

Supplementary Material for
“Probabilistic Machine Learning: Advanced Topics”

Kevin Murphy

August 15, 2023

Contents

1	Introduction	5
I Fundamentals		7
2	Probability	9
2.1	More fun with MVNs	9
2.1.1	Inference in the presence of missing data	9
2.1.2	Sensor fusion with unknown measurement noise	9
2.2	Google's PageRank algorithm	12
2.2.1	Retrieving relevant pages using inverted indices	12
2.2.2	The PageRank score	13
2.2.3	Efficiently computing the PageRank vector	14
2.2.4	Web spam	15
2.2.5	Personalized PageRank	16
3	Bayesian statistics	17
3.1	Bayesian concept learning	17
3.1.1	Learning a discrete concept: the number game	17
3.1.2	Learning a continuous concept: the healthy levels game	23
3.2	Informative priors	26
3.2.1	Domain specific priors	27
3.2.2	Gaussian prior	28
3.2.3	Power-law prior	29
3.2.4	Erlang prior	29
3.3	Tweedie's formula (Empirical Bayes without estimating the prior)	30
4	Graphical models	33
4.1	More examples of DGMs	33
4.1.1	Water sprinkler	33
4.1.2	Asia network	34
4.1.3	The QMR network	35

1	4.1.4	Genetic linkage analysis	37
2	4.2	More examples of UGMs	40
3	4.3	Restricted Boltzmann machines (RBMs) in more detail	40
4	4.3.1	Binary RBMs	40
5	4.3.2	Categorical RBMs	41
6	4.3.3	Gaussian RBMs	41
7	4.3.4	RBMs with Gaussian hidden units	42
8	5	Information theory	43
9	5.1	Minimizing KL between two Gaussians	43
10	5.1.1	Moment projection	43
11	5.1.2	Information projection	43
12	6	Optimization	45
13	6.1	Proximal methods	45
14	6.1.1	Proximal operators	45
15	6.1.2	Computing proximal operators	48
16	6.1.3	Proximal point methods (PPM)	51
17	6.1.4	Mirror descent	53
18	6.1.5	Proximal gradient method	53
19	6.1.6	Alternating direction method of multipliers (ADMM)	55
20	6.2	Dynamic programming	57
21	6.2.1	Example: computing Fibonacci numbers	57
22	6.2.2	ML examples	58
23	6.3	Conjugate duality	58
24	6.3.1	Introduction	58
25	6.3.2	Example: exponential function	60
26	6.3.3	Conjugate of a conjugate	61
27	6.3.4	Bounds for the logistic (sigmoid) function	61
28	6.4	The Bayesian learning rule	63
29	6.4.1	Deriving inference algorithms from BLR	64
30	6.4.2	Deriving optimization algorithms from BLR	66
31	6.4.3	Variational optimization	70
32	II	Inference	71
33	7	Inference algorithms: an overview	73
34	8	Inference for state-space models	75
35	8.1	More Kalman filtering	75
36	8.1.1	Example: tracking an object with spiral dynamics	75
37	8.1.2	Derivation of RLS	75
38	8.1.3	Handling unknown observation noise	77
39	8.1.4	Predictive coding as Kalman filtering	78
40	8.2	More extended Kalman filtering	79

