z}) - \mathbb{E}[\mathbf{f}(\mathbf{z})])^\top] \quad (8.198)$$

25

$$\approx \mathbb{E}[(\mathbf{f}(\mathbf{z}) - \mathbf{f}(\boldsymbol{\mu}))(\mathbf{f}(\mathbf{z}) - \mathbf{f}(\boldsymbol{\mu}))^\top] \quad (8.199)$$

26

$$\approx \mathbb{E}[(\mathbf{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\delta\mathbf{z} - \mathbf{f}(\boldsymbol{\mu}))(\mathbf{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\delta\mathbf{z} - \mathbf{f}(\boldsymbol{\mu}))^\top] \quad (8.200)$$

27

$$= \mathbb{E}[(\mathbf{F}(\boldsymbol{\mu})\delta\mathbf{z})(\mathbf{F}(\boldsymbol{\mu})\delta\mathbf{z})^\top] \quad (8.201)$$

28

$$= \mathbf{F}(\boldsymbol{\mu})\mathbb{E}[\delta\mathbf{z}\delta\mathbf{z}^\top]\mathbf{F}(\boldsymbol{\mu})^\top \quad (8.202)$$

29

$$= \mathbf{F}(\boldsymbol{\mu})\boldsymbol{\Sigma}\mathbf{F}(\boldsymbol{\mu})^\top \quad (8.203)$$

30

31 Let us consider a 1d example. In Figure 8.10a, we show what happens when we pass a Gaussian distribution  $p(x)$ , shown on the bottom right, through a nonlinear function  $y = f(x)$ , shown on the top right. The resulting distribution (approximated by Monte Carlo) is shown in the shaded gray area in the top left corner. The best Gaussian approximation to this, computed from  $\mathbb{E}[f(x)]$  and

32

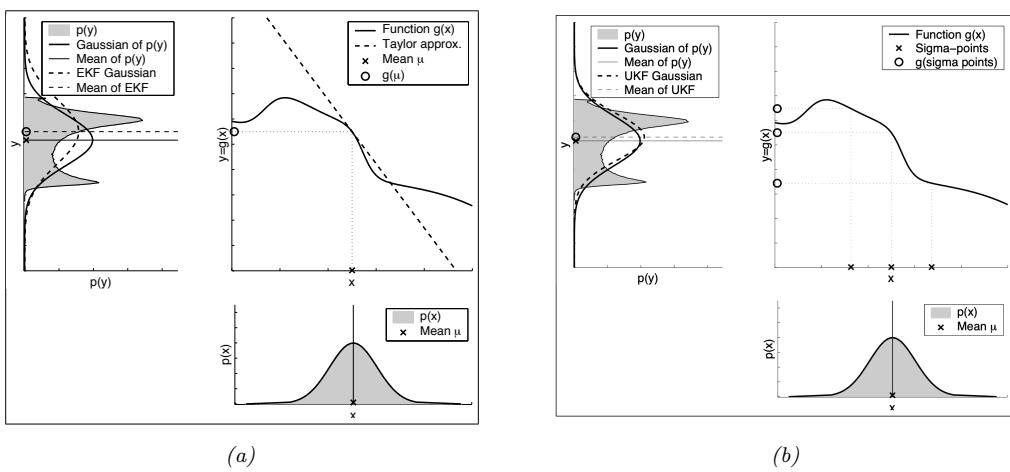


Figure 8.10: Nonlinear transformation of a Gaussian random variable. The prior  $p(x)$  is shown on the bottom right. The function  $y = f(x)$  is shown on the top right. The transformed distribution  $p(y)$  is shown in the top left. A linear function induces a Gaussian distribution, but a non-linear 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.

$\nabla [f(x)]$  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,  $\mu$ , and then passes the Gaussian distribution  $p(x)$  through this linearized function. In this example, the result is quite a good approximation to the first and second moments of  $p(y)$ , for much less cost than an MC approximation.

We often also need to compute a Gaussian approximation to the joint distribution  $p(z, y)$ . We can compute this in the same way, by defining the augmented function  $\tilde{f}(z) = [z, f(z)]$ . This gives

$$\mathbb{E} [\tilde{f}(z)] \approx \begin{pmatrix} \mu \\ f(\mu) \end{pmatrix} \quad (8.204)$$

$$\text{Cov} [\tilde{f}(z)] \approx \begin{pmatrix} \mathbf{I} \\ \mathbf{F}(\mu) \end{pmatrix} \Sigma \begin{pmatrix} \mathbf{I} \\ \mathbf{F}(\mu) \end{pmatrix}^\top = \begin{pmatrix} \Sigma & \Sigma \mathbf{F}(\mu) \\ \mathbf{F}(\mu) \Sigma & \mathbf{F}(\mu) \Sigma \mathbf{F}(\mu)^\top \end{pmatrix} \quad (8.205)$$

When deriving the EKF, we need the following slightly more general version:

$$z \sim \mathcal{N}(\mu, \Sigma), y = f(z) + q, q \sim \mathcal{N}(\mathbf{0}, Q) \quad (8.206)$$

The resulting linear approximation to the joint is

$$\begin{pmatrix} z \\ y \end{pmatrix} \sim \mathcal{N} \left( \begin{pmatrix} \mu \\ \mu_L \end{pmatrix}, \begin{pmatrix} \Sigma & \mathbf{C}_L \\ \mathbf{C}_L^\top & \mathbf{S}_L \end{pmatrix} \right) \quad (8.207)$$

$$\mu_L = f(\mu) \quad (8.208)$$

$$\mathbf{S}_L = \mathbf{F}(\mu) \Sigma \mathbf{F}(\mu)^\top + \mathbf{Q} \quad (8.209)$$

$$\mathbf{C}_L = \Sigma \mathbf{F}(\mu)^\top \quad (8.210)$$

<sup>1</sup> It is also possible to derive an approximation for the case of non-additive Gaussian noise, where  
<sup>2</sup>  $\mathbf{y} = \mathbf{f}(\mathbf{z}, \mathbf{q})$ . See [Sar13, Sec 5.1] for details.  
<sup>3</sup>

### <sup>4</sup> 8.5.2 The extended Kalman filter (EKF)

<sup>5</sup> In this section, we discuss the **extended Kalman filter** or **EKF**, which can compute a Gaussian  
<sup>6</sup> approximation to the posterior even when the model has nonlinear dynamics and/or observations.  
<sup>7</sup> The basic idea is to linearize the dynamics and observation models about the previous state estimate  
<sup>8</sup> using a first order Taylor series expansion, as in Section 8.5.1, and then to apply the standard Kalman  
<sup>9</sup> filter equations from Section 8.4.1. Thus we approximate a stationary non-linear dynamical system  
<sup>10</sup> with a non-stationary linear dynamical system. (However, the noise variance terms,  $\mathbf{Q}$  and  $\mathbf{R}$ , are  
<sup>11</sup> not changed, i.e., the additional error due to linearization is not modeled.)  
<sup>12</sup>

#### <sup>13</sup> 8.5.2.1 Algorithm

<sup>14</sup> The EKF linearizes the model at each step by computing the following Jacobian matrices:  
<sup>15</sup>

$$\mathbf{F}_t = \frac{\partial \mathbf{f}_t(\mathbf{z})}{\partial \mathbf{z}}|_{\boldsymbol{\mu}_{t-1}} \quad (8.211)$$

$$\mathbf{H}_t = \frac{\partial \mathbf{h}_t(\mathbf{z})}{\partial \mathbf{z}}|_{\boldsymbol{\mu}_{t|t-1}} \quad (8.212)$$

<sup>22</sup> The updates then become

$$\boldsymbol{\mu}_{t|t-1} = \mathbf{f}(\boldsymbol{\mu}_{t-1}) \quad (8.213)$$

$$\boldsymbol{\Sigma}_{t|t-1} = \mathbf{F}_t \boldsymbol{\Sigma}_{t-1} \mathbf{F}_t + \mathbf{Q}_t \quad (8.214)$$

$$\mathbf{e}_t = \mathbf{y}_t - \mathbf{h}(\boldsymbol{\mu}_{t|t-1}) \quad (8.215)$$

$$\mathbf{S}_t = \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top + \mathbf{R}_t \quad (8.216)$$

$$\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t \mathbf{S}_t^{-1} \quad (8.217)$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t \mathbf{e}_t \quad (8.218)$$

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} = \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t \mathbf{S}_t \mathbf{K}_t^\top \quad (8.219)$$

#### <sup>34</sup> 8.5.2.2 Derivation

<sup>35</sup> The derivation of the EKF is similar to the derivation of the Kalman filter (Section 8.4.1.4), except  
<sup>36</sup> we also need to apply the linear approximation from Section 8.5.1.

<sup>37</sup> 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  
<sup>38</sup>

$$p(\mathbf{z}_{t-1}, \mathbf{z}_t | \mathbf{y}_{1:t-1}) \approx \mathcal{N}\left(\begin{pmatrix} \mathbf{z}_{t-1} \\ \mathbf{z}_t \end{pmatrix} | \mathbf{m}', \boldsymbol{\Sigma}'\right) \quad (8.220)$$

$$\mathbf{m}' = \begin{pmatrix} \boldsymbol{\mu}_{t-1} \\ \mathbf{f}(\boldsymbol{\mu}_{t-1}) \end{pmatrix} \quad (8.221)$$

$$\boldsymbol{\Sigma}' = \begin{pmatrix} \boldsymbol{\Sigma}_{t-1} & \boldsymbol{\Sigma}_{t-1} \mathbf{F}_t^\top \\ \mathbf{F}_t \boldsymbol{\Sigma}_{t-1} & \mathbf{F}_t \boldsymbol{\Sigma}_{t-1} \mathbf{F}_t^\top + \mathbf{Q}_{t-1} \end{pmatrix} \quad (8.222)$$

where  $\mathbf{F}_t$  is the Jacobian of  $\mathbf{f}$  evaluated at  $\boldsymbol{\mu}_{t-1}$ . From this we can derive the marginal  $p(\mathbf{z}_t | \mathbf{y}_{1:t-1})$ , which gives us the predict step.

For the update step, we first consider a Gaussian approximation to  $p(\mathbf{z}_t, \mathbf{y}_t | \mathbf{y}_{1:t-1})$  as follows:

$$p(\mathbf{z}_t, \mathbf{y}_t | \mathbf{y}_{1:t-1}) \approx \mathcal{N}(\begin{pmatrix} \mathbf{z}_t \\ \mathbf{y}_t \end{pmatrix} | \mathbf{m}'', \Sigma'') \quad (8.223)$$

$$\mathbf{m}'' = \begin{pmatrix} \boldsymbol{\mu}_{t|t-1} \\ \mathbf{h}(\boldsymbol{\mu}_{t|t-1}) \end{pmatrix} \quad (8.224)$$

$$\Sigma'' = \begin{pmatrix} \boldsymbol{\Sigma}_{t|t-1} & \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top \\ \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} & \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top + \mathbf{R}_{t-1} \end{pmatrix} \quad (8.225)$$

where  $\mathbf{H}_t$  is the Jacobian of  $\mathbf{h}$  evaluated at  $\boldsymbol{\mu}_{t|t-1}$ .

Finally, we use Equation (2.50) to get the posterior

$$p(\mathbf{z}_t | \mathbf{y}_t, \mathbf{y}_{1:t-1}) \approx \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t) \quad (8.226)$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top (\mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top + \mathbf{R}_t)^{-1} [\mathbf{y}_t - \mathbf{h}(\boldsymbol{\mu}_{t|t-1})] \quad (8.227)$$

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t|t-1} - \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top (\mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top + \mathbf{R}_t)^{-1} \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \quad (8.228)$$

This gives us the update step.

See Section 31.3.1 for an example.

### 8.5.2.3 Accuracy

There are two cases when the EKF works poorly. The first is when the prior covariance is large. In this case, the prior distribution is broad, so we end up sending a lot of probability mass through different parts of the function that are far from  $\boldsymbol{\mu}_{t-1|t-1}$ , where the function has been linearized. See Figure 8.11 for an illustration. The other setting where the EKF works poorly is when the function is highly nonlinear near the current mean. See Figure 8.12 for an illustration.

In Section 8.6.2, we will discuss an algorithm called the UKF which works better than the EKF in both of these settings. An alternative approach is to use a second-order Taylor series approximation. The resulting updates can still be computed in closed form (see [Sar13, Sec 5.2] for details). We can further improve performance by repeatedly re-linearizing the equations around  $\boldsymbol{\mu}_t$  instead of  $\boldsymbol{\mu}_{t|t-1}$ ; this is called the **iterated EKF**.

### 8.5.2.4 Diagonal approximation

The cost of the EKF is  $O(N_y N_z^2)$ , which can be prohibitive for large state spaces. In such cases, a natural approximation is to use a block diagonal approximation. Let us define the following Jacobian matrices for block  $i$ :

$$\mathbf{F}_t^i = \frac{\partial \mathbf{f}_t^i(\mathbf{z})}{\partial \mathbf{z}}|_{\boldsymbol{\mu}_{t-1}} \quad (8.229)$$

$$\mathbf{H}_t^i = \frac{\partial \mathbf{h}_t^i(\mathbf{z})}{\partial \mathbf{z}}|_{\boldsymbol{\mu}_{t|t-1}} \quad (8.230)$$

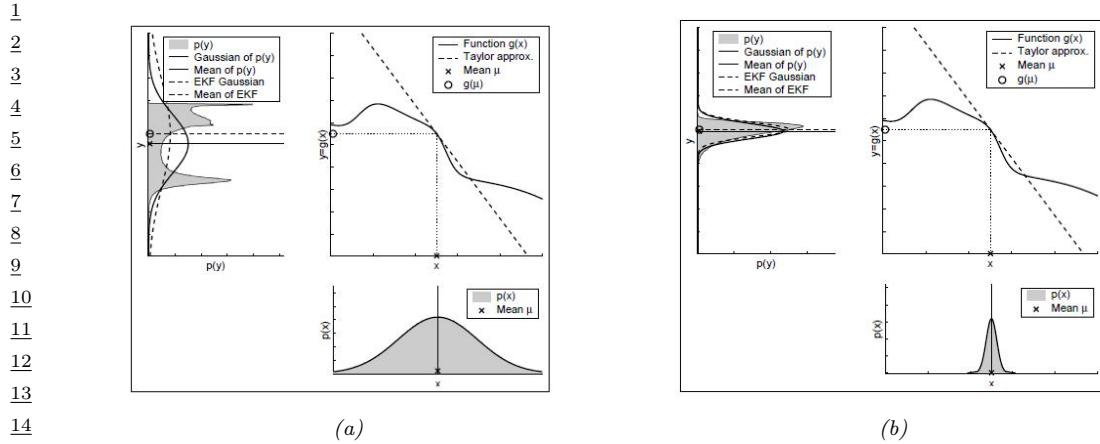


Figure 8.11: Illustration of the fact that a broad prior (a) may result in a more complex posterior than a narrow prior (b). Consequently, the EKF approximation may work poorly in situations of high uncertainty. From Figure 3.5 of [TBF06]. Used with kind permission of Dieter Fox.

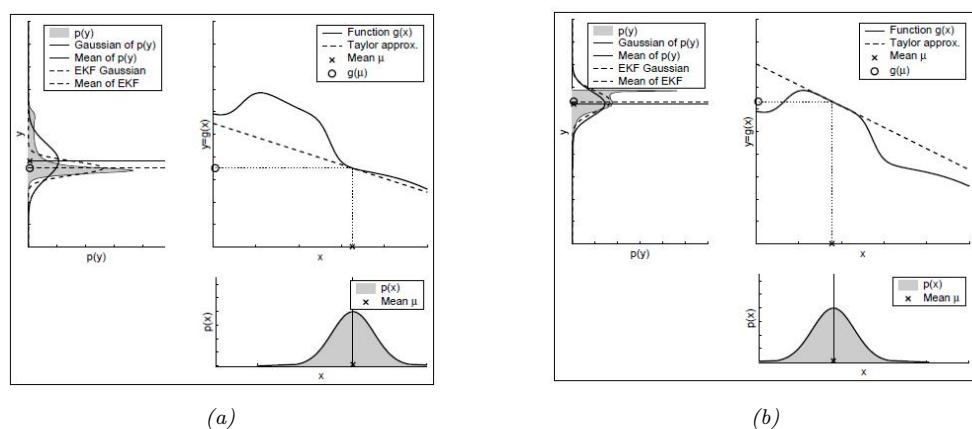


Figure 8.12: Illustration of the fact that if the function function is very nonlinear (a) at the current operating point, the posterior will be less well approximated by the EKF than if the function is locally linear (b). From Figure 3.6 of [TBF06]. Used with kind permission of Dieter Fox.

We then compute the following updates for each block:

$$\boldsymbol{\mu}_{t|t-1}^i = \mathbf{f}^i(\boldsymbol{\mu}_{t-1}) \quad (8.231)$$

$$\boldsymbol{\Sigma}_{t|t-1}^i = (\mathbf{F}_t^i)^\top \boldsymbol{\Sigma}_{t-1}^i \mathbf{F}_t^i + \mathbf{Q}_t^i \quad (8.232)$$

$$\mathbf{S}_t = \sum_i (\mathbf{H}_t^i)^\top \boldsymbol{\Sigma}_{t|t-1}^i \mathbf{H}_t^i + \mathbf{R}_t \quad (8.233)$$

$$\mathbf{K}_t^i = \boldsymbol{\Sigma}_{t|t-1}^i \mathbf{H}_t^i \mathbf{S}_t^{-1} \quad (8.234)$$

$$\boldsymbol{\mu}_t^i = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t^i \mathbf{e}_t \quad (8.235)$$

$$\boldsymbol{\Sigma}_t^i = \boldsymbol{\Sigma}_{t|t-1}^i - \mathbf{K}_t^i \mathbf{H}_t^i \boldsymbol{\Sigma}_{t|t-1}^i \quad (8.236)$$

### 8.5.3 The extended Kalman smoother

We can extend the EKF to the offline smoothing case as follows (see e.g., [Sar13, Sec 9.1] for the derivation):

$$p(\mathbf{z}_t | \mathbf{y}_{1:T}) = \mathcal{N}(\boldsymbol{\mu}_{t|T}, \boldsymbol{\Sigma}_{t|T}) \quad (8.237)$$

$$\boldsymbol{\mu}_{t|T} = \boldsymbol{\mu}_{t|t} + \mathbf{G}_t (\boldsymbol{\mu}_{t+1|T} - \boldsymbol{\mu}_{t+1|t}) \quad (8.238)$$

$$\boldsymbol{\Sigma}_{t|T} = \boldsymbol{\Sigma}_{t|t} + \mathbf{G}_t (\boldsymbol{\Sigma}_{t+1|T} - \boldsymbol{\Sigma}_{t+1|t}) \mathbf{G}_t^\top \quad (8.239)$$

$$\mathbf{G}_t = \boldsymbol{\Sigma}_{t|t} \mathbf{F}(\boldsymbol{\mu}_t)^\top \boldsymbol{\Sigma}_{t+1|t}^{-1} \quad (8.240)$$

The only difference from the RTS smoother in Section 8.4.4 is that we replace the fixed  $\mathbf{F}$  matrix with the Jacobian  $\mathbf{F}(\boldsymbol{\mu}_t)$ .

### 8.5.4 Exponential-family EKF

In this section, we present an extension of the EKF to the case where the observation model is in the exponential family, as proposed in [Oll18]. We call this the **Exponential family EKF** or **EEKF**. This allows us to apply the EKF for online parameter estimation of classification models, as we illustrate in Section 8.5.4.3.

#### 8.5.4.1 Modeling assumptions

We assume the dynamics model is the usual nonlinear model plus Gaussian noise, with optional inputs  $\mathbf{u}_t$ :

$$\mathbf{z}_t = f(\mathbf{z}_{t-1}, \mathbf{u}_t) + \mathcal{N}(\mathbf{0}, \mathbf{Q}_t) \quad (8.241)$$

We assume the observation model is

$$p(\mathbf{y}_t | \mathbf{z}_t) = \text{Expfam}(\mathbf{y}_t | \hat{\mathbf{y}}_t) \quad (8.242)$$

where the mean (moment) parameter of the exponential family is computed deterministically using a nonlinear observation model:

$$\hat{\mathbf{y}}_t = h(\mathbf{z}_t, \mathbf{u}_t) \quad (8.243)$$

The standard EKF corresponds to the special case of a Gaussian output with fixed observation covariance  $\mathbf{R}_t$ , with  $\hat{\mathbf{y}}_t$  being the mean.

#### 8.5.4.2 Algorithm

The EEKF algorithm is as follows. First, the prediction step:

$$\boldsymbol{\mu}_{t|t-1} = f(\boldsymbol{\mu}_{t-1}, \mathbf{u}_t) \quad (8.244)$$

$$\mathbf{F}_t = \frac{\partial f}{\partial \mathbf{z}}|_{(\boldsymbol{\mu}_{t-1}, \mathbf{u}_t)} \quad (8.245)$$

$$\boldsymbol{\Sigma}_{t|t-1} = \mathbf{F}_t \boldsymbol{\Sigma}_{t-1} \mathbf{F}_t^\top + \mathbf{Q}_t \quad (8.246)$$

$$\hat{\mathbf{y}}_t = h(\boldsymbol{\mu}_{t|t-1}, \mathbf{u}_t) \quad (8.247)$$

1      Second, after seeing observation  $\mathbf{y}_t$ , we compute the following:  
 2

$$3 \quad \mathbf{e}_t = \mathcal{T}(\mathbf{y}_t) - \hat{\mathbf{y}}_t \quad (8.248)$$

$$4 \quad \mathbf{R}_t = \text{Cov} [\mathcal{T}(\mathbf{y}) | \hat{\mathbf{y}}_t] \quad (8.249)$$

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

$$10 \quad \mathbf{H}_t = \frac{\partial h}{\partial \mathbf{z}}|_{(\boldsymbol{\mu}_{t|t-1}, \mathbf{u}_t)} \quad (8.250)$$

$$11 \quad \mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top (\mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\top + \mathbf{R}_t)^{-1} \quad (8.251)$$

$$12 \quad \boldsymbol{\Sigma}_t = (\mathbf{I} - \mathbf{K}_t \mathbf{H}_t) \boldsymbol{\Sigma}_{t|t-1} \quad (8.252)$$

$$13 \quad \boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t \mathbf{e}_t \quad (8.253)$$

14     In [Oll18], they show that this is equivalent to an online version of natural gradient descent  
 15    (Section 6.4).  
 16

#### 20 8.5.4.3 EEKF for training logistic regression

21     For example, consider the case where  $y$  is a class label with  $C$  possible values. (We drop the time  
 22    index for brevity.) Following Section 2.5.2.2, Let

$$23 \quad \mathcal{T}(\mathbf{y}) = [\mathbb{I}(y=1), \dots, \mathbb{I}(y=C-1)] \quad (8.254)$$

24     be the  $(C-1)$ -dimensional vector of sufficient statistics, and let  $\hat{\mathbf{y}} = [p_1, \dots, p_{C-1}]$  be the corre-  
 25    sponding predicted probabilities of each class label. The probability of the  $C$ 'th class is given by  
 26     $p_C = 1 - \sum_{c=1}^{C-1} \hat{y}_c$ ; we avoid including this to ensure that  $\mathbf{R}$  is not singular. The  $(C-1) \times (C-1)$   
 27    covariance matrix  $\mathbf{R}$  is given by  
 28

$$29 \quad R_{ij} = \text{diag}(p_i) - p_i p_j \quad (8.255)$$

30     Now consider the simpler case where we have two class labels, so  $C = 2$ . In this case,  $\mathcal{T}(\mathbf{y}) =$   
 31     $\mathbb{I}(y=1)$ , and  $\hat{\mathbf{y}} = p(\mathbf{y}=1) = p$ . The covariance matrix of the observation noise becomes the scalar  
 32     $r = p(1-p)$ . Of course, we can make the output probabilities depend on the input covariates, as  
 33    follows:  
 34

$$35 \quad p(y_t | \mathbf{z}_t, \mathbf{u}_t) = \text{Ber}(y_t | \boldsymbol{\sigma}(\mathbf{z}_t^\top \mathbf{u}_t)) \quad (8.256)$$

36     We can use the exponential family representation of the output discussed in Section 8.5.4.3.  
 37

38     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  
 39    sequentially compute the posterior using the EEKF, and compare to the offline estimate computed  
 40    using a Laplace approximation (where the MAP estimate is computed using BFGS) and an MCMC  
 41    approximation, which we take as “ground truth”. In Figure 8.13c, we see that the resulting posterior  
 42    predictive distributions are similar. Finally, in Figure 8.14, we visualize how the posterior marginals  
 43    converge over time. (See also Section 8.8.4, where we solve this same problem using ADF.)  
 44

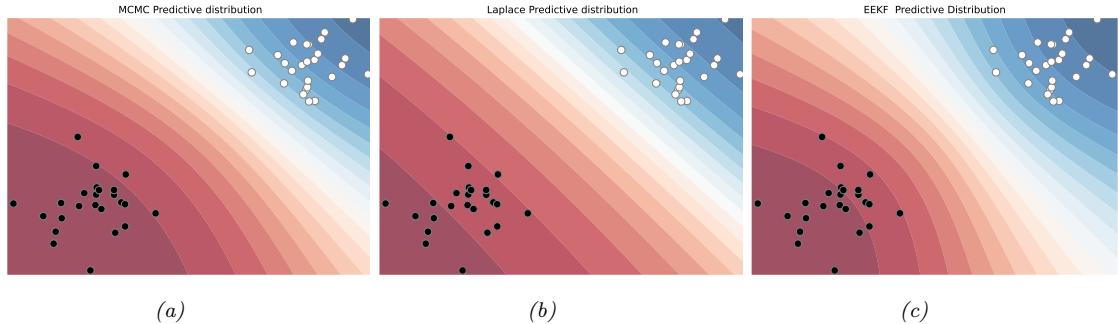


Figure 8.13: Bayesian inference applied to a 2d binary logistic regression problem,  $p(y = 1|\mathbf{x}) = \sigma(w_0 + w_1x_1 + w_2x_2)$ . We show the training data and the posterior predictive produced by different methods. (a) Offline MCMC approximation. (b) Offline Laplace approximation. (c) Online EKF approximation at the final step of inference. Generated by `eekf_logistic_regression.py`.

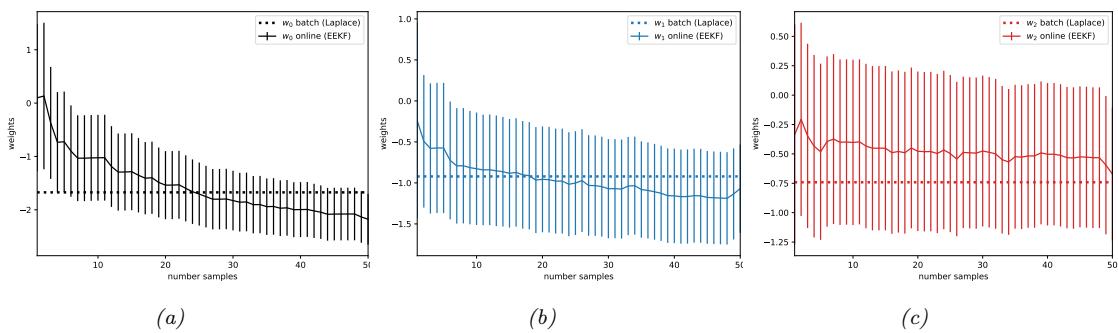


Figure 8.14: Marginal posteriors over time for the EKF method. The horizontal line is the offline MAP estimate. Generated by `eekf_logistic_regression.py`.

## 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

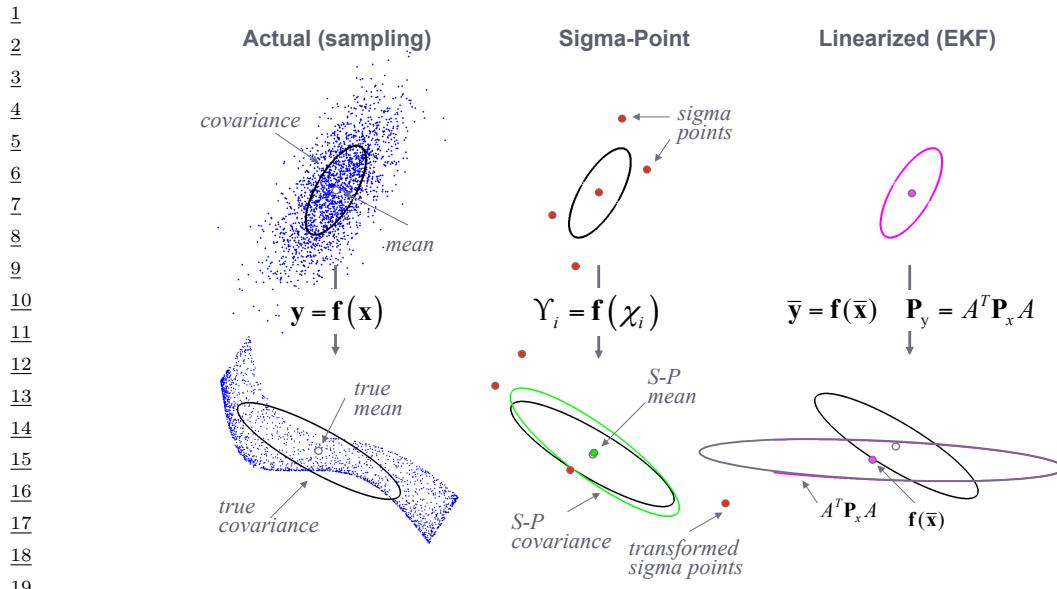


Figure 8.15: An example of the unscented transform in two dimensions. From [WM01]. Used with kind permission of Eric Wan.

[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  $\lambda$  is a scaling parameter given by

$$\lambda = \alpha^2(d + \kappa) - d \quad (8.260)$$

2. Propagate the sigma points through the nonlinear function to get the following  $2d + 1$  outputs:

$$\mathbf{y}_i = \mathbf{f}(\mathbf{y}_i) \quad (8.261)$$

1  
2 3. Estimate the mean and covariance of the resulting bag of points:

3  
4  $\mathbb{E}[\mathbf{f}(\mathbf{z})] \approx \boldsymbol{\mu}_U = \sum_{i=0}^{2d} w_i^m \mathbf{y}_i$  (8.262)

5  
6  $\text{Cov}[\mathbf{f}(\mathbf{z})] \approx \mathbf{S}_U = \sum_{i=0}^{2d} w_i^c (\mathbf{y}_i - \boldsymbol{\mu}_U)(\mathbf{y}_i - \boldsymbol{\mu}_U)^\top$  (8.263)

7  
8 where the  $w$ 's are weighting terms, given by the following:

9  
10  $w_0^m = \frac{\lambda}{d + \lambda}$  (8.264)

11  
12  $w_0^c = \frac{\lambda}{d + \lambda} + (1 - \alpha^2 + \beta)$  (8.265)

13  
14  $w_i^m = w_i^c = \frac{1}{2(d + \lambda)}$  (8.266)

15  
16 In general, the optimal values of  $\alpha$ ,  $\beta$  and  $\kappa$  are problem dependent. A typical recommendation  
17 is  $\alpha = 10^{-3}$ ,  $\kappa = 1$ ,  $\beta = 2$  [Bit16]. See Figure 8.10b for a 1d illustration and Figure 8.15 for a 2d  
18 illustration.

19  
20 Now suppose we want to approximate the joint distribution  $p(\mathbf{z}, \mathbf{y})$ , where  $\mathbf{y} = \mathbf{f}(\mathbf{z}) + \mathbf{q}$ , and  
21  $\mathbf{q} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ . We have

22  
23 
$$\begin{pmatrix} \mathbf{z} \\ \mathbf{y} \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_U \end{pmatrix}, \begin{pmatrix} \Sigma & \mathbf{C}_U \\ \mathbf{C}_U^\top & \mathbf{S}_u \end{pmatrix}\right)$$
 (8.267)

24  
25 where

26  
27 
$$\boldsymbol{\mu}_U = \sum_{i=0}^{2d} w_i^m \mathbf{y}_i$$
 (8.268)

28  
29 
$$\mathbf{S}_U = \sum_{i=0}^{2d} w_i^c (\mathbf{y}_i - \boldsymbol{\mu}_U)(\mathbf{y}_i - \boldsymbol{\mu}_U)^\top + \mathbf{Q}$$
 (8.269)

30  
31 
$$\mathbf{C}_U = \sum_{i=0}^{2d} w_i^c (\mathbf{y}_i - \boldsymbol{\mu})(\mathbf{y}_i - \boldsymbol{\mu}_U)^\top$$
 (8.270)

32  
33 The unscented transform is a third-order method in the sense that the mean of  $\mathbf{y}$  is exact for  
34 polynomials up to order 3. However the covariance is only exact for linear functions.

### 35 36 8.6.2 The unscented Kalman filter (UKF)

37  
38 The UKF uses the unscented transform twice, once to approximate passing through the system model  
39  $\mathbf{f}$ , and once to approximate passing through the measurement model  $\mathbf{h}$ . The derivation is analogous  
40 to that of the EKF. The resulting algorithm is as follows.

### $\frac{1}{2}$ 8.6.2.1 Prediction step

<sup>3</sup> We perform these steps.

### 5 1. Form the sigma points

$$z_{t-1,0} = \mu_{t-1} \quad (8.271)$$

$$z_{t-1,i} = \mu_{t-1} + \sqrt{d+\lambda} [\sqrt{\Sigma_{t-1}}]_{:,i} \quad (8.272)$$

$$\tilde{z}_{t-1,i+n} = \mu_{t-1} - \sqrt{d+\lambda} [\sqrt{\Sigma_{t-1}}]_i \quad (8.273)$$

$\frac{11}{11}$  2 Propagate these points through the dynamics model:

$$\hat{z}_{t-i} = f(z_{t-1,i}) \quad (8.274)$$

15 3. Compute the predicted mean and covariance.

$$\boldsymbol{\mu}_{t|t-1} = \sum_{i=0}^{2d} w_i^m \hat{z}_{t,i} \quad (8.275)$$

$$\Sigma_{t|t-1} = \sum_{i=-\infty}^{2d} w_i^c (\hat{z}_{t,i} - \mu_{t|t-1})(\hat{z}_{t,i} - \mu_{t|t-1})^\top + \mathbf{Q}_t \quad (8.276)$$

### 23 8.6.2.2 Update step

<sup>24</sup> We perform those steps

26 1. Form the sigma points

$$z_{t+0} \equiv u_{t+1} \quad (8.277)$$

$$z_{t,i} = \mu_{t|t-1} + \sqrt{d+\lambda} [\sqrt{\Sigma_{t|t-1}}]_{:i} \quad (8.278)$$

$$z_{t,i+n} = \mu_{t|t-1} - \sqrt{d+\lambda} [\sqrt{\Sigma_{t|t-1}}]_{:i} \quad (8.279)$$

<sup>33</sup> 2. Propagate these points through the measurement model:

$$\hat{z}_{t+1} = \mathbf{f}(z_{t+1}) \quad (8.280)$$

37 3. Compute the predicted mean and covariance of  $p(\mathbf{y}_t, \mathbf{z}_t | \mathbf{y}_{1:t-1})$ :

$$\mathbf{m}_t = \sum_i^{2d} w_i^m \hat{\mathbf{z}}_{t,i} \quad (8.281)$$

$$\mathbf{S}_t = \sum_{i=0}^{2d} w_i^c (\hat{\mathbf{z}}_{t,i} - \mathbf{m}_t) (\hat{\mathbf{z}}_{t,i} - \mathbf{m}_t)^T + \mathbf{R}_t \quad (8.282)$$

$$\mathbf{C}_t = \sum_{i=1}^{2d} w_i^c (\hat{\mathbf{z}}_{t,i} - \boldsymbol{\mu}_{t|t-1})(\hat{\mathbf{z}}_{t,i} - \mathbf{m}_t)^\top + \mathbf{R}_t \quad (8.283)$$

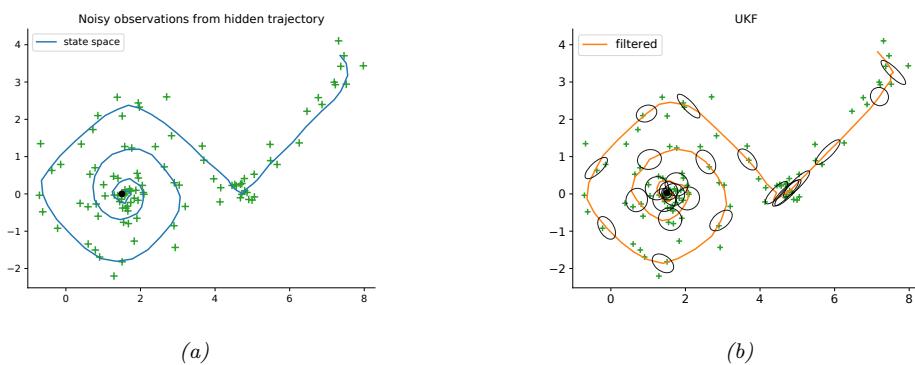


Figure 8.16: Illustration of UKF applied to a 2d nonlinear dynamical system. (a) True underlying state and observed data. (b) UKF estimate. Generated by `ekf_vs_ukf.py`.

4. Apply Bayes rule for Gaussians to get the posterior:

$$\mathbf{K}_t = \mathbf{C}_t \mathbf{S}_t^{-1} \quad (8.284)$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t (\mathbf{y}_t - \mathbf{m}_t) \quad (8.285)$$

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t \mathbf{S}_t \mathbf{K}_t^\top \quad (8.286)$$

### 8.6.2.3 Example: noisy 2d tracking problem

Let us revisit the 2d nonlinear tracking problem from Section 31.3.1. In Figure 8.16b, we see that the UKF algorithm (with  $\alpha = 1$ ,  $\beta = 0$ ,  $\kappa = 2$ ) works well on this problem.

### 8.6.3 The unscented Kalman smoother

The unscented Kalman smoother is a simple modification of the usual Kalman smoothing step, and is given by the following:

$$p(\mathbf{z}_t | \mathbf{y}_{1:T}) = \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{t|T}, \boldsymbol{\Sigma}_{t|T}) \quad (8.287)$$

$$\boldsymbol{\mu}_{t|T} = \boldsymbol{\mu}_{t|t} + \mathbf{G}_t (\boldsymbol{\mu}_{t+1|T} - \boldsymbol{\mu}_{t+1|t}) \quad (8.288)$$

$$\boldsymbol{\Sigma}_{t|T} = \boldsymbol{\Sigma}_t + \mathbf{G}_t (\boldsymbol{\Sigma}_{t+1|T} - \boldsymbol{\Sigma}_{t+1|t}) \mathbf{G}_t^\top \quad (8.289)$$

$$\mathbf{G}_t = \mathbf{D}_{t+1} \boldsymbol{\Sigma}_{t+1|t}^{-1} \quad (8.290)$$

$$\mathbf{D}_{t+1} = \sum_{i=0}^{2d} w_i^c (\mathbf{z}_{t,i} - \boldsymbol{\mu}_t)(\mathbf{z}_{t+1,i} - \boldsymbol{\mu}_{t+1|t})^\top \quad (8.291)$$

See e.g., [Sar13, Sec 9.3] for the derivation.

1 **8.7 Other variants of the Kalman filter**

3 In this section, we briefly mention some other variants of Kalman filtering. For a more extensive  
4 review, see [Li+17d].  
5

6 **8.7.1 Ensemble Kalman filter**

9 The **ensemble Kalman filter (EnKF)** is a technique developed in the geoscience (meteorology)  
10 community to perform approximate online inference in large nonlinear systems. The canonical  
11 reference is [Eve09], but a more accessible tutorial (using the same Bayesian signal processing  
12 approach we adopt in this chapter) is in [Rot+17].

13 EnKF is mostly used for problems where the hidden state represents an unknown physical quantity  
14 (e.g., temperature and pressure) at each point on a spatial grid, and the measurements are sparse and  
15 spatially localized. Combining this information over space and time is called **data assimilation**.  
16 However, the technique can be applied to other nonlinear state estimation problems. For example, it  
17 was recently used in [Li+20b] to model the spread of the SARS-CoV2 virus, using an ODE dynamics  
18 model, based on the EnKF method described in [And01]. We briefly explain the method below,  
19 following [Rot+17].

20 The key idea is to represent the belief state  $p(\mathbf{z}_t | \mathbf{y}_{1:t})$  by a finite number of samples  $\mathbf{z}_{t|t} = \{\mathbf{z}_{t|t}^s : s = 1 : S\}$ , where each  $\mathbf{z}_{t|t}^s \in N_z$ . In contrast to particle filtering (Section 13.2), the samples are  
21 updated in a manner that closely resembles the Kalman filter, so there is no importance sampling or  
22 resampling step. The downside is that the posterior does not converge to the true Bayesian posterior  
23 even as  $S \rightarrow \infty$  [LGMT11], except in the linear-Gaussian case. However, sometimes the performance  
24 of EnKF can be better for small number of samples (although this depends of course on the PF  
25 proposal distribution).

26 The posterior mean and covariance can be derived from the ensemble of samples as follows:

$$\tilde{\mu}_{t|t} = \frac{1}{S} \sum_{s=1}^S \mathbf{z}_{t|t}^s = \frac{1}{S} \mathbf{z}_{t|t} \mathbf{1} \quad (8.292)$$

$$\tilde{\Sigma}_{t|t} = \frac{1}{S-1} \sum_{s=1}^S (\mathbf{z}_{t|t}^s - \tilde{\mu}_{t|t})(\mathbf{z}_{t|t}^s - \tilde{\mu}_{t|t})^\top = \frac{1}{S-1} \tilde{\mathbf{Z}}_{t|t} \tilde{\mathbf{Z}}_{t|t}^\top \quad (8.293)$$

35 where  $\tilde{\mathbf{Z}}_{t|t} = \mathbf{z}_{t|t} - \tilde{\mu}_{t|t} \mathbf{1}^\top$ .

36 We update the samples as follows. For the time update, we first draw  $S$  system noise variables  
37  $\mathbf{v}_t^s \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ , and then we pass these, and the previous state estimate, through the dynamics  
38 model to get the one-step-ahead state predictions,  $\mathbf{z}_{t|t-1} = f_z(\mathbf{z}_{t-1|t-1}, \mathbf{V}_t)$ . This is the analog of  
39 Equation (8.114).

40 Next we draw  $S$  observation noise variables  $\mathbf{e}_t^s \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ , and use them to compute the one-step-  
41 ahead observation predictions,  $\mathbf{y}_{t|t-1} = f_x(\mathbf{z}_{t|t-1}, \mathbf{E}_t)$ . This is the analog of Equation (8.115).

42 Finally we compute the measurement update using

$$\mathbf{z}_{t|t} = \mathbf{z}_{t|t-1} + \tilde{\mathbf{K}}_t (\mathbf{y}_t \mathbf{1}^\top - \mathbf{y}_{t|t-1}) \quad (8.294)$$

45 which is the analog of Equation (8.95).  
46

47

We now discuss how to compute  $\tilde{\mathbf{K}}_t$ , which is the analog of the Kalman gain matrix in Equation (8.94). First note that we can write the exact Kalman gain matrix (in the linear-Gaussian case) as  $\mathbf{K}_t = \Sigma_{t|t-1} \mathbf{C}^\top \mathbf{S}_t^{-1} = \mathbf{M}_t \mathbf{S}_t^{-1}$ , where  $\mathbf{S}_t$  is the covariance of the measurements, and  $\mathbf{M}_t$  is the cross-covariance between the state and output predictions. In the EnKF, we approximate  $\mathbf{S}_t$  and  $\mathbf{M}_t$  empirically as follows. First we compute the deviations from predictions:

$$\tilde{\mathbf{Z}}_{t|t-1} = \mathbf{z}_{t|t-1} (\mathbf{I} - \frac{1}{N} \mathbf{1} \mathbf{1}^\top), \quad \tilde{\mathbf{Y}}_{t|t-1} = \mathbf{y}_{t|t-1} (\mathbf{I} - \frac{1}{N} \mathbf{1} \mathbf{1}^\top) \quad (8.295)$$

Then we compute the sample covariance matrices

$$\tilde{\mathbf{S}}_t = \frac{1}{S-1} \tilde{\mathbf{Y}}_{t|t-1} \tilde{\mathbf{Y}}_{t|t-1}^\top, \quad \tilde{\mathbf{M}}_t = \frac{1}{S-1} \tilde{\mathbf{Z}}_{t|t-1} \tilde{\mathbf{Y}}_{t|t-1}^\top \quad (8.296)$$

Finally we compute

$$\tilde{\mathbf{K}}_t = \tilde{\mathbf{M}}_t \tilde{\mathbf{S}}_t^{-1} \quad (8.297)$$

In practice, we should not perform this matrix inversion, but instead solve the linear system

$$\tilde{\mathbf{K}}_t \tilde{\mathbf{Y}}_{t|t-1} \tilde{\mathbf{Y}}_{t|t-1}^\top = \tilde{\mathbf{Z}}_{t|t-1} \tilde{\mathbf{Y}}_{t|t-1}^\top \quad (8.298)$$

We now compare the computational complexity to the KF algorithm. We will assume  $N_z > S > N_y$ , as occurs in most geospatial problems. The EnKF time update takes  $O(N_z^2 S)$  operations, and the measurement update takes  $O(N_z N_y S)$ , where  $N_z$  is the number of latent dimensions and  $N_y$  is the number of observed dimensions. By contrast, in the KF, the time update takes  $O(N_z^3)$  operations, and the measurement update takes  $O(N_z^2 N_y)$ . So we see that the EnKF is faster for high dimensional state-spaces, because it uses a low-rank approximation to the posterior covariance.

Unfortunately, if  $S$  is too small, the EnKF can become overconfident, and the filter can diverge. Various heuristics (e.g., covariance inflation) have been proposed to fix this. However, most of these methods are ad-hoc. A variety of more well-principled solutions have also been proposed, see e.g., [FK13b; Rei13].

### 8.7.2 Robust Kalman filters

In practice we often have noise that is non-Gaussian. A common example is when we have clutter, or outliers, in the observation model, or sudden changes in the process model. In this case, we might use the Laplace distribution [Ara+09] or the Student- $t$  distribution [Ara10; RÖG13; Ara+17] as noise models.

[Hua+17b] proposes a variational Bayes (Section 10.2.3) approach, that allows the dynamical prior and the observation model to both be (linear) Student distributions, but where the posterior is approximated at each step using a Gaussian, conditional on the noise scale matrix, which is modeled using an inverse Wishart distribution. An extension of this, to handle mixture distributions, can be found in [Hua+19b].

### 8.7.3 Gaussian filtering

In this section, we discuss a simple unified framework, known as the **Gaussian filter** [IX00; Wu+06], which includes EKF, UKF, and various other algorithms. Our presentation is based on [Sar13, Ch. 6].

1    **8.7.3.1 The Gaussian approximation**

3 To explain the approach, we temporarily drop the time indices, and the conditioning on past  
4 information, and consider a single time step of inference. Furthermore, we will use the shorthand  
5

6    
$$\int_x f(x) = \int_{-\infty}^{\infty} f(x) dx \tag{8.299}$$
  
7

9    Let  $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z)$  and  $p(\mathbf{y}|\mathbf{z}) = \mathcal{N}(\mathbf{y}|\mathbf{g}(\mathbf{z}), \mathbf{Q})$  for some function  $\mathbf{g}$ . Let  $p(\mathbf{z}, \mathbf{y}) = p(\mathbf{z})p(\mathbf{y}|\mathbf{z})$   
10 be the exact joint distribution. The best Gaussian approximation to the joint can be obtained by  
11 **moment matching**, i.e.,

12    
$$q(\mathbf{z}, \mathbf{y}) = \mathcal{N}\left(\begin{pmatrix} \mathbf{z} \\ \mathbf{y} \end{pmatrix} \mid \begin{pmatrix} \boldsymbol{\mu}_z \\ \boldsymbol{\mu}_y \end{pmatrix}, \begin{pmatrix} \boldsymbol{\Sigma}_z & \boldsymbol{\Sigma}_{zy} \\ \boldsymbol{\Sigma}_{zy}^\top & \boldsymbol{\Sigma}_y \end{pmatrix}\right) \tag{8.300}$$
  
13

14 where

15    
$$\boldsymbol{\mu}_y = \int_z \mathbf{g}(\mathbf{z}) \mathcal{N}(\mathbf{z}|\mathbf{m}, \boldsymbol{\Sigma}) \tag{8.301}$$
  
16

17    
$$\boldsymbol{\Sigma}_y = \int_z (\mathbf{g}(\mathbf{z}) - \boldsymbol{\mu}_y)(\mathbf{g}(\mathbf{z}) - \boldsymbol{\mu}_y)^\top \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z) + \mathbf{Q} \tag{8.302}$$
  
18

19    
$$\boldsymbol{\Sigma}_{zy} = \int_z (\mathbf{z} - \boldsymbol{\mu}_z)(\mathbf{g}(\mathbf{z}) - \boldsymbol{\mu}_y)^\top \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z) \tag{8.303}$$
  
20

21 We can either compute these integrals by linearizing  $\mathbf{g}$  and using closed form expressions, or by using  
22 numerical integration, as we discuss in Section 8.7.3.3.

23 Once we have computed the joint  $q(\mathbf{z}, \mathbf{y})$ , we can compute the posterior conditional  $q(\mathbf{z}|\mathbf{y})$  using  
24 the usual rules for conditioning a Gaussian:

25    
$$q(\mathbf{z}|\mathbf{y}) = \mathcal{N}\left(\mathbf{z} \mid \underbrace{\boldsymbol{\mu}_z + \boldsymbol{\Sigma}_{zy} \boldsymbol{\Sigma}_y^{-1}(\mathbf{y} - \boldsymbol{\mu}_y)}_{\boldsymbol{\mu}_{z|y}}, \underbrace{\boldsymbol{\Sigma}_z - \boldsymbol{\Sigma}_{zy} \boldsymbol{\Sigma}_y^{-1} \boldsymbol{\Sigma}_{zy}^\top}_{\boldsymbol{\Sigma}_{zz|y}}\right) \tag{8.304}$$
  
26

27 In practice,  $\boldsymbol{\Sigma}_y$  may be rank deficient, so we should avoid computing  $\boldsymbol{\Sigma}_y^{-1}$ . Fortunately we can  
28 compute  $\boldsymbol{\mu}_{z|x}$  and  $\boldsymbol{\Sigma}_{zz|x}$  in a numerically stable way by solving a linear system. In particular, the  
29 posterior mean is the solution to

30    
$$\boldsymbol{\mu}_{z|y} = \boldsymbol{\mu}_z + \boldsymbol{\Sigma}_{zy} \mathbf{a} \tag{8.305}$$
  
31

32    
$$\boldsymbol{\Sigma}_y \mathbf{a} = \mathbf{y} - \boldsymbol{\mu}_y \tag{8.306}$$
  
33

34 and the posterior covariance is the solution to

35    
$$\boldsymbol{\Sigma}_{zz|y} = \boldsymbol{\Sigma}_z - \boldsymbol{\Sigma}_{zy} \mathbf{A} \tag{8.307}$$
  
36

37    
$$\boldsymbol{\Sigma}_y \mathbf{A} = \boldsymbol{\Sigma}_{zy}^\top \tag{8.308}$$
  
38

39 These solutions are unique, even if  $\boldsymbol{\Sigma}_y$  is degenerate, as shown in [Wütt+16, App. A].

40

---

### 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  $\kappa$  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.

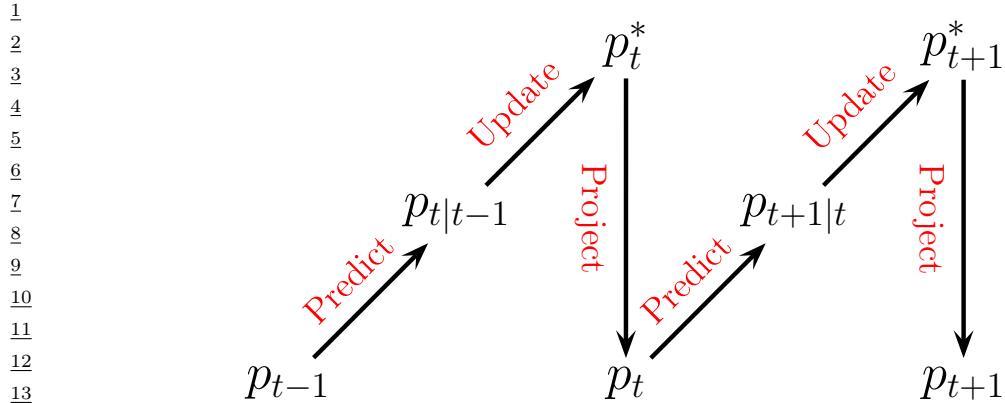


Figure 8.17: Illustration of the predict-update-project cycle of assumed density filtering.

## 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:

$$J(q) = - \int_{\mathbf{z}, \mathbf{y}} p^*(\mathbf{z}, \mathbf{y}) \log q(\mathbf{z}|\mathbf{y}) \quad (8.324)$$

$$\begin{aligned} &= \int_{\mathbf{z}, \mathbf{y}} p^*(\mathbf{z}|\mathbf{y}) p^*(\mathbf{y}) \log \left( \frac{p^*(\mathbf{z}|\mathbf{y})}{q(\mathbf{z}|\mathbf{y})} \right) - \underbrace{\int_{\mathbf{z}, \mathbf{y}} p^*(\mathbf{z}, \mathbf{y}) \log p^*(\mathbf{z}|\mathbf{y})}_{c} \end{aligned} \quad (8.325)$$

$$= \mathbb{E}_{\mathbf{y}} [D_{\text{KL}}(p^*(\mathbf{z}|\mathbf{y}) \| q(\mathbf{z}|\mathbf{y}))] + c \quad (8.326)$$

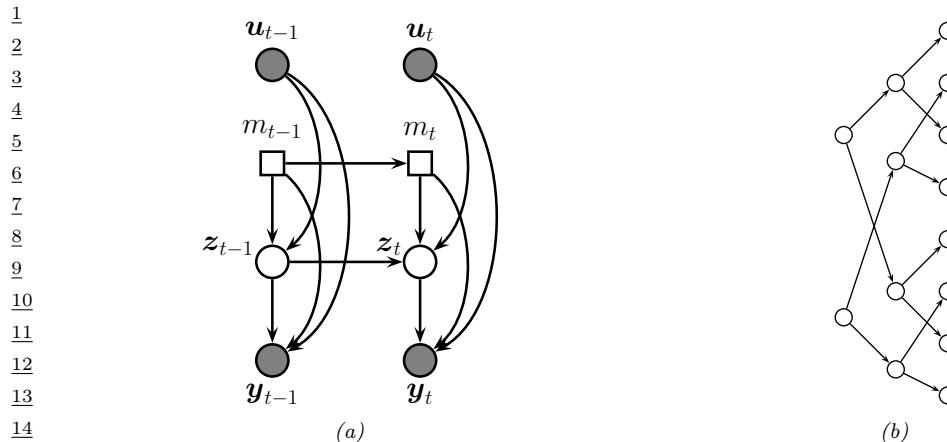
Thus we see that Gaussian filtering is like an “averaged” version of ADF. In particular, GF takes expectations wrt  $p^*(\mathbf{z}, \mathbf{y})$ , which is easier to approximate than taking expectations wrt  $p^*(\mathbf{z}|\mathbf{y})$ , as required by ADF. See [Wüt+16] for further discussion.

## 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.*

20 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  
21 discrete variable can be used to represent different kinds of system dynamics or operating regimes  
22 (e.g., normal or abnormal), or different kinds of observation models (e.g., to handle outliers due to  
23 sensor noise or failures).

<sup>25</sup> If the system is linear-Gaussian, it is called a **switching linear dynamical system (SLDS)**, or  
<sup>26</sup> a **jump Markov linear system (JMLS)** [DGK01]. This corresponds to the following model:

$$p(c_t = k | c_{t-1} = i) = A_{ik} \quad (8.327)$$

$$p(z_t | z_{t-1}, c_t = k, \mathbf{u}_t) = \mathcal{N}(z_t | \mathbf{F}_k z_{t-1} + \mathbf{B}_k \mathbf{u}_t, \mathbf{Q}_k) \quad (8.328)$$

$$p(\mathbf{u}_t | \mathbf{z}_t, c_t = k, \mathbf{u}^*) = \mathcal{N}(\mathbf{u}_t | \mathbf{H}_k \mathbf{z}_t + \mathbf{B}_k \mathbf{u}^*, \mathbf{R}_k) \quad (8.329)$$

<sup>32</sup> 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.

### $\frac{37}{8.8}$ 8.8.3.2 Posterior inference

<sup>39</sup> Unfortunately exact inference in such switching models is intractable, even in the linear Gaussian  
<sup>40</sup> case. To see why, suppose for simplicity that the latent discrete switching variable  $c_t$  is binary,  
<sup>41</sup> and that only the dynamics matrix  $\mathbf{F}$  depend on  $c_t$ , not the observation matrix  $\mathbf{H}$ . Our initial  
<sup>42</sup> belief state will be a mixture of 2 Gaussians, corresponding to  $p(\mathbf{z}_1|\mathbf{y}_1, c_1 = 1)$  and  $p(\mathbf{z}_1|\mathbf{y}_1, c_1 = 2)$ .  
<sup>43</sup> The one-step-ahead predictive density will be a mixture of 4 Gaussians  $p(\mathbf{z}_2|\mathbf{y}_1, c_1 = 1, c_2 = 1)$ ,  
<sup>44</sup>  $p(\mathbf{z}_2|\mathbf{y}_1, c_1 = 1, c_2 = 2)$ ,  $p(\mathbf{z}_2|\mathbf{y}_1, c_1 = 2, c_2 = 1)$ , and  $p(\mathbf{z}_2|\mathbf{y}_1, c_1 = 2, c_2 = 2)$ , obtained by passing  
<sup>45</sup> each of the prior modes through the 2 possible transition models. The belief state at step 2 will  
<sup>46</sup> also be a mixture of 4 Gaussians, obtained by updating each of the above distributions with  $\mathbf{y}_2$ .

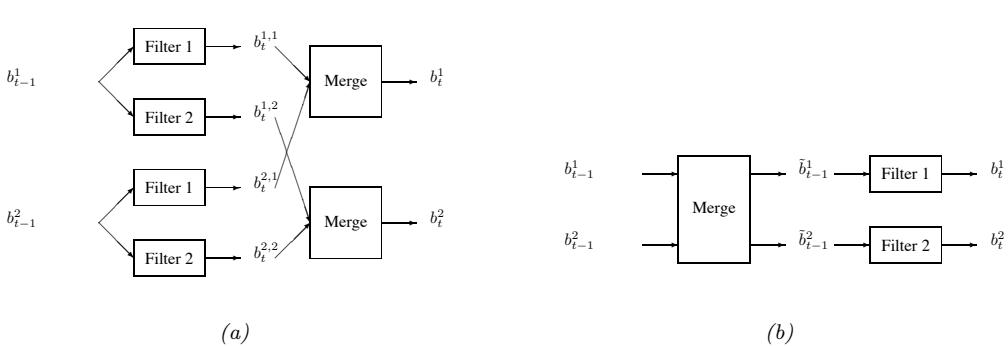


Figure 8.19: ADF for a switching linear dynamical system with 2 discrete states. (a) GPB2 method. (b) IMM method.

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

$$b_t^{ij} \triangleq p(\mathbf{z}_t, c_{t-1} = i, c_t = j | \mathbf{y}_{1:t}) = \pi_{tij} \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{t,ij}, \boldsymbol{\Sigma}_{t,ij}) \quad (8.331)$$

1 where  $\pi_{tij} = \pi_{t-1,i} p(c_t = j | c_{t-1} = i)$ . Finally, for each value of  $j$ , we collapse the  $K$  Gaussian  
 2 mixtures down to a single mixture to give  
 3

$$4 b_t^j \triangleq p(\mathbf{z}_t, c_t = j | \mathbf{y}_{1:t}) = \pi_{tj} \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{t,j}, \boldsymbol{\Sigma}_{t,j}) \quad (8.332) \\ 5$$

6 See Figure 8.19a for a sketch.  
 7

8 The optimal way to approximate a mixture of Gaussians with a single Gaussian is given by  
 9  $q = \arg \min_q D_{\text{KL}}(q \| p)$ , where  $p(\mathbf{z}) = \sum_k \pi_k \mathcal{N}(\mathbf{z} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$  and  $q(\mathbf{z}) = \mathcal{N}(\mathbf{z} | \boldsymbol{\mu}, \boldsymbol{\Sigma})$ . This can be solved  
 10 by moment matching, that is,

$$11 \boldsymbol{\mu} = \mathbb{E}[\mathbf{z}] = \sum_k \pi_k \boldsymbol{\mu}_k \quad (8.333) \\ 12$$

$$13 \boldsymbol{\Sigma} = \text{Cov}[\mathbf{z}] = \sum_k \pi_k (\boldsymbol{\Sigma}_k + (\boldsymbol{\mu}_k - \boldsymbol{\mu})(\boldsymbol{\mu}_k - \boldsymbol{\mu})^\top) \quad (8.334) \\ 14$$

15 In the graphical model literature, this is called **weak marginalization** [Lau92], since it preserves  
 16 the first two moments. Applying these equations to our model, we can go from  $b_t^{ij}$  to  $b_t^j$  as follows  
 17 (where we drop the  $t$  subscript for brevity):  
 18

$$19 \pi_j = \sum_i \pi_{ij} \quad (8.335) \\ 20$$

$$21 \pi_{j|i} = \frac{\pi_{ij}}{\sum_{j'} \pi_{ij'}} \quad (8.336) \\ 22$$

$$23 \boldsymbol{\mu}_j = \sum_i \pi_{j|i} \boldsymbol{\mu}_{ij} \quad (8.337) \\ 24$$

$$25 \boldsymbol{\Sigma}_j = \sum_i \pi_{j|i} (\boldsymbol{\Sigma}_{ij} + (\boldsymbol{\mu}_{ij} - \boldsymbol{\mu}_j)(\boldsymbol{\mu}_{ij} - \boldsymbol{\mu}_j)^\top) \quad (8.338) \\ 26$$

27 This algorithm requires running  $K^2$  filters at each step. A cheaper alternative, known as **interactive multiple models** or **IMM** [BSF88], can be obtained by first collapsing the prior to a single  
 28 Gaussian (by moment matching), and then updating it using  $K$  different Kalman filters, one per  
 29 value of  $c_t$ . See Figure 8.19b for a sketch.

30

#### 31 8.8.4 ADF for training logistic regression

32 In this section, we discuss how to use the assumed density filtering algorithm of Section 8.8 to  
 33 recursively compute (i.e., in an online fashion) the (approximate) posterior  $p(\mathbf{w}_t | \mathcal{D}_{1:t})$  for a logistic  
 34 regression model using a Gaussian prior, where  $\mathcal{D}_{1:t} = \{(\mathbf{y}_n, y_n) : n = 1 : t\}$  is all the data we have  
 35 seen so far. This is particularly useful in cases where the data is arriving in a continual stream, such  
 36 as online advertising (see e.g., [Gra+10]) and recommender systems (see e.g., [Aga+14]). We follow  
 37 the presentation of [Zoe07]. (See also Section 17.6.2 where we extend this to MLPs.)

38 We assume our model has the following form:  
 39

$$40 p(y_t | \mathbf{y}_t, \mathbf{w}_t) = \text{Ber}(y_t | \boldsymbol{\sigma}(\mathbf{y}_t^\top \mathbf{w}_t)) \quad (8.339) \\ 41$$

$$42 p(\mathbf{w}_t | \mathbf{w}_{t-1}) = \mathcal{N}(\mathbf{w}_t | \mathbf{w}_{t-1}, \mathbf{Q}) \quad (8.340) \\ 43$$

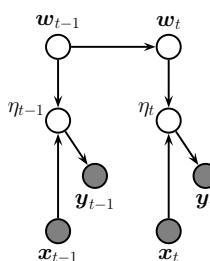


Figure 8.20: A dynamic logistic regression model.  $\mathbf{w}_t$  are the regression weights at time  $t$ , and  $\eta_t = \mathbf{w}_t^\top \mathbf{y}_t$ . Compare to Figure 8.9a.

where  $\mathbf{Q}$  is the covariance of the process noise, which allows the parameters to change slowly over time. We will assume  $\mathbf{Q} = \epsilon \mathbf{I}$ ; we can also set  $\epsilon = 0$ , as in the recursive least squares method (Section 8.4.2), if we believe the parameters will not change. See Figure 8.20 for an illustration of the model.

As our approximating family, we will use diagonal Gaussians, for computational efficiency. Thus the prior is the posterior from the previous timestep, and has the form

$$p(\mathbf{w}_{t-1} | \mathcal{D}_{1:t-1}) \approx p_{t-1}(\mathbf{w}_{t-1}) = \mathcal{N}(\mathbf{w}_{t-1} | \boldsymbol{\mu}_{t-1}, \text{diag}(\boldsymbol{\tau}_{t-1})) = \prod_j \mathcal{N}(w_{t-1,j} | \mu_{t-1,j}, \tau_{t-1,j}) \quad (8.341)$$

where  $\mu_{t-1,j}$  and  $\tau_{t-1,j}$  are the posterior mean variance for parameter  $j$  given past data. Now we discuss how to update this prior.

First we compute the one-step-ahead predictive density  $p_{t|t-1}(\mathbf{w}_t)$  using the standard linear-Gaussian update, i.e.,  $\boldsymbol{\mu}_{t|t-1} = \boldsymbol{\mu}_{t-1}$  and  $\boldsymbol{\tau}_{t|t-1} = \boldsymbol{\tau}_{t-1} + \mathbf{Q}$ .

Now we concentrate on the measurement update step. Define the scalar sum (corresponding to the logits, if we are using binary classification) as  $\eta_t = \mathbf{w}_t^\top \mathbf{y}_t$ . If  $p_{t|t-1}(\mathbf{w}_t) = \prod_j \mathcal{N}(w_{t,j} | \mu_{t|t-1,j}, \tau_{t|t-1,j})$ , then we can compute the prior predictive distribution for  $\eta_t$  as follows:

$$p(\eta_t | \mathcal{D}_{1:t-1}, \mathbf{y}_t) \approx p_{t|t-1}(\eta_t) = \mathcal{N}(\eta_t | m_{t|t-1}, v_{t|t-1}) \quad (8.342)$$

$$m_{t|t-1} = \sum_j x_{t,j} \mu_{t|t-1,j} \quad (8.343)$$

$$v_{t|t-1} = \sum_j x_{t,j}^2 \tau_{t|t-1,j} \quad (8.344)$$

1 The posterior for  $\eta_t$  is given by  
2

$$\underline{3} \quad p(\eta_t | \mathcal{D}_{1:t}) \approx p_t(\eta_t) = \mathcal{N}(\eta_t | m_t, v_t) \quad (8.345)$$

$$\underline{4} \quad m_t = \int \eta_t \frac{1}{Z_t} p(y_t | \eta_t) p_{t|t-1}(\eta_t) d\eta_t \quad (8.346)$$

$$\underline{5} \quad v_t = \int \eta_t^2 \frac{1}{Z_t} p(y_t | \eta_t) p_{t|t-1}(\eta_t) d\eta_t - m_t^2 \quad (8.347)$$

$$\underline{6} \quad Z_t = \int p(y_t | \eta_t) p_{t|t-1}(\eta_t) d\eta_t \quad (8.348)$$

7 where  $p(y_t | \eta_t) = \text{Ber}(y_t | \eta_t)$ . These integrals are one dimensional, and so can be efficiently computed  
8 using Gaussian quadrature, as explained in [Zoe07; KB00].

9 The above update is the same as one step of the unscented Kalman filtering algorithm, (Section 8.6.2), which also uses quadrature, as we discussed in Section 8.7.3. We can extend this  
10 approximation to the offline setting, where we see future data, using expectation propagation (Section 10.7). This computes a Gaussian approximation to the likelihood using a prior coming from the  
11 smoothing posterior (leaving the current observation out), as opposed to coming from the filtered  
12 posterior based just on past data. This is called **quadrature EP** [ZH05].

13 Having inferred  $p_t(\eta_t)$ , whether using one-step EKF or EP, we need to compute  $p_t(w | \eta_t)$ . This  
14 can be done as follows. Define  $\delta_m$  as the change in the mean and  $\delta_v$  as the change in the variance:

$$\underline{15} \quad m_t = m_{t|t-1} + \delta_m, \quad v_t = v_{t|t-1} + \delta_v \quad (8.349)$$

16 Then one can show that the new factored posterior over the model parameters is given by  
17

$$\underline{18} \quad p_t(w_{t,j}) = \mathcal{N}(w_{t,j} | \mu_{t,j}, \tau_{t,j}) \quad (8.350)$$

$$\underline{19} \quad \mu_{t,j} = \mu_{t|t-1,j} + a_j \delta_m \quad (8.351)$$

$$\underline{20} \quad \tau_{t,j} = \tau_{t|t-1,j} + a_j^2 \delta_v \quad (8.352)$$

$$\underline{21} \quad a_j \triangleq \frac{x_{t,j} \tau_{t|t-1,j}}{\sum_{j'} x_{t,j'}^2 \tau_{t|t-1,j}^2} \quad (8.353)$$

22 Thus we see that the parameters which correspond to inputs with larger magnitude (big  $|x_{t,j}|$ ) or  
23 larger uncertainty (big  $\tau_{t|t-1,j}$ ) get updated most, which makes intuitive sense.

24 As an example, consider again the 2d binary classification problem in Section 8.5.4.3. We  
25 sequentially compute the posterior using the ADF, and compare to the offline estimate computed  
26 using a Laplace approximation (where the MAP estimate is computed using BFGS) and an MCMC  
27 approximation, which we take as “ground truth”. In Figure 8.21, we see that the resulting posterior  
28 predictive distributions are similar. Finally, in Figure 8.22, we visualize how the posterior marginals  
29 converge over time.

30 Note that the whole algorithm only takes  $O(D)$  time and space per step, the same as SGD. However,  
31 unlike SGD, there are no step-size parameters, since the diagonal covariance implicitly specifies the  
32 size of the update for each dimension. Furthermore, we get a posterior approximation, not just a  
33 point estimate. And since it is an online algorithm, it can also handle massive datasets. [ZGH10]  
34 extend this approach to the multi-label setting, and use it for online ranking problems.

35

36

37

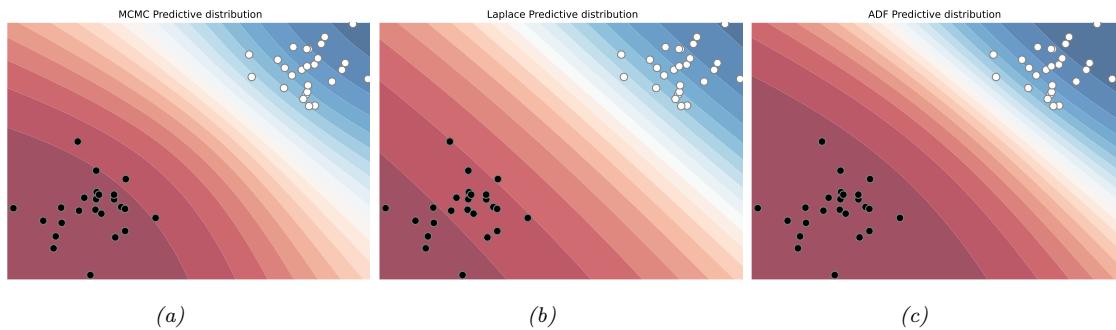


Figure 8.21: Bayesian inference applied to a 2d binary logistic regression problem,  $p(y = 1|\mathbf{x}) = \sigma(w_0 + w_1x_1 + w_2x_2)$ . We show the training data and the posterior predictive produced by different methods. (a) Offline MCMC approximation. (b) Offline Laplace approximation. (c) Online ADF approximation at the final step of inference. Generated by [adf logistic regression demo.py](#).

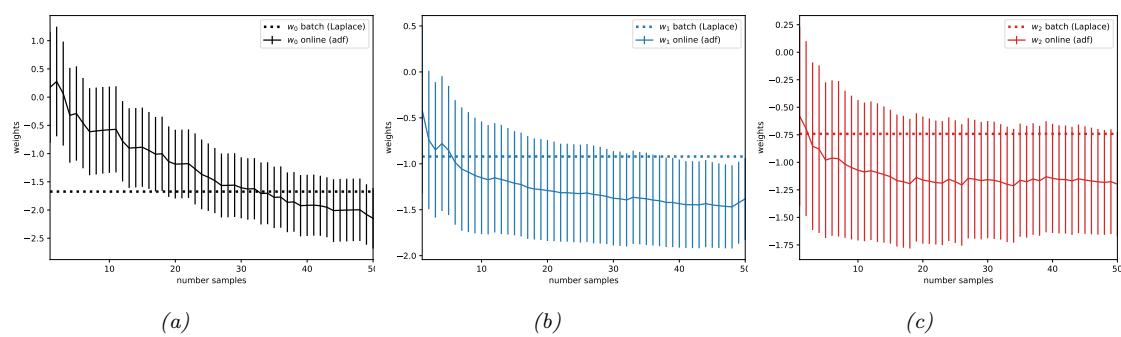


Figure 8.22: Marginal posteriors over time for the ADF method. The horizontal line is the offline MAP estimate. Generated by `adf_logistic_regression_demo.py`.



# 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  $\psi_c$  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  $\psi_c$  for brevity. This requires marginalizing over all the hidden variables. If each variable is discrete, with  $K$  possible states, and there are  $V$  such variables, this takes  $O(K^V)$  time to compute in the worst case. Similar computational problems arise with continuous state spaces.

The algorithms we discuss will leverage the conditional independence properties encoded in the graph structure in order to perform efficient inference. In particular, we will use the principle of **dynamic programming**, which finds an optimal solution by solving subproblems and then combining them. DP can be implemented by computing functions (such as posterior marginals) for each node (or clique) in the graph, and then sending **messages** to neighboring nodes (or cliques) so that all nodes can come to an overall consensus about the global solutions. Hence these are known as **message passing algorithms**.

1 Message passing generalizes the forwards-backwards algorithm discussed in Section 8.3.3, and  
 2 the Kalman smoothing algorithm discussed in Section 8.4.4, to work with general graph structures.  
 3 However, the resulting methods have a running time that is exponential in the treewidth of the graph.  
 4 (We define this in Section 9.4.2, but it is basically a measure of how non-treelike the graph is.) We  
 5 will therefore also consider various approximate inference algorithms. For more details that we don't  
 6 have space to cover, see e.g., [Yed11].  
 7

## 9.2 Belief propagation on trees

11 The forwards-backwards algorithm for HMMs (Section 8.3.3) and the Kalman smoother algorithm  
 12 for LDS (Section 8.4.4) can both be interpreted as **message passing** algorithms, that pass messages  
 13 along edges in the graph, in order to compute posterior marginals at each node, as illustrated in  
 14 Figure 8.6. The posterior marginals  $p(\mathbf{z}_t | \mathbf{x}_{1:t})$  and  $p(\mathbf{z}_t | \mathbf{x}_{1:T})$  are often called **belief states**, so these  
 15 algorithms are also called **belief propagation (BP)**) algorithms.

16 To implement such methods, we have to specify how to multiply messages together to compute a joint  
 17 distribution (product operation), and how to marginalize out some variables from a joint distribution  
 18 (sum operation). For discrete distributions, this just requires manipulating multidimensional tables  
 19 (tensors). For Gaussians, we can use the rules defined in Section 2.3.7. However, we can also generalize  
 20 these operations to any commutative semi-ring, as we explain in Section 9.5.3.

21 In addition to specifying the local update rules, we need to specify a **message passing schedule**  
 22 such that every node gets to “see” information from the entire rest of the graph exactly once (to  
 23 avoid overcounting of evidence). For chains the obvious schedule is left-to-right and then right-to-left,  
 24 as we illustrated above. For trees, we can go up to the root and then back down to the leaves, as we  
 25 discuss in Section 9.2.1. General graphs may have cycles or loops; we discuss this case in Section 9.3.  
 26

### 9.2.1 BP for polytrees

29 In this section, we generalize the forwards-backwards algorithm for chains to work on a **polytree**,  
 30 which is a directed graph whose undirected “backbone” is a tree, i.e., a graph with no loops. (That is,  
 31 a polytree is a directed tree with multiple root nodes, in which a node may have multiple parents,  
 32 whereas in a singly rooted tree, each node has a single parent.) This algorithm is called **belief  
 33 propagation** and is due to [Pea88].

34 We consider the case of a general discrete node  $X$  with parents  $U_i$  and children  $Y_j$ . We partition  
 35 the evidence in the graph,  $e$ , into the evidence upstream of node  $X$ ,  $e_X^+$ , and all the rest,  $e_X^-$ . Thus  
 36  $e_X^+$  contains all the evidence separated from  $X$  if its incoming arcs were deleted, and  $e_X^-$  contains the  
 37 evidence below  $X$  and the evidence in  $X$  itself, if any. The posterior on node  $X$  can be computed as  
 38 follows.

$$39 \quad \text{bel}_X(x) \triangleq \Pr(X = x | e) = c' \lambda_X(x) \pi_X(x) \quad (9.3)$$

$$41 \quad \lambda_X(x) \triangleq P(e_X^- | X = x) \quad (9.4)$$

$$42 \quad \pi_X(x) \triangleq \Pr(X = x | e_X^+) \quad (9.5)$$

44 where  $c'$  is a normalizing constant.

45 Consider the graph shown in Figure 9.1. We will use the notation  $e_{U_1 \rightarrow X}^+$  to denote the evidence  
 46 above the edge from  $U_1$  to  $X$  (i.e., in the “triangle” above  $U_1$ ), and  $e_{X \rightarrow Y_1}^-$  to denote the evidence  
 47

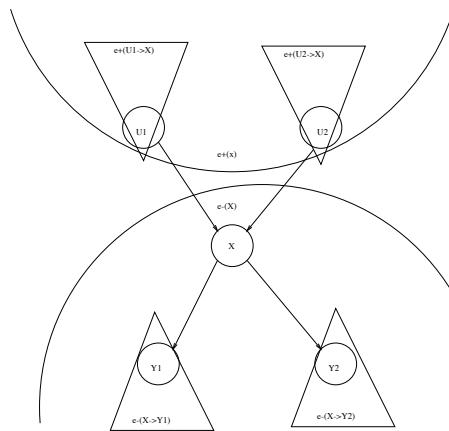


Figure 9.1: Message passing a polytree.

below the edge from  $X$  to  $Y_1$  (i.e., in the triangle below  $Y_1$ ). We use  $e_X$  to denote the local evidence attached to node  $X$  (if any).

We can compute  $\lambda_X$  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  $\lambda$  “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  $\pi_X$  by introducing  $X$ 's parents, to break the dependence on the upstream evidence, and then summing them out. We partition the evidence above  $X$  into the evidence in each subtree

1 above each parent  $U_i$ .

3  $\Pr(X = x|e_X^+) = \sum_{u1, u2} \Pr(X = x, U1 = u1, U2 = u2|e_X^+)$  (9.9)

4

5  $= \sum_{u1, u2} \Pr(X = x|u1, u2) \Pr(u1, u2|e_{U1 \rightarrow X}^+, e_{U2 \rightarrow X}^+)$  (9.10)

6

7  $= \sum_{u1, u2} \Pr(X = x|u1, u2) \Pr(u1|e_{U1 \rightarrow X}^+) \Pr(u2|e_{U2 \rightarrow X}^+)$  (9.11)

8 If we define the  $\pi$  “message” that a node  $X$  sends to its children  $Y_j$  as

9

10  $\Pi_{X \rightarrow Y_j}(x) \triangleq \Pr(X = x|e_{X \rightarrow Y_j}^+)$  (9.12)

11 we can write in general that

12

13  $\pi_X(x) = \sum_u P(X = x|u) \prod_i \Pi_{U_i \rightarrow X}(u_i)$  (9.13)

14 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  $\lambda$  message.

22

23  $\lambda_{X \rightarrow U1}(u1) = \Pr(e_X^-, e_{U2 \rightarrow X}^+|u1)$  (9.14)

24 all the ev. except in the U1 triangle (9.15)

25

26  $= \sum_x \sum_{u2} \Pr(e_X^-, e_{U2 \rightarrow X}^+|u1, u2, x) \Pr(u2, x|u1)$  (9.16)

27

28  $= \sum_x \sum_{u2} \Pr(e_X^-|x) \Pr(e_{U2 \rightarrow X}^+|u2) \Pr(u2, x|u1)$  (9.17)

29 since  $X$  separates the U2 triangle from  $e_X^-$ , and  $U2$  separates the U2 triangle from U1 (9.18)

30

31  $= c \sum_x \sum_{u2} \Pr(e_X^-|x) \frac{\Pr(u2|e_{U2 \rightarrow X}^+)}{\Pr(u2)} \Pr(x|u2, u1) \Pr(u2|u1)$  (9.19)

32 using Bayes’ rule, where  $c = \Pr(e_{U2 \rightarrow X}^+)$  is a constant (9.20)

33

34  $= c \sum_x \sum_{u2} \Pr(e_X^-|x) \Pr(u2|e_{U2 \rightarrow X}^+) \Pr(x|u2, u1)$  (9.21)

35 since U1 and U2 are marginally independent (9.22)

36

37  $= c \sum_x \sum_{u2} \lambda_X(x) \Pi_{U2 \rightarrow X}(u2) \Pr(x|u2, u1)$  (9.23)

38 In general, we have

39

40  $\lambda_{X \rightarrow U_i}(u_i) = c \sum_x \lambda_X(x) \left[ \sum_{u_k: k \neq i} P(X = x|u) \prod_{k \neq i} \Pi_{U_k \rightarrow X}(u_k) \right]$  (9.24)

If the graph is a rooted tree (as opposed to a polytree), each node has a unique parent, and this simplifies to

$$\lambda_{X \rightarrow U_i}(u_i) = c \sum_x \lambda_X(x) \Pr(X = x | u) \quad (9.25)$$

Finally, we derive the  $\pi$  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  $\pi$  messages; if we go against an arrow, the messages are  $\lambda$  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  $\lambda$  messages, and the second pass will only consist of sending  $\pi$  messages. This is analogous to a reversed version of the forwards-backwards algorithm, where we first send backwards likelihood messages to the root (node  $x_1$ ) and then send them forwards posterior messages to the end of the chain (node  $x_T$ ).

### 9.2.2 BP for undirected graphs with pairwise potentials

Suppose we have a directed tree in which each node (except for the root) has a single parent, so the joint distribution has the form

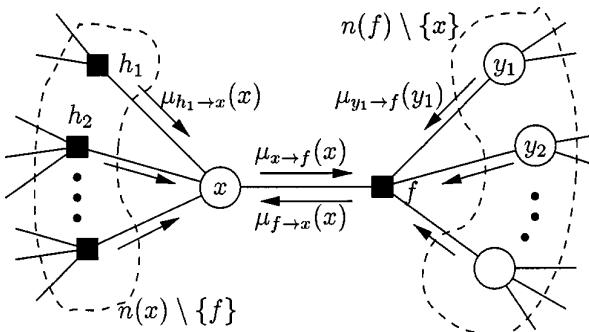
$$p(\mathbf{x}|\mathbf{v}) = \frac{1}{p(\mathbf{v})} \left[ p(x_r) \prod_{i \neq r} p(x_i | x_{\text{pa}(i)}) \right] \left[ \prod_i p(\mathbf{v}_i | x_i) \right] \quad (9.27)$$

We can write this in a more symmetric way by working with undirected graphs, in which each  $(s, t)$  edge has a corresponding pairwise potential  $\psi_{s,t}(x_s, x_t)$ , and each node has a local potential  $\psi_s(x_s)$ :

$$p(\mathbf{x}|\mathbf{v}) = \frac{1}{Z(\mathbf{v})} \prod_{s \in \mathcal{V}} \psi_s(x_s) \prod_{(s,t) \in \mathcal{E}} \psi_{s,t}(x_s, x_t) \quad (9.28)$$

We now describe a fully parallel message passing algorithm that can be applied to such pairwise undirected graphs, including those with cycles (loops). The basic idea is that all nodes receive messages from their neighbors in parallel, they then update their belief states, and finally they send new messages back out to their neighbors. This message passing process repeats until convergence. This kind of computing architecture is called a **systolic array**, due to its resemblance to a beating heart.

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



14 *Figure 9.2: Message passing on a bipartite factor graph. Square nodes represent factors, and circles represent*  
15 *variables. The  $y_i$  nodes correspond to the neighbors  $x'_i$  of  $f$  other than  $x$ . From Figure 6 of [KFL01]. Used*  
16 *with kind permission of Brendan Frey.*

17  
18

19 More precisely, we initialize all messages to the all 1's vector. Then, in parallel, each node absorbs  
20 messages from all its neighbors using

21

$$22 \quad \text{bel}_s(x_s) \propto \psi_s(x_s) \prod_{t \in \text{nbr}_s} m_{t \rightarrow s}(x_s) \quad (9.29)$$

23

24 Then, in parallel, each node sends messages to each of its neighbors:

25

$$26 \quad m_{s \rightarrow t}(x_t) = \sum_{x_s} \left( \psi_s(x_s) \psi_{st}(x_s, x_t) \prod_{u \in \text{nbr}_s \setminus t} m_{u \rightarrow s}(x_s) \right) \quad (9.30)$$

27

28 The  $m_{s \rightarrow t}$  message is computed by multiplying together all incoming messages, except the one sent by  
30 the recipient, and then passing through the  $\psi_{st}$  potential. (Note that we no longer need to distinguish  
31  $\lambda$  messages from  $\pi$  messages, since all messages are just a product of factors that will be normalized  
32 at the end to get a proper probability.)

33 We continue this process until convergence. If the graph is a tree, the method is guaranteed  
34 to converge after  $D(G)$  iterations, where  $D(G)$  is the **diameter** of the graph, that is, the largest  
35 distance between any two nodes. But for loopy graphs, the method may not converge at all, and  
36 even if it does, it is not clear if the resulting belief states are accurate. We discuss these issues below.

37

### 38 9.2.3 BP for factor graphs

39 To handle models with higher-order clique potentials (beyond pairwise), it is useful to use a factor  
40 graph representation described in Section 4.4.5. In this section, we describe the BP equations for  
41 bipartite factor graphs, as proposed in [KFL01].<sup>1</sup> For a version that works for Forney factor graphs,  
42 see [Loe+07].

44 1. For an efficient JAX implementation of BP for factor graphs (including graphs with loops, as discussed in Section 9.3),  
45 see <https://github.com/vicariousinc/PGMax>. For some C++ code, that handles discrete and continuous nodes, see  
46 <https://github.com/borglab/gtsam>.

47

In the case of bipartite factor graphs, we have two kinds of messages: variables to factors

$$m_{x \rightarrow f}(x) = \prod_{h \in \text{nbr}(x) \setminus \{f\}} m_{h \rightarrow x}(x) \quad (9.31)$$

and factors to variables:

$$m_{f \rightarrow x}(x) = \sum_{\mathbf{x}'} f(x, \mathbf{x}') \prod_{x' \in \text{nbr}(f) \setminus \{x\}} m_{x' \rightarrow f}(x') \quad (9.32)$$

Here  $\text{nbr}(x)$  are all the factors that are connected to variable  $x$ , and  $\text{nbr}(f)$  are all the variables that are connected to factor  $f$ . These messages are illustrated in Figure 9.2. At convergence, we can compute the final beliefs as a product of incoming messages:

$$\text{bel}(x) \propto \prod_{f \in \text{nbr}(x)} m_{f \rightarrow x}(x) \quad (9.33)$$

#### 9.2.4 Max product belief propagation

So far we have considered **sum-product belief propagation**. We can replace the sum operation with the max operation to get **max-product belief propagation**. The result of this computation (whether serial or parallel) is that we compute the **max marginals** for each node:

$$\zeta_i(k) = \max_{\mathbf{x}_{-i}} p(x_i = k, \mathbf{x}_{-i} | \mathbf{v}) \quad (9.34)$$

(By replacing  $p(\mathbf{x}|\mathbf{v})$  with  $-\log p(\mathbf{x}|\mathbf{v})$ , we can replace max-product with min-sum; this will yield the same result.) By contrast, sum-product computes the posterior marginals:

$$\gamma_i(k) = \sum_{\mathbf{x}_{-i}} p(x_i = k, \mathbf{x}_{-i} | \mathbf{v}) \quad (9.35)$$

We can derive two different kinds of “MAP” estimates from these local quantities. Let  $\hat{x}_i = \operatorname{argmax}_k \gamma_i(k)$ ; this is known as the **maximizer of the posterior marginal** or **MPM** estimate (see e.g., [MMP87; SM12]); let  $\hat{\mathbf{x}} = [\hat{x}_1, \dots, \hat{x}_V]$  be the sequence of such estimates.

Now consider  $\tilde{x}_i = \operatorname{argmax}_k \zeta_i(k)$ ; we call this the **maximizer of the max marginal** or **MMM** estimate; let  $\tilde{\mathbf{x}} = [\tilde{x}_1, \dots, \tilde{x}_V]$ .

An interesting question is: what, if anything, do these estimates have to do with the “true” MAP estimate,  $\mathbf{x}^* = \operatorname{argmax}_{\mathbf{x}} p(\mathbf{x}|\mathbf{v})$ ? We discuss this below.

##### 9.2.4.1 Connection between MMM and MAP

In [YW04], they showed that, if the max marginals are unique and computed exactly (e.g., if the graph is a tree), then  $\tilde{\mathbf{x}} = \mathbf{x}^*$ . This means we can recover the global MAP estimate by running max product BP and then setting each node to its local max (i.e., using the MMM estimate).

However, if there are ties in the max marginals (corresponding to the case where there is more than one globally optimal solution), this “local stitching” process may result in global inconsistencies.

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

19 To illustrate why this could be a problem, consider the error correcting code example from  
20 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  
21 binary. The priors  $p(x_1)$  and  $p(x_2)$  are uniform. The conditional term  $p(x_3|x_1, x_2)$  is deterministic,  
22 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  
23 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  
24 flipping noisy channel model, with noise level  $\alpha = 0.2$ .

25 Suppose we observe  $y = (1, 0, 0)$ . In this case, the exact posterior marginals are as follows:<sup>2</sup>  $\gamma_1 =$   
26  $[0.3469, 0.6531]$ ,  $\gamma_2 = [0.6531, 0.3469]$ ,  $\gamma_3 = [0.6531, 0.3469]$ . The exact max marginals are all the same,  
27 namely  $\zeta_i = [0.3265, 0.3265]$ . Finally, the 3 global MAP estimates are  $\mathbf{x}^* \in \{[0, 0, 0], [1, 1, 0], [1, 0, 1]\}$ ,  
28 each of which corresponds to a single bit flip from the observed vector. The MAP estimates are all  
29 valid code words (they have an even number of 1s), and hence are sensible hypotheses about the  
30 value of  $\mathbf{x}$ . By contrast, the MPM estimate is  $\hat{\mathbf{x}} = [1, 0, 0]$ , which is not a legal codeword. (And in  
31 this example, the MMM estimate is not well defined, since the max marginals are not unique.)

32 So, which method is better? This depends on our loss function, as we discuss in Section 3.8. If  
33 we want to minimize the prediction error of each  $x_i$ , also called **bit error**, we should compute the  
34 MPM. If we want to minimize the prediction error for the entire sequence  $\mathbf{x}$ , also called **word error**,  
35 we should use MAP, since this can take global constraints into account.

36 For example, suppose we are performing speech recognition and someone says “recognize speech”.  
37 MPM decoding may return “wreck a nice beach”, since locally it may be that “beach” is the most  
38 probable interpretation of “speech” when viewed in isolation. However, MAP decoding would infer  
39 that “recognize speech” is the more likely overall interpretation, by taking into account the language  
40 model prior,  $p(\mathbf{x})$ .

41 On the other hand, if we don’t have strong constraints, the MPM estimate can be more robust  
42 [MMP87; SM12], since it marginalizes out the other nodes, rather than maxing them out. For  
43 example, in the casino HMM example in Figure 8.4, we see that the MPM method makes 49 bit  
44 errors (out of a total possible of  $T = 300$ ), and the MAP path makes 60 errors.  
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.

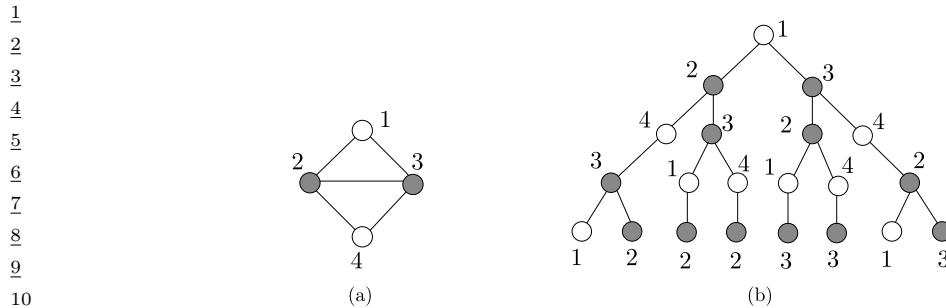
To perform message passing in models with non-Gaussian potentials, one approach is to extend the idea behind unscented Kalman filtering (Section 8.6.1), which is like a deterministic sampling method that uses  $2K + 1$  samples of the form  $\boldsymbol{\mu}$  and  $\boldsymbol{\mu} \pm \mathbf{e}_k \sigma_k$ , where  $\sigma_k$  is the standard deviation of the  $k$ 'th dimension and  $\mathbf{e}_k$  is a unit vector; the resulting method is called **sigma point BP** [MHH14].

Another approach to is to use sampling methods to approximate the relevant integrals (cf. particle filtering in Section 13.2); this is called **non-parametric BP** or **particle BP** (see e.g., [Sud+03; Isa03; Sud+10; Pac+14]).

## 9.3 Loopy belief propagation

In this section, we extend belief propagation to work on graphs with cycles or loops; this is called **loopy belief propagation** or **LBP**. Unfortunately, this method may not converge, and even if it does, it is not clear if the resulting estimates are valid. Indeed, Judea Pearl, who invented belief propagation for trees, wrote the following about loopy BP in 1988:

When loops are present, the network is no longer singly connected and local propagation schemes will invariably run into trouble ... If we ignore the existence of loops and permit the nodes to continue communicating with each other as if the network were singly connected, messages may circulate indefinitely around the loops and the process may not converge to a stable equilibrium ... Such oscillations do not normally occur in probabilistic networks ... which tend to bring all messages to some stable equilibrium as time goes on. However, this asymptotic equilibrium is not coherent, in the sense that it does not represent the posterior probabilities of all nodes of the network — [Pea88, p.195]



11 *Figure 9.3: (a) A simple loopy graph. (b) The computation tree, rooted at node 1, after 4 rounds of message*  
 12 *passing. Nodes 2 and 3 occur more often in the tree because they have higher degree than nodes 1 and 2.*  
 13 *From Figure 8.2 of [WJ08]. Used with kind permission of Martin Wainwright.*

14  
 15  
 16 Despite these reservations, Pearl advocated the use of belief propagation in loopy networks as an  
 17 approximation scheme (J. Pearl, personal communication). [MWJ99] found empirically that it works  
 18 on various graphical models, and it is now used in many real world applications, some of which we  
 19 discuss below. In addition, there is now some theory justifying its use in certain cases, as we discuss  
 20 below.  
 21

### 22 9.3.1 Convergence

23 Loopy BP may not converge, or may only converge slowly. In this section, we discuss some techniques  
 24 that increase the chances of convergence, and the speed of convergence.  
 25

#### 26 9.3.1.1 When will LBP converge?

27 The details of the analysis of when LBP will converge are beyond the scope of this chapter, but  
 28 we briefly sketch the basic idea. The key analysis tool is the **computation tree**, which visualizes  
 29 the messages that are passed as the algorithm proceeds. Figure 9.3 gives a simple example. In the  
 30 first iteration, node 1 receives messages from nodes 2 and 3. In the second iteration, it receives one  
 31 message from node 3 (via node 2), one from node 2 (via node 3), and two messages from node 4 (via  
 32 nodes 2 and 3). And so on.  
 33

34 The key insight is that  $T$  iterations of LBP is equivalent to exact computation in a computation  
 35 tree of height  $T + 1$ . If the strengths of the connections on the edges is sufficiently weak, then the  
 36 influence of the leaves on the root will diminish over time, and convergence will occur. See [MK05;  
 37 WJ08] and references therein for more information.  
 38

#### 39 9.3.1.2 Making LBP converge

40 Although the theoretical convergence analysis is very interesting, in practice, when faced with a  
 41 model where LBP is not converging, what should we do?  
 42

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

$$46 \quad m_{t \rightarrow s}^k(x_s) = \lambda m_{t \rightarrow s}(x_s) + (1 - \lambda) m_{t \rightarrow s}^{k-1}(x_s) \quad (9.37)$$

47

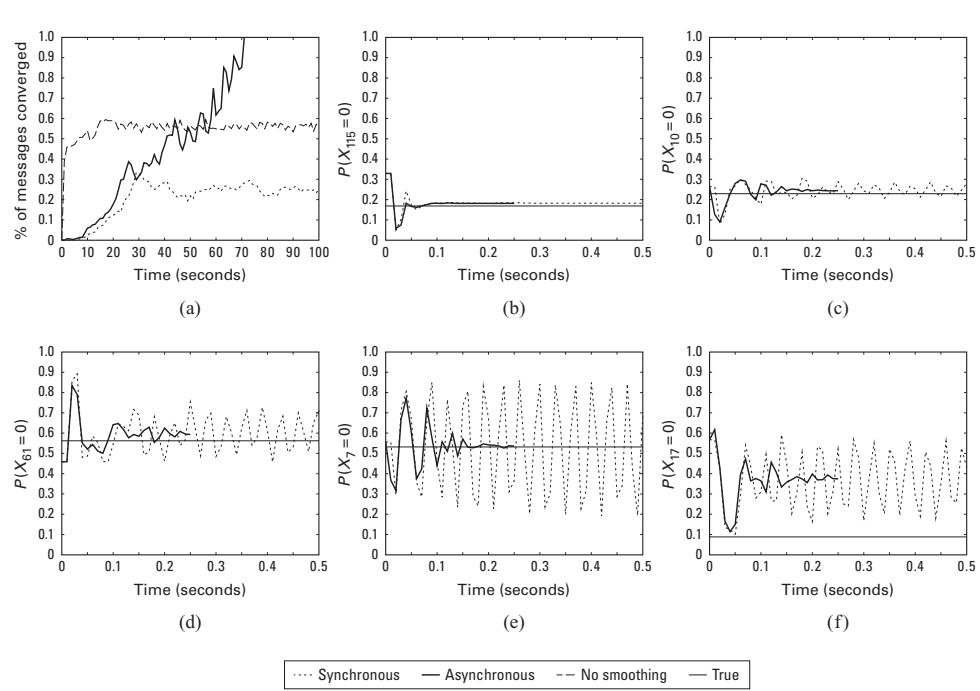


Figure 9.4: Illustration of the behavior of loopy belief propagation on an  $11 \times 11$  Ising grid with random potentials,  $w_{ij} \sim \text{Unif}(-C, C)$ , where  $C = 11$ . For larger  $C$ , inference becomes harder. (a) Percentage of messages that have converged vs time for 3 different update schedules: Dotted = damped synchronous (few nodes converge), dashed = undamped asynchronous (half the nodes converge), solid = damped asynchronous (all nodes converge). (b-f) Marginal beliefs of certain nodes vs time. Solid straight line = truth, dashed = synchronous, solid = damped asynchronous. From Figure 11.C.1 of [KF09a]. Used with kind permission of Daphne Koller.

where  $m_{t \rightarrow s}(x_s)$  is the standard undamped message, where  $0 \leq \lambda \leq 1$  is the damping factor. Clearly if  $\lambda = 1$  this reduces to the standard scheme, but for  $\lambda < 1$ , this partial updating scheme can help improve convergence. Using a value such as  $\lambda \sim 0.5$  is standard practice. The benefits of this approach are shown in Figure 9.4, where we see that damped updating results in convergence much more often than undamped updating (see [ZLG20] for some analysis of the benefits of damping).

It is possible to devise methods, known as **double loop algorithms**, which are guaranteed to converge to a local minimum of the same objective that LBP is minimizing [Yui01; WT01]. Unfortunately, these methods are rather slow and complicated, and the accuracy of the resulting marginals is usually not much greater than with standard LBP. (Indeed, oscillating marginals is sometimes a sign that the LBP approximation itself is a poor one.) Consequently, these techniques are not very widely used.

---

### 9.3.1.3 Increasing the convergence rate with adaptive scheduling

The standard approach when implementing LBP is to perform **synchronous updates**, where all nodes absorb messages in parallel, and then send out messages in parallel. That is, the new messages at iteration  $k + 1$  are computed in parallel using

$$\mathbf{m}^{k+1} = (f_1(\mathbf{m}^k), \dots, f_E(\mathbf{m}^k)) \quad (9.38)$$

where  $E$  is the number of edges, and  $f_{st}(\mathbf{m})$  is the function that computes the message for edge  $s \rightarrow t$  given all the old messages. This is analogous to the Jacobi method for solving linear systems of equations.

It is well known [Ber97] that the Gauss-Seidel method, which performs **asynchronous updates** in a fixed round-robin fashion, converges faster when solving linear systems of equations. We can apply the same idea to LBP, using updates of the form

$$\mathbf{m}_i^{k+1} = f_i(\{\mathbf{m}_j^{k+1} : j < i\}, \{\mathbf{m}_j^k : j > i\}) \quad (9.39)$$

where the message for edge  $i$  is computed using new messages (iteration  $k + 1$ ) from edges earlier in the ordering, and using old messages (iteration  $k$ ) from edges later in the ordering.

This raises the question of what order to update the messages in. One simple idea is to use a fixed or random order. The benefits of this approach are shown in Figure 9.4, where we see that (damped) asynchronous updating results in convergence much more often than synchronous updating.

However, we can do even better by using an adaptive ordering. The intuition is that we should focus our computational efforts on those variables that are most uncertain. [EMK06] proposed a technique known as **residual belief propagation**, in which messages are scheduled to be sent according to the norm of the difference from their previous value. That is, we define the residual of new message  $m_{s \rightarrow t}$  at iteration  $k$  to be

$$r(s, t, k) = \|\log m_{s \rightarrow t} - \log m_{s \rightarrow t}^k\|_\infty = \max_i |\log \frac{m_{s \rightarrow t}(i)}{m_{s \rightarrow t}^k(i)}| \quad (9.40)$$

We can store messages in a priority queue, and always send the one with highest residual. When a message is sent from  $s$  to  $t$ , all of the other messages that depend on  $m_{s \rightarrow t}$  (i.e., messages of the form  $m_{t \rightarrow u}$  where  $u \in \text{nbr}(t) \setminus s$ ) need to be recomputed; their residual is recomputed, and they are added back to the queue. In [EMK06], they showed (experimentally) that this method converges more often, and much faster, than using synchronous updating, asynchronous updating with a fixed order, and the TRP approach.

A refinement of residual BP was presented in [SM07]. In this paper, they use an upper bound on the residual of a message instead of the actual residual. This means that messages are only computed if they are going to be sent; they are not just computed for the purposes of evaluating the residual. This was observed to be about five times faster than residual BP, although the quality of the final results is similar.

41

### 9.3.2 Accuracy

For a graph with a single loop, one can show that the max-product version of LBP will find the correct MAP estimate, if it converges [Wei00]. For more general graphs, one can bound the error in the approximate marginals computed by LBP, as shown in [WJW03; IFW05; Vin+10].

47

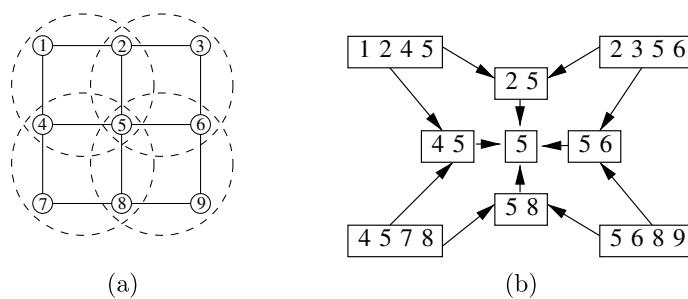


Figure 9.5: (a) Clusters superimposed on a  $3 \times 3$  lattice graph. (b) Corresponding hyper-graph. Nodes represent clusters, and edges represent set containment. From Figure 4.5 of [WJ08]. Used with kind permission of Martin Wainwright.

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.

The diagram illustrates a hierarchical tree structure, likely representing a neural network architecture. The structure is organized into several concentric layers, each containing nodes represented by black squares. These nodes are interconnected by lines, forming a complex web of connections. The layers appear to be fully connected to their immediate neighbors, creating a dense network of interactions across the different levels of the hierarchy.

(a)

(b)

<sup>14</sup> Figure 9.6: (a) A simple factor graph representation of a  $(2,3)$  low-density parity check code. Each message  
<sup>15</sup> bit (hollow round circle) is connected to two parity factors (solid black squares), and each parity factor is  
<sup>16</sup> 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  
<sup>17</sup> the xor operator. The local evidence factors for each hidden node are not shown. (b) A larger example of a  
<sup>18</sup> random LDPC code. We see that this graph is “locally tree-like”, meaning there are no short cycles; rather,  
<sup>19</sup> each cycle has length  $\sim \log m$ , where  $m$  is the number of nodes. This gives us a hint as to why loopy BP  
<sup>20</sup> works so well on such graphs. (Note, however, that some error correcting code graphs have short loops, so this  
<sup>21</sup> is not the full explanation.) From Figure 2.9 from [WJ08]. Used with kind permission of Martin Wainwright.

<sup>24</sup> However, the main impetus behind the interest in LBP arose when McEliece, MacKay, and Cheng  
<sup>25</sup> [MMC98] showed that a popular algorithm for error correcting codes, known as **turbocodes** [BGT93]  
<sup>26</sup>, 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

$$\psi_{stu}(x_s, x_t, x_u) \triangleq \begin{cases} 1 & \text{if } x_s \otimes x_t \otimes x_u = 1 \\ 0 & \text{otherwise} \end{cases} \quad (9.41)$$

<sup>41</sup> If the degrees of the parity checks and variable nodes remain bounded as the blocklength  $m$  increases,  
<sup>42</sup> this is called a **low-density parity check code**, or **LDPC code**. (Turbo codes are constructed in  
<sup>43</sup> 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

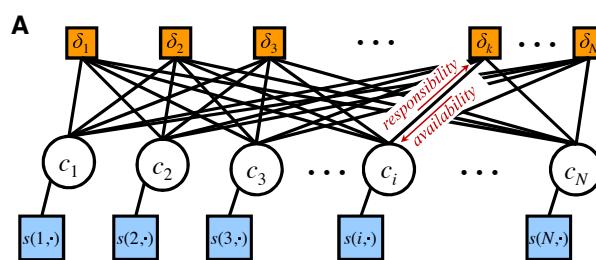


Figure 9.7: Factor graphs for affinity propagation. Circles are variables, squares are factors. Each  $c_i$  node has  $N$  possible states. From Figure S2 of [FD07a]. Used with kind permission of Brendan Frey.

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

$$S(\mathbf{c}) = \sum_{i=1}^N s(i, c_i) + \sum_{k=1}^N \delta_k(\mathbf{c}) \quad (9.42)$$

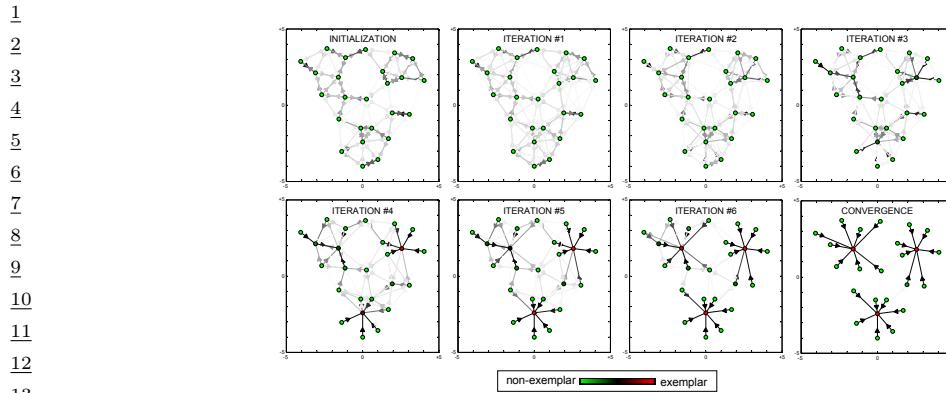
The first term measures the similarity of each point to its centroid. The second term is a penalty term that is  $-\infty$  if some data point  $i$  has chosen  $k$  as its exemplar (i.e.,  $c_i = k$ ), but  $k$  has not chosen itself as an exemplar (i.e., we do not have  $c_k = k$ ). More formally,

$$\delta_k(\mathbf{c}) = \begin{cases} -\infty & \text{if } c_k \neq k \text{ but } \exists i : c_i = k \\ 0 & \text{otherwise} \end{cases} \quad (9.43)$$

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  $\delta_k$ . It turns out that this vector of  $N$  numbers can be reduced to a scalar message,



*Figure 9.8: Example of affinity propagation. Each point is colored coded by how much it wants to be an exemplar (red is the most, green is the least). This can be computed by summing up all the incoming availability messages and the self-similarity term. The darkness of the  $i \rightarrow k$  arrow reflects how much point  $i$  wants to belong to exemplar  $k$ . From Figure 1 of [FD07a]. Used with kind permission of Brendan Frey.*

denoted  $r_{i \rightarrow k}$ , known as the responsibility. This is a measure of how much  $i$  thinks  $k$  would make a good exemplar, compared to all the other exemplars  $i$  has looked at. In addition, each factor node  $\delta_k$  sends a message to each variable node  $c_i$ . Again this can be reduced to a scalar message,  $a_{i \leftarrow k}$ , known as the availability. This is a measure of how strongly  $k$  believes it should be an exemplar for  $i$ , based on all the other data points  $k$  has looked at.

As usual with loopy BP, the method might oscillate, and convergence is not guaranteed. However, by using damping, the method is very reliable in practice. If the graph is densely connected, message passing takes  $O(N^2)$  time, but with sparse similarity matrices, it only takes  $O(E)$  time, where  $E$  is the number of edges or non-zero entries in  $\mathbf{S}$ .

The number of clusters can be controlled by scaling the diagonal terms  $S(i, i)$ , which reflect how much each data point wants to be an exemplar. Figure 9.8 gives a simple example of some 2d data, where the negative Euclidean distance was used to measured similarity. The  $S(i, i)$  values were set to be the median of all the pairwise similarities. The result is 3 clusters. Many other results are reported in [FD07a], who show that the method significantly outperforms K-medoids.

### 9.3.7 Emulating BP with graph neural nets

There is a close connection between message passing in PGMs and message passing in graph neural networks (GNNs), which we discuss in Section 16.3.5. However, for PGMs, the message computations are computing using (non-learned) update equations that work for any model; all that is needed is the graph structure  $G$ , model parameters  $\theta$ , and evidence  $v$ . By contrast, GNNs are trained to emulate specific functions using labeled input-output pairs.

It is natural to wonder what happens if we train a GNN on the exact posterior marginals derived from a small PGM, and then apply that trained GNN to a different test PGM. In [Yoo+18; Zha+19c], they show this method can work quite well if the test PGM is similar in structure to the one used for training.

47

An alternative approach is to start with a known PGM, and then “unroll” the BP message passing algorithm to produce a layered feedforward model, whose connectivity is derived from the graph. The resulting network can then be trained discriminatively for some end-task (not necessarily computing posterior marginals). Thus the BP procedure applied to the PGM just provides a way to design the neural network structure. This method is called **deep unfolding** (see e.g., [HLRW14]), and can often give very good results. (See also [SW20] for a more recent version of this approach, called “**neural enhanced BP**”.)

These neural methods are useful if the PGM is fixed, and we want to repeatedly perform inference or prediction with it, using different values of the evidence, but where the set of nodes which are observed is always the same. This is an example of amortized inference, where we train a model to emulate the results of running an iterative optimization scheme (see Section 10.3.7 for more discussion).

## 9.4 The variable elimination (VE) algorithm

In this section, we discuss an algorithm to compute a posterior marginal  $p(\mathbf{x}_Q|\mathbf{v})$  for any query set  $Q$ , assuming  $p$  is defined by a graphical model. Unlike loopy BP, it is guaranteed to give the correct answers even if the graph has cycles. We assume all the hidden nodes are discrete, although a version of the algorithm can be created for the Gaussian case by using the rules for sum and product defined in Section 2.3.7.

### 9.4.1 Derivation of the algorithm

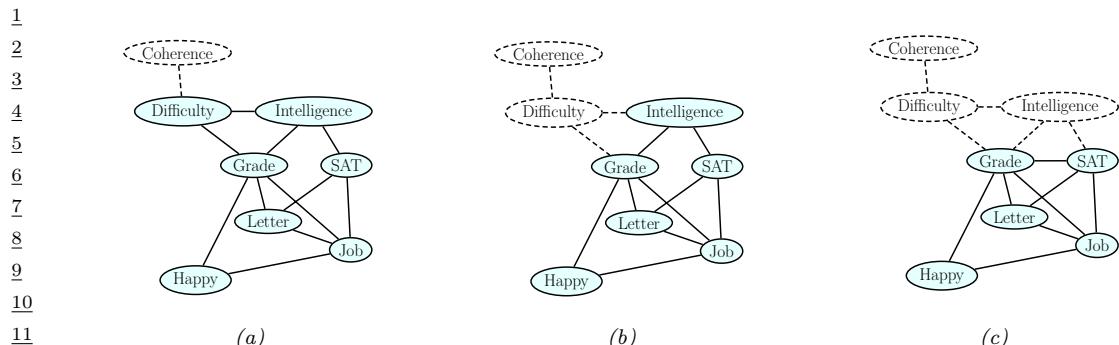
We will explain the algorithm by applying it to an example. Specifically, we consider the student network from Section 4.2.2.2. Suppose we want to compute  $p(J=1)$ , the marginal probability that a person will get a job. Since we have 8 binary variables, we could simply enumerate over all possible assignments to all the variables (except for  $J$ ), adding up the probability of each joint instantiation:

$$p(J) = \sum_L \sum_S \sum_G \sum_H \sum_I \sum_D \sum_C p(C, D, I, G, S, L, J, H) \quad (9.44)$$

However, this would take  $O(2^7)$  time. We can be smarter by **pushing sums inside products**. This is the key idea behind the **variable elimination** algorithm [ZP96], also called **bucket elimination** [Dec96], or, in the context of genetic pedigree trees, the **peeling algorithm** [CTS78].

In our example, we get

$$\begin{aligned} p(J) &= \sum_{L,S,G,H,I,D,C} p(C, D, I, G, S, L, J, H) \\ &= \sum_{L,S,G,H,I,D,C} \psi_C(C) \psi_D(D, C) \psi_I(I) \psi_G(G, I, D) \psi_S(S, I) \psi_L(L, G) \\ &\quad \times \psi_J(J, L, S) \psi_H(H, G, J) \\ &= \sum_{L,S} \psi_J(J, L, S) \sum_G \psi_L(L, G) \sum_H \psi_H(H, G, J) \sum_I \psi_S(S, I) \psi_I(I) \\ &\quad \times \sum_D \psi_G(G, I, D) \sum_C \psi_C(C) \psi_D(D, C) \end{aligned}$$



13 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].

<sup>17</sup> We now evaluate this expression, working right to left as shown in Table 9.1. First we multiply <sup>18</sup> together all the terms in the scope of the  $\sum_C$  operator to create the temporary factor

(9.45)

<sup>21</sup> Then we marginalize out  $C$  to get the new factor

$$\tau_1(D) = \sum_C \tau'_1(C, D) \quad (9.46)$$

25 Next we multiply together all the terms in the scope of the  $\sum_D$  operator and then marginalize out  
26 to create

$$\tau'(G, L, D) = \psi_G(G, L, D)\tau_*(D) \quad (8.47)$$

$$\underline{29} \quad \tau_2(G, I) = \sum \tau'_2(G, I, D) \quad (9.48)$$

32  $\Delta_{\text{mod}} \approx 0$

<sup>33</sup> The above technique can be used to compute any marginal of interest, such as  $p(J)$  or  $p(J, H)$ . To  
<sup>34</sup> compute a conditional, we can take a ratio of two marginals, where the visible variables have been  
<sup>35</sup> clamped to their known values (and hence don't need to be summed over). For example,

$$p(J = j | I = 1, H = 0) = \frac{p(J = j, I = 1, H = 0)}{\sum_{j'} p(J = j', I = 1, H = 0)} \quad (9.49)$$

## $\frac{40}{40}$ 9.4.2 Computational complexity of VE

<sup>42</sup> The running time of VE is clearly exponential in the size of the largest factor, since we have to sum  
<sup>43</sup> over all of the corresponding variables. Some of the factors come from the original model (and are  
<sup>44</sup> thus unavoidable), but new factors may also be created in the process of summing out. For example,  
<sup>45</sup> in Table 9.1, we created a factor involving G, I and S; but these nodes were not originally present  
<sup>46</sup> together in any factor.

---


$$\begin{aligned}
& \sum_L \sum_S \psi_J(J, L, S) \sum_G \psi_L(L, G) \sum_H \psi_H(H, G, J) \sum_I \psi_S(S, I) \psi_I(I) \sum_D \psi_G(G, I, D) \underbrace{\sum_C \psi_C(C) \psi_D(D, C)}_{\tau_1(D)} \\
& \sum_L \sum_S \psi_J(J, L, S) \sum_G \psi_L(L, G) \sum_H \psi_H(H, G, J) \sum_I \psi_S(S, I) \psi_I(I) \underbrace{\sum_D \psi_G(G, I, D) \tau_1(D)}_{\tau_2(G, I)} \\
& \sum_L \sum_S \psi_J(J, L, S) \sum_G \psi_L(L, G) \sum_H \psi_H(H, G, J) \underbrace{\sum_I \psi_S(S, I) \psi_I(I) \tau_2(G, I)}_{\tau_3(G, S)} \\
& \sum_L \sum_S \psi_J(J, L, S) \sum_G \psi_L(L, G) \underbrace{\sum_H \psi_H(H, G, J) \tau_3(G, S)}_{\tau_4(G, J)} \\
& \sum_L \sum_S \psi_J(J, L, S) \underbrace{\sum_G \psi_L(L, G) \tau_4(G, J) \tau_3(G, S)}_{\tau_5(J, L, S)} \\
& \sum_L \underbrace{\sum_S \psi_J(J, L, S) \tau_5(J, L, S)}_{\tau_6(J, L)} \\
& \sum_L \underbrace{\tau_6(J, L)}_{\tau_7(J)}
\end{aligned}$$

---

Table 9.1: Eliminating variables from Figure 4.28 in the order  $C, D, I, H, G, S, L$  to compute  $P(J)$ .

The order in which we perform the summation is known as the **elimination order**. This can have a large impact on the size of the intermediate factors that are created. For example, consider the ordering in Table 9.1: the largest created factor (beyond the original ones in the model) has size 3, corresponding to  $\tau_5(J, L, S)$ . Now consider the ordering in Table 9.2: now the largest factors are  $\tau_1(I, D, L, J, H)$  and  $\tau_2(D, L, S, J, H)$ , which are much bigger.

We can determine the size of the largest factor graphically, without worrying about the actual numerical values of the factors, by running the VE algorithm “symbolically”. When we eliminate a variable  $X_t$ , we connect together all variables that share a factor with  $X_t$  (to reflect the new temporary factor  $\tau'_t$ ). 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

$$\sum_D \sum_C \psi_D(D, C) \sum_H \sum_L \sum_S \psi_J(J, L, S) \sum_I \underbrace{\psi_I(I) \psi_S(S, I) \sum_G \psi_G(G, I, D) \psi_L(L, G) \psi_H(H, G, J)}_{\tau_1(I, D, L, J, H)}$$

2

$$\sum_D \sum_C \psi_D(D, C) \sum_H \sum_L \sum_S \psi_J(J, L, S) \underbrace{\sum_I \psi_I(I) \psi_S(S, I) \tau_1(I, D, L, J, H)}_{\tau_2(D, L, S, J, H)}$$

3

$$\sum_D \sum_C \psi_D(D, C) \sum_H \sum_L \underbrace{\sum_S \psi_J(J, L, S) \tau_2(D, L, S, J, H)}_{\tau_3(D, L, J, H)}$$

4

$$\sum_D \sum_C \psi_D(D, C) \sum_H \sum_L \underbrace{\tau_3(D, L, J, H)}_{\tau_4(D, J, H)}$$

5

$$\sum_D \sum_C \psi_D(D, C) \sum_H \underbrace{\sum_L \tau_4(D, L, J, H)}_{\tau_5(D, J)}$$

6

$$\sum_D \sum_C \psi_D(D, C) \underbrace{\sum_H \tau_5(D, J)}_{\tau_6(D, J)}$$

7

$$\sum_D \sum_C \psi_D(D, C) \underbrace{\sum_H \tau_6(D, J)}_{\tau_7(J)}$$

8

---

Table 9.2: Eliminating variables from Figure 4.28 in the order  $G, I, S, L, H, C, D$ .

It is clear that the time complexity of VE is

$$\sum_{c \in \mathcal{C}(G_\prec)} K^{|c|} \tag{9.52}$$

where  $\mathcal{C}(G)$  are the (maximal) cliques in graph  $G$ ,  $|c|$  is the size of the clique  $c$ , and we assume for notational simplicity that all the variables have  $K$  states each.

Let us define the **induced width** of a graph given elimination ordering  $\prec$ , denoted  $w_\prec$ , as the size of the largest factor (i.e., the largest clique in the induced graph) minus 1. Then it is easy to see that the complexity of VE with ordering  $\prec$  is  $O(K^{w_\prec+1})$ . The smallest possible induced width for a graph is known at its **treewidth** (see Section 9.4.2). Unfortunately finding the corresponding optimal elimination order is an NP-complete problem [Yan81; ACP87]. See Section 9.5.1.4 for a discussion of some approximate methods for finding good elimination orders.

41

### 42 9.4.3 Computational complexity of exact inference

43

We have seen that variable elimination takes  $O(NK^{w+1})$  time to compute the marginals for a graph with  $N$  nodes, and treewidth  $w$ , where each variable has  $K$  states. If the graph is densely connected, then  $w = O(N)$ , and so inference will take time exponential in  $N$ .

47

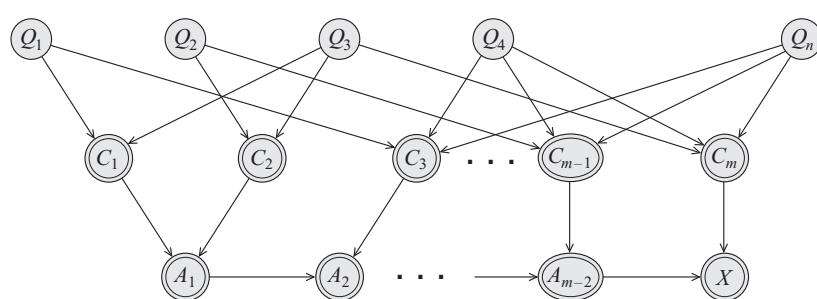


Figure 9.10: Encoding a 3-SAT problem on  $n$  variables and  $m$  clauses as a DGM. The  $Q_s$  variables are binary random variables. The  $C_t$  variables are deterministic functions of the  $Q_s$ 's, and compute the truth value of each clause. The  $A_t$  nodes are a chain of AND gates, to ensure 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

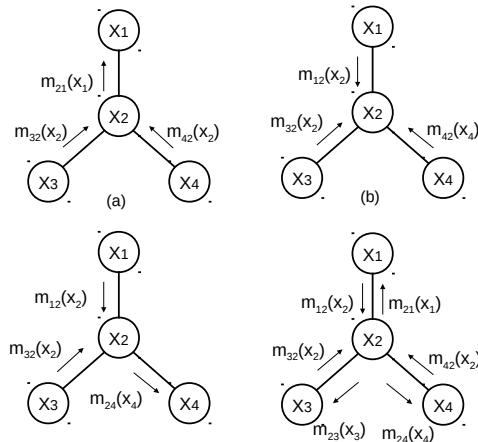


Figure 9.11: Sending multiple messages along a tree. (a)  $X_1$  is root. (b)  $X_2$  is root. (c)  $X_4$  is root. (d) All of the messages needed to compute all singleton marginals. Adapted from Figure 4.3 of [Jor07].

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  $\tau$  factors created by VE). Similarly we can compute  $p(x_2|\mathbf{v})$ ,  $p(x_3|\mathbf{v})$  and then  $p(x_4|\mathbf{v})$ . We see that some of the messages used to compute the marginal on one node can be re-used to compute the marginals on the other nodes. By storing the messages for later re-use, we can compute all the marginals in  $O(K^2T)$  time, as we show in Section 9.2.

The question is: how do we get these benefits of message passing on a tree when the graph is not a tree? We give the answer in Section 9.5.

## 9.5 The junction tree algorithm (JTA)

The **junction tree algorithm** or **JTA** is a generalization of variable elimination that lets us efficiently compute all the posterior marginals without repeating redundant work, thus avoiding the problems mentioned in Section 9.4.4. The basic idea is to convert the graph into a tree, and then to run belief propagation, as we briefly describe below. For more details, see e.g., [Lau96; Cow+99; KF09a].

### 9.5.1 Creating a junction tree

A **junction tree** is a tree-structured graph derived from the original graph, which satisfies certain properties, as we explain below. The process of converting a graph to a tree is If we have a general

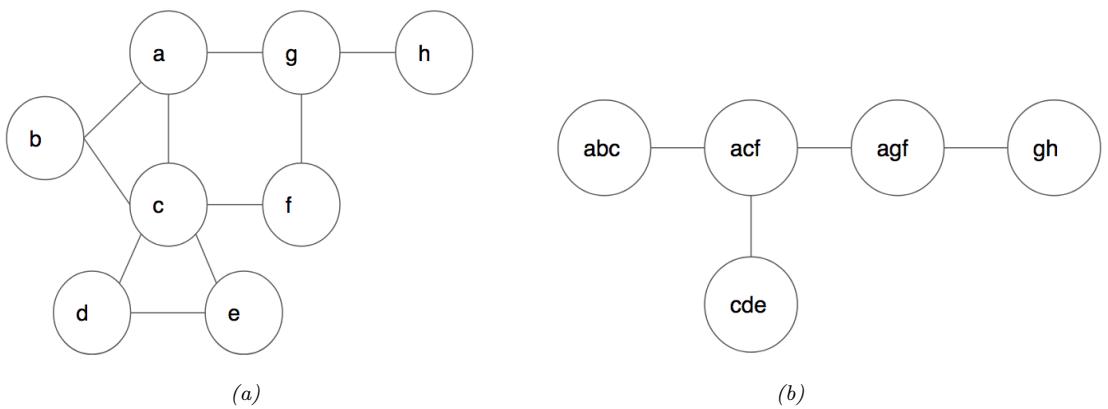


Figure 9.12: (a) An undirected graph. (b) Its corresponding junction tree.

graph, we can convert it into a tree using a process called **tree decomposition** [Hal76; RS84]. This process is briefly explained below. For more detail, see e.g., [Hei13; Sat15; Che14; VA15].

### 9.5.1.1 What is a tree decomposition?

Intuitively, we can convert a graph into a tree by grouping together nodes in the original graph to make “meganodes” until we end up with a tree, as illustrated in Figure 9.12. More formally, we say that  $T = (\mathcal{V}_T, \mathcal{E}_T)$  is a tree decomposition of an undirected graph  $G = (\mathcal{V}, \mathcal{E})$  if it satisfies the following properties:

- $\cup_{t \in \mathcal{V}_T} X_t = \mathcal{V}$ . Thus each graph vertex is associated with at least one tree node.
- For each edge  $(u, v) \in \mathcal{E}$  there exists a node  $t \in \mathcal{V}_T$  such that  $u \in X_t$  and  $v \in X_t$ . (For example, in Figure 9.12, we see that the edge  $a - b$  in  $G$  is contained in the meganode  $abc$  in  $T$ .)
- For each  $v \in \mathcal{V}$ , the set  $\{t : v \in X_t\}$  is a subtree of  $T$ . (For example, in Figure 9.12, we see that the set of meganodes in the tree containing graph node  $c$  forms the subtree  $(abc) - (acf) - (cde)$ .) Put another way, if  $X_i$  and  $X_j$  both contain a vertex  $v$ , then all the nodes  $X_k$  of the tree on the unique path from  $X_i$  to  $X_j$  also contain  $v$ , i.e., for any node  $X_k$  on the path from  $X_i$  to  $X_j$ , we have  $X_i \cap X_j \subseteq X_k$ . This is called the **running intersection property**. (For example, in Figure 9.12, if  $X_i = (abc)$  and  $X_j = (afg)$ , then we see that  $X_i \cap X_j = \{a\}$  is contained in node  $X_k = (acf)$ .)

A tree that satisfied these properties is also called a **junction tree** or **jtree**. The **width** of a jtree is defined to be the size of the largest meganode

$$\text{width}(T) = \max_{t \in T} |X_t| \quad (9.53)$$

For example, the width of the jtree in Figure 9.12(b) is 3.

There are many possible tree compositions of a graph, as we discuss below. We therefore define the **treewidth** of a graph  $G$  as the minimum width of any tree decomposition for  $G$  minus 1:

$$\text{treewidth}(G) \triangleq \left( \min_{T \in \mathcal{T}(G)} \text{width}(T) \right) - 1 \quad (9.54)$$

We see that the treewidth of a tree is 1, and the treewidth of the graph in Figure 9.12(a) is 2.

### 9.5.1.2 Why create a tree decomposition?

Before we discuss how to compute a tree decomposition, we pause and explain why we want to do this. The reason is that trees have a number of properties that make them useful for computational purposes. In particular, given a pair of nodes,  $u, v \in \mathcal{V}$ , we can always find a single node  $s \in \mathcal{V}$  on the path from  $u$  to  $v$  that is a **separator**, i.e., that partitions the graph into two subgraphs, one containing  $u$  and the other containing  $v$ . This is conducive to using algorithms based on dynamic programming, where we recursively solve the subproblems defined on the two subtrees, and then combine their solutions via the separator node  $s$ . This is useful for graphical model inference (see Section 9.5), solving sparse systems of linear equations (see e.g., [PL03]), etc.

### 9.5.1.3 Computing a tree decomposition

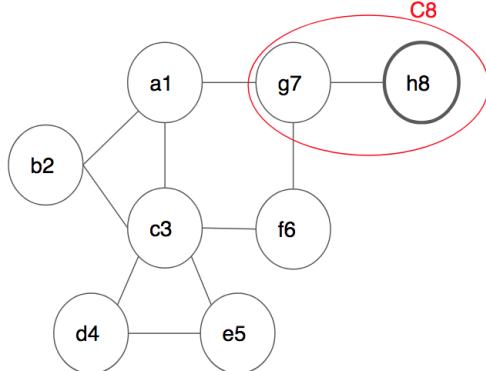
We now describe an algorithm known as **triangulation** or **elimination** for constructing a junction tree from an undirected graph. We first choose an ordering of the nodes,  $\pi$ . We then work backwards in this ordering, eliminating the nodes one at a time. We initially let  $\mathcal{U} = \{1, \dots, N\}$  be the set of all uneliminated nodes, and set the counter to  $i = N$ . At each step  $i$ , we pick node  $v_i = \pi_i$ , we create the set  $N_i = \text{nbr}_i \cap \mathcal{U}$  of uneliminated neighbors and the set  $C_i = v_i \cup N_i$ , we add **fill-in** edges between all nodes in  $C_i$  to make it a clique, we eliminate  $v_i$  by removing it from  $\mathcal{U}$ , and we decrement  $i$  by 1, until all nodes are eliminated.

We illustrate this method by applying it to the graph in Figure 9.13, using the ordering  $\pi = (a, b, c, d, e, f, g, h)$ . We initialize with  $i = 8$ , and start by eliminating  $v_i = \pi(8) = h$ , as shown in Figure 9.13(a). We create the set  $C_8 = \{g, h\}$  from node  $v_i$  and all its uneliminated neighbors. Then we add fill-in edges between them, if necessary. (In this case all the nodes in  $C_8$  are already connected.) In the next step, we eliminate  $v_i = \pi(7) = g$ , and create the clique  $C_7 = \{a, f, g\}$ , adding the fill-in edge  $a - f$ , as shown in Figure 9.13(b). We continue in this way until all nodes are eliminated, as shown in Figure 9.13(c).

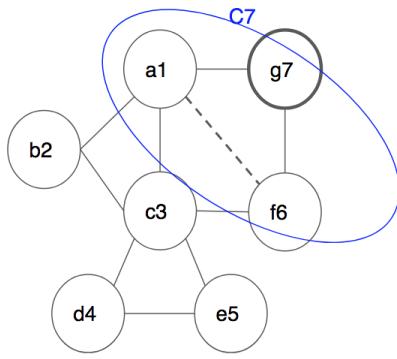
If we add the fill-in edges back to the original graph, the resulting graph will be **chordal**, which means that every undirected cycle  $X_1 - X_2 \cdots X_k - X_1$  of length  $k \geq 4$  has a chord. The largest loop in a chordal graph is length 3. Consequently chordal graphs are sometimes called **triangulated**.

Figure 9.13(d) illustrates the maximal cliques of the resulting chordal graph. In general, computing the maximal cliques of a graph is NP-hard, but in the case of a chordal graph, the process is easy: at step  $i$  of the elimination algorithm, we create clique  $C_i$  by connecting  $v_i$  to all its uneliminated neighbors; if this clique is contained in an already created clique, we simply discard it, otherwise we add it to our list of cliques. For example, when triangulating the graph in Figure 9.13, we drop clique  $C_4 = \{c, d\}$  since it is already contained in  $C_5 = \{c, d, e\}$ . Similarly we drop cliques  $C_2 = \{a, b\}$  and  $C_1 = \{a\}$ .

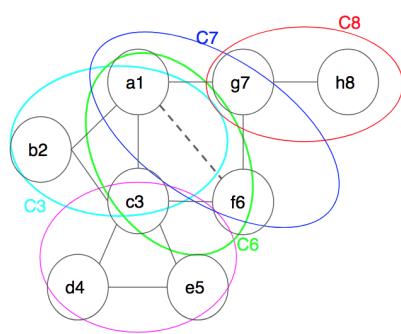
There are several ways to create a jtree from this set of cliques. One approach is as follows: create a **junction graph**, in which we add an edge between  $i$  and  $j$  if  $C_i \cap C_j \neq \emptyset$ . We set the weight of



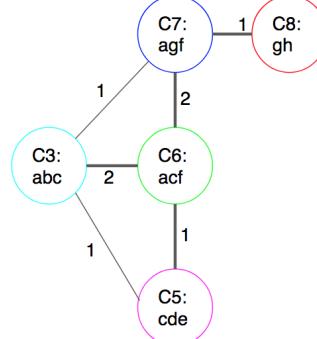
(a)



(b)



(c)



(d)

Figure 9.13: (a-b) Illustration of two steps of graph triangulation using the elimination order  $(a, b, c, d, e, f, g, h)$  applied to the graph in Figure 9.12a. The node being eliminated is shown with a darker border. Cliques are numbered by the vertex that created them. The dotted a-f line is a fill-in edge created when node g is eliminated. (c) Corresponding set of maximal cliques of the chordal graph. (d) Resulting junction graph.

this edge to be  $|C_i \cap C_j|$ , i.e., the number of variables they have in common. One can show [JJ94; AM00] that any maximal weight spanning tree (MST) of the junction graph is a junction tree. This is illustrated in Figure 9.13d, which corresponds to the jtree in Figure 9.12b.

#### 9.5.1.4 Picking a good elimination order

Many algorithms take time (or space) which is exponential in the width of the jtree, so we would like to find an elimination ordering that minimizes the width. We say that an ordering  $\pi$  is a **perfect elimination ordering** if it does not introduce any fill-in edges. Every graph that is already triangulated (e.g., a tree) has a perfect elimination ordering. We call such graphs **decomposable**.

In general, we will need to add fill-in edges to ensure the resulting graph is decomposable. Different orderings can introduce different numbers of fill-in edges, which affects the width of the resulting

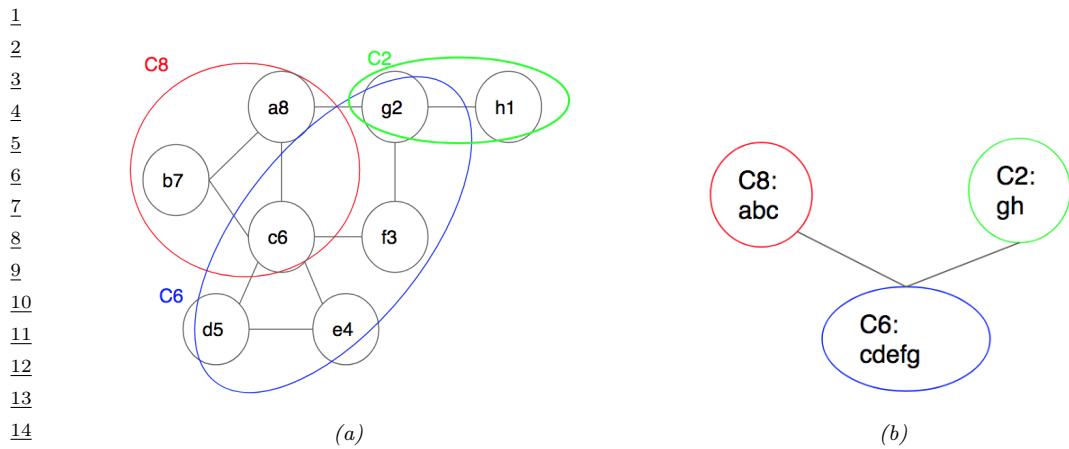


Figure 9.14: (a) Maximal cliques induced by using the elimination order  $(h, g, f, e, d, c, b, a)$ . The fill-in edges for clique  $C_6$  are not shown, to reduce clutter. (b) Corresponding junction graph/tree. The largest clique has size 5, which is much larger than in Figure 9.13d.

chordal graph. For example, Figure 9.13d uses the elimination ordering  $\pi = (a, b, c, d, e, f, g, h)$ , and Figure 9.14 illustrates what happens if we use the elimination ordering  $\pi = (h, g, f, e, d, c, b, a)$ . The width of the resulting chordal graph increases from 3 to 5.

Choosing an elimination ordering with minimal width is NP-complete [Yan81; ACP87]. It is common to use greedy approximation known as the **min-fill heuristic**, which works as follows: eliminate any node which would not result in any fill-ins (i.e., all of whose uneliminated neighbors already form a clique); if there is no such node, eliminate the node which would result in the minimum number of fill-in edges. When nodes have different weights (e.g., representing different numbers of states), we can use the **min-weight heuristic**, where we try to minimize the weight of the created cliques at each step.

Of course, many other methods are possible. [Kja90; Kja92] compared simulated annealing with the above greedy method, and found that it sometimes works better (although it is much slower). [MJ97] approximate the discrete optimization problem by a continuous optimization problem. [BG96] present a randomized approximation algorithm. [Gil88] present the nested dissection order, which is always within  $O(\log N)$  of optimal. [Ami01] discuss various constant-factor approximation algorithms. [Dav+04] present the approximate minimum degree ordering algorithm, which is implemented in Matlab.<sup>3</sup>

38

### 39 9.5.1.5 Application to graphical models

40 In this section, we show how to create a junction tree from a PGM-D. For example, consider the  
 41 “student” network from Figure 4.28(a). We can “moralize” this (by connecting unmarried parents  
 42 with a common child, and then dropping all edge orientations), to get the undirected graph in  
 43

44 3. See the description of the symamd command at <https://bit.ly/31N6E2b>. (“sym” stands for symbolic, “amd” stands  
 45 approximated minimum degree.) This method is used to find good elimination orderings for Cholesky decompositions of  
 46 sparse matrices.

47

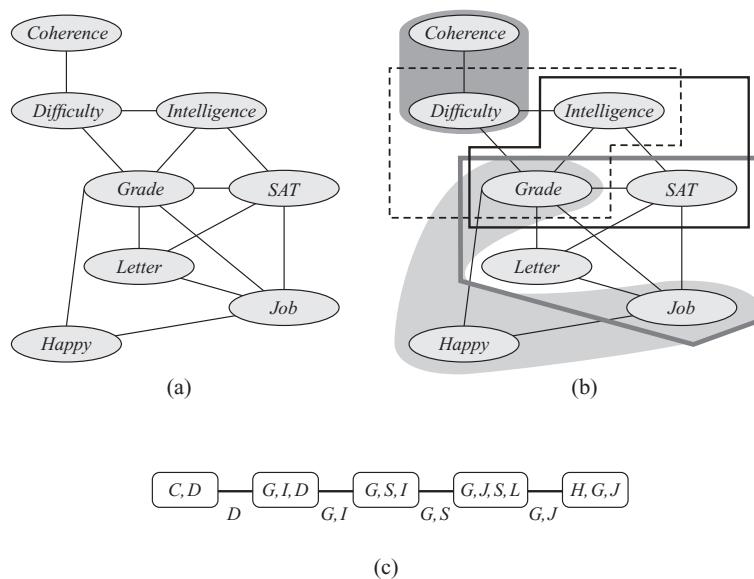


Figure 9.15: (a) A triangulated version of the (moralized) student graph from Figure 4.28(b). The extra fill-in edges (such as  $G-S$ ) are derived from the elimination ordering used in Figure 9.9. (b) The maximal cliques. (c) The junction tree. From Figure 9.11 of [KF09a]. Used with kind permission of Daphne Koller.

Figure 4.28(b). We can then derive a tree decomposition by applying the variable elimination algorithm from Section 9.4. The difference is that this time, we keep track of all the fill-in edges, and add them to the original graph, in order to make it chordal. We then extract the maximal cliques and convert them into a tree. The corresponding tree decomposition is illustrated in Figure 9.15. We see that the nodes of the jtree  $T$  are cliques of the chordal graph:

$$\mathcal{C}(T) = \{C, D\}, \{G, I, D\}, \{G, S, I\}, \{G, J, S, L\}, \{H, G, J\} \quad (9.55)$$

### 9.5.2 Running belief propagation on a junction tree

Once we have a junction tree, we can apply belief propagation to it in the usual way. More precisely, there are two versions: the sum-product form, also known as the **Shafer-Shenoy** algorithm, named after [SS90]; and the belief-updating form (which involves division), also known as the **Lauritzen-Spiegelhalter** algorithm, named after [LS88]. See [LS98] for a detailed comparison of these methods, and [SHJ97] for a worked example of applying the Lauritzen-Spiegelhalter algorithm to an HMM (which is equivalent to the forwards-backwards algorithm).

	Domain	+	$\times$	Name
2	$[0, \infty)$	$(+, 0)$	$(\times, 1)$	sum-product
3	$[0, \infty)$	$(\max, 0)$	$(\times, 1)$	max-product
4	$(-\infty, \infty]$	$(\min, \infty)$	$(+, 0)$	min-sum
5	$\{T, F\}$	$(\vee, F)$	$(\wedge, T)$	Boolean satisfiability

Table 9.3: Some commutative semirings.

### 9.5.3 The generalized distributive law

The key idea behind message passing is to note that sums distribute over products. For example, consider an HMM with 4 hidden states. We can write the partition function as follows:

$$Z = \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \psi_{12}(x_1, x_2) \psi_{23}(x_2, x_3) \psi_{34}(x_3, x_4) \quad (9.56)$$

$$= \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{23}(x_2, x_3) \sum_{x_4} \psi_{34}(x_3, x_4) \quad (9.57)$$

Then we can work backwards, summing out  $x_4$ , then  $x_3$ , etc.

For a chain of length  $T$ , where each node has  $K$  states, the version in Equation (9.57) (which is implemented by the forwards algorithm) takes  $O(TK^2)$  time, whereas the naive version in Equation (9.56) takes  $O(K^T)$  time. For the Gaussian case, we can use the Kalman filter to compute  $Z$  in  $O(TK^3)$  time (where  $K^3$  is the time to invert a  $K \times K$  covariance matrix), whereas naively marginalizing the corresponding joint Gaussian would take  $O((TK)^3)$  time.

We can apply this method to many other kinds of problems, provided the local clique functions  $\psi_c$  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  $\psi_c$  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  $\psi_c$  is a Gaussian potential, so the resulting model is a Gaussian joint distribution. In both cases, any subpiece of the joint distribution is conjugate to any other subpiece, so the subpieces can be combined exactly, as required by message passing. Thus the relevant factors are closed under the operation of marginalization and combination. (A more general family that satisfies this property is the **mixture of truncated basis functions** representation from [Lan+12].) We call such models “**structured conjugate models**”, since the joint distribution can be decomposed into a product of conjugate pieces.

In many cases we may have a model that does not exactly satisfy the above requirements. However, we may be able to approximate the model locally in order to meet the requirements (e.g., by linearizing a nonlinear dependence in a Gaussian model). We will see some examples of this later in this chapter. Alternatively, we may be able to “clamp” some hidden variables to specific values (e.g., by sampling), such that the remaining factors satisfy the conjugacy requirements (conditional on the sample). Thus the message methods we discuss in this chapter can be used as subroutines inside the approximate inference methods we discuss in Chapter 10 and Chapter 11.

#### 9.5.4 Other applications of the JTA

We have seen how to use the JTA algorithm to compute posterior marginals in a graphical model. There are several possible generalizations of this algorithm, some of which we mention below. All of these exploit graph decomposition in some form or other. They only differ in terms of how they define/ compute messages and “beliefs”. The key requirement is that the operators which compute messages form a commutative semiring (see Section 9.5.3).

- Computing the MAP estimate. We just replace the sum-product with max-product in the collect phase, and use traceback in the distribute phase, as in the Viterbi algorithm (Section 8.3.6). See [Daw92] for details.
- Computing the N-most probable configurations [Nil98].
- Computing posterior samples. The collect pass is the same as usual, but in the distribute pass, we sample variables given the values higher up in the tree, thus generalizing forwards-filtering backwards-sampling for HMMs described in Section 8.3.7. See [Daw92] for details.
- Solving linear systems of the form  $\mathbf{Ax} = \mathbf{b}$  where  $\mathbf{A}$  is a sparse matrix [BP92; PL03; Bic09].
- Solving constraint satisfaction problems [Dec03].
- Solving logical reasoning problems [AM05].

## 9.6 Inference as backpropagation

In this section, we present an approach to (exact) inference which exploits the connection between graphical models and the exponential family (Section 2.5). For notational simplicity, we focus on undirected graphical models, where the joint can be represented as follows:

$$p(\mathbf{x}) = \frac{1}{Z} \prod_c \psi_c(\mathbf{x}_c) = \exp\left(\sum_c \boldsymbol{\eta}_c^\top \mathcal{T}(\mathbf{x}_c) - \log A(\boldsymbol{\eta})\right) = \exp(\boldsymbol{\eta}^\top \mathcal{T}(\mathbf{x}) - \log A(\boldsymbol{\eta})) \quad (9.59)$$

where  $\psi_c$  is the potential function for clique  $c$ ,  $\boldsymbol{\eta}_c$  are the natural parameters for clique  $c$ ,  $\mathcal{T}(\mathbf{x}_c)$  are the corresponding sufficient statistics, and  $A = \log Z$  is the log partition function.

We will consider pairwise models (with node and edge potentials), and discrete variables. The natural parameters are the node and edge log potentials,  $\boldsymbol{\eta} = (\{\eta_{s;j}\}, \{\eta_{s,t;j,k}\})$ , and the sufficient statistics are node and edge indicator functions,  $\mathcal{T}(\mathbf{x}) = (\{\mathbb{I}(x_s = j)\}, \{\mathbb{I}(x_s = j, x_t = k)\})$ . (Note: we use  $s, t \in \mathcal{V}$  to index nodes and  $j, k \in \mathcal{X}$  to index states.)

The mean of the sufficient statistics are given by

$$\boldsymbol{\mu} = \mathbb{E}[\mathcal{T}(\mathbf{x})] = (\{p(x_s = j)\}_s, \{p(x_s = j, x_t = k)\}_{s \neq t}) = (\{\mu_{s;j}\}_s, \{\mu_{s,t;j,k}\}_{s \neq t}) \quad (9.60)$$

The key result, from Equation (2.197), is that  $\boldsymbol{\mu} = \nabla_{\boldsymbol{\eta}} A(\boldsymbol{\eta})$ . Thus as long as we have a function that computes  $A(\boldsymbol{\eta}) = \log Z(\boldsymbol{\eta})$ , we can use automatic differentiation (Section 6.2) to compute gradients, and then we can extract the corresponding node marginals from the gradient vector. (If we have evidence (known values) on some of the variables, we simply “clamp” the corresponding entries to 0 or 1 in the node potentials.)

As a concrete example, consider a chain structured model  $x_1 - x_2 - x_3$ , where each node has  $K$  states. We can represent the node potentials as  $K \times 1$  tensors (table of numbers), and the edge potentials by  $K \times K$  tensors. The partition function is given by

$$Z(\boldsymbol{\psi}) = \sum_{x_1, x_2, x_3} \psi_1(x_1) \psi_2(x_2) \psi_3(x_3) \psi_{12}(x_1, x_2) \psi_{23}(x_2, x_3) \quad (9.61)$$

Let  $\boldsymbol{\eta} = \log(\boldsymbol{\psi})$  be the log potentials, and  $A(\boldsymbol{\eta}) = \log Z(\boldsymbol{\eta})$  be the log partition function. We can compute the single mode marginals  $\boldsymbol{\mu}_s = p(x_s = 1 : K)$  using  $\boldsymbol{\mu}_s = \nabla_{\boldsymbol{\eta}_s} A(\boldsymbol{\eta})$ , and the pairwise marginals  $\boldsymbol{\mu}_{s,t}(j, k) = p(x_s = j, x_t = k)$  using  $\boldsymbol{\mu}_{s,t} = \nabla_{\boldsymbol{\eta}_{s,t}} A(\boldsymbol{\eta})$ .

We can compute the partition function  $Z$  efficiently use numpy’s **einsum** function, which implements tensor contraction using Einstein summation notation. We label each dimension of the tensors by A, B and C, so einsum knows how to match things up. We then compute gradients using an auto-diff library.<sup>4</sup> The result is that inference can be done in two lines of Python code:

```
logZ_fun = lambda logpots: np.log(jnp.einsum('A,B,C,AB,BC',
                                              *[np.exp(lp) for lp in logpots]))
probs = grad(logZ_fun)(logpots)
```

To perform conditional inference, such as  $p(x_s = k | x_t = e)$ , we multiply in one-hot indicator vectors to clamp  $x_t$  to the value  $e$  so that the unnormalized joint only assigns non-zero probability to state combinations that are valid. We then sum over all values of the unclamped variables to get the constrained partition function  $Z_e$ . The gradients will now give us the marginals conditioned on the evidence [Dar03].

To handle real-valued observations, we can just write down the unnormalized log joint where we include factors for each piece of local evidence. For example, consider an HMM (Chapter 30) defining the following log joint:

$$\log p(\mathbf{z}, \mathbf{y}) = \log \text{Cat}(z_1 | \boldsymbol{\pi}) + \sum_{t=1}^{T-1} \log \text{Cat}(z_{t+1} | \mathbf{P}_{z_t,:}) + \sum_{t=1}^T \log p(\mathbf{y}_t | z_t) \quad (9.62)$$

<sup>4</sup> 4. We use JAX. See [ugm\\_inf\\_autodiff.py](#) for the full code, and see <https://github.com/srush/ProbTalk> for a PyTorch version by Sasha Rush.

1 Let  $p(z_1 = k) = \pi_k$ ,  $p(z_t = j | z_{t-1} = i) = P_{ij}$ , and  $p(\mathbf{y}_t | z_t = k) = p(\mathbf{y}_t | \boldsymbol{\theta}_k)$ . Then the log joint is  
2 given by  
3

$$\log p(\mathbf{z}, \mathbf{y}) = \exp \left[ \sum_{k=1}^K \underbrace{\mathbb{I}(z_1 = k)}_{\mathcal{T}_{1k}(\mathbf{z})} \underbrace{\log \pi_{1,k}}_{\eta_{1k}} + \sum_{t=1}^{T-1} \underbrace{\mathbb{I}(z_t = i, z_{t+1} = j)}_{\mathcal{T}_{tij}(\mathbf{z})} \underbrace{\log P_{ij}}_{\eta_{ij}} \right] \quad (9.63)$$

$$\sum_{t=1}^T \sum_{k=1}^K \underbrace{\mathbb{I}(z_t = k)}_{\mathcal{T}_{tk}(\mathbf{z})} \underbrace{\log p(\mathbf{y}_t | \boldsymbol{\theta}_k)}_{\eta_{tk}} - \underbrace{\log Z(\mathbf{y}_{1:T} | \boldsymbol{\theta})}_{A(\boldsymbol{\eta})} \quad (9.64)$$

12 We can use the forwards-backwards algorithm (Section 8.3.3) to compute the posterior marginals,  
13  $p(z_t = k | \mathbf{y})$ . Alternatively, we can use automatic differentiation to get  
14

$$\nabla_{\eta_{tk}} A(\boldsymbol{\eta}) = \mathbb{E}[\mathcal{T}_{tk}(\mathbf{z})] = p(z_t = k | \mathbf{y}) \quad (9.65)$$

17 Since numpy's einsum implementation searches for an optimal elimination ordering [GASG18], as  
18 discussed in Section 9.5.1.4, it will push sum inside products, and can thus compute the above  
19 quantities in  $O(K^2T)$  time, which is the same efficiency as the forwards-backwards algorithm.  
20 The same method can also be used to implement Kalman smoothing, although in that case, the  
21 computation of  $Z$  must be done using manually written code because it is not discrete summation  
22 (although recent efforts [Obe+19] are attempting to automate this).  
23

24 The observation that probabilistic inference can be performed using automatic differentiation  
25 has been discovered independently by several groups (e.g., [Dar03; PD03; Eis16; ASM17]). This  
26 significantly simplifies implementation, and graphical modeling inference libraries to leverage fast AD  
27 libraries, which are available in many deep learning systems. It also lends itself to the development  
28 of differentiable approximations to inference (see e.g., [MB18]).  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47



# 10 Variational inference

## 10.1 Introduction

In this chapter, we discuss **variational inference**, which reduces posterior inference to an optimization problem. Note that VI is a large topic. This chapter just gives a high level overview. For more details, see e.g., [Jor+98; JJ00; Jaa01; WJ08; Zha+19a; Bro18].

### 10.1.1 Variational free energy

Consider a model with unknown variables  $\mathbf{z}$ , known variables  $\mathbf{x}$ , and fixed parameters  $\boldsymbol{\theta}$ . (If the parameters are unknown, they can be added to  $\mathbf{z}$ .) Since computing the true posterior  $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$  is assumed intractable, we will use the approximation  $q(\mathbf{z})$ , which we choose to minimize the following loss:

$$q = \underset{q \in \mathcal{Q}}{\operatorname{argmin}} D_{\text{KL}}(q(\mathbf{z}) \| p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})) \quad (10.1)$$

Since we are minimizing over functions (namely distributions  $q$ ), this is called a **variational method**.

In practice we pick a parametric family  $\mathcal{Q}$ , where we use  $\boldsymbol{\psi}$  to represent the **variational parameters**. We compute the best variational parameters (for given  $\mathbf{x}$ ) as follows:

$$\boldsymbol{\psi}^* = \underset{\boldsymbol{\psi}}{\operatorname{argmin}} D_{\text{KL}}(q(\mathbf{z}|\boldsymbol{\psi}) \| p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})) \quad (10.2)$$

$$= \underset{\boldsymbol{\psi}}{\operatorname{argmin}} \mathbb{E}_{q(\mathbf{z}|\boldsymbol{\psi})} \left[ \log q(\mathbf{z}|\boldsymbol{\psi}) - \log \left( \frac{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{x})} \right) \right] \quad (10.3)$$

$$= \underset{\boldsymbol{\psi}}{\operatorname{argmin}} \underbrace{\mathbb{E}_{q(\mathbf{z}|\boldsymbol{\psi})} [\log q(\mathbf{z}|\boldsymbol{\psi}) - \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) - \log p_{\boldsymbol{\theta}}(\mathbf{z})]}_{\mathcal{L}(\boldsymbol{\psi}|\boldsymbol{\theta}, \mathbf{x})} + \log p_{\boldsymbol{\theta}}(\mathbf{x}) \quad (10.4)$$

The final term  $\log p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) d\mathbf{z}$  is generally intractable to compute. Fortunately, it is independent of  $\boldsymbol{\psi}$ , so we can drop it. This leaves us with the first term:<sup>1</sup>

$$\mathcal{L}(\boldsymbol{\psi}|\boldsymbol{\theta}, \mathbf{x}) = D_{\text{KL}}(q(\mathbf{z}|\boldsymbol{\psi}) \| p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})) \triangleq \mathbb{E}_{q(\mathbf{z}|\boldsymbol{\psi})} [\log q(\mathbf{z}|\boldsymbol{\psi}) - \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})] \quad (10.5)$$

---

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.