8.2.1	Derivation of the EKF	79
8.2.2	Example: Tracking a pendulum	80
8.3	Exponential-family EKF	81
8.3.1	Modeling assumptions	81
8.3.2	Algorithm	82
8.3.3	EEKF for training logistic regression	82
8.3.4	EEKF performs online natural gradient descent	83
9	Inference for graphical models	89
9.1	Belief propagation on trees	89
9.1.1	BP for polytrees	89
9.2	The junction tree algorithm (JTA)	92
9.2.1	Tree decompositions	92
9.2.2	Message passing on a junction tree	96
9.2.3	The generalized distributive law	99
9.2.4	JTA applied to a chain	100
9.2.5	JTA for general temporal graphical models	101
9.3	MAP estimation for discrete PGMs	103
9.3.1	Notation	103
9.3.2	The marginal polytope	104
9.3.3	Linear programming relaxation	105
9.3.4	Graphcuts	108
10	Variational inference	113
10.1	More Gaussian VI	113
10.1.1	Example: Full-rank vs diagonal GVI on 1d linear regression	113
10.1.2	Example: Full-rank vs rank-1 GVI for logistic regression	115
10.1.3	Structured (sparse) Gaussian VI	116
10.2	Online variational inference	118
10.2.1	FOO-VB	118
10.2.2	Bayesian gradient descent	119
10.3	Beyond mean field	120
10.3.1	Exploiting partial conjugacy	120
10.3.2	Structured mean for factorial HMMs	124
10.4	VI for graphical model inference	126
10.4.1	Exact inference as VI	126
10.4.2	Mean field VI	127
10.4.3	Loopy belief propagation as VI	128
10.4.4	Convex belief propagation	131
10.4.5	Tree-reweighted belief propagation	132
10.4.6	Other tractable versions of convex BP	133
11	Monte Carlo Inference	135
12	Markov Chain Monte Carlo (MCMC) inference	137

1	13 Sequential Monte Carlo (SMC) inference	139
2	13.1 More applications of particle filtering	139
3	13.1.1 1d pendulum model with outliers	139
4	13.1.2 Visual object tracking	139
5	13.1.3 Online parameter estimation	141
6	13.1.4 Monte Carlo robot localization	141
7	13.2 Particle MCMC methods	142
8	13.2.1 Particle Marginal Metropolis Hastings	143
9	13.2.2 Particle Independent Metropolis Hastings	143
10	13.2.3 Particle Gibbs	144
11		
12		
13		
14	III Prediction	145
15		
16	14 Predictive models: an overview	147
17		
18	15 Generalized linear models	149
19	15.1 Variational inference for logistic regression	149
20	15.1.1 Binary logistic regression	149
21	15.1.2 Multinomial logistic regression	151
22	15.2 Converting multinomial logistic regression to Poisson regression	155
23	15.2.1 Beta-binomial logistic regression	155
24	15.2.2 Poisson regression	156
25	15.2.3 GLMM (hierarchical Bayes) regression	157
26		
27	16 Deep neural networks	161
28	16.1 More canonical examples of neural networks	161
29	16.1.1 Transformers	161
30	16.1.2 Graph neural networks (GNNs)	163
31		
32	17 Bayesian neural networks	169
33	17.1 More details on EKF for training MLPs	169
34	17.1.1 Global EKF	169
35	17.1.2 Decoupled EKF	169
36	17.1.3 Mini-batch EKF	170
37		
38	18 Gaussian processes	171
39	18.1 Deep GPs	171
40	18.2 GPs and SSMs	176
41		
42	19 Beyond the iid assumption	177
43		
44	IV Generation	179
45		
46	20 Generative models: an overview	181
47		

<u>1</u>	21 Variational autoencoders	183
<u>2</u>	21.0.1 VAEs with missing data	183
<u>3</u>		
<u>4</u>	22 Auto-regressive models	187
<u>5</u>		
<u>6</u>	23 Normalizing flows	189
<u>7</u>		
<u>8</u>	24 Energy-based models	191
<u>9</u>		
<u>10</u>	25 Denoising diffusion models	193
<u>11</u>		
<u>12</u>	26 Generative adversarial networks	195
<u>13</u>		
<u>14</u>	V Discovery	197
<u>15</u>		
<u>16</u>	27 Discovery methods: an overview	199
<u>17</u>		
<u>18</u>	28 Latent factor models	201
<u>19</u>	28.1 Inference in topic models	201
<u>20</u>	28.1.1 Collapsed Gibbs sampling for LDA	201
<u>21</u>	28.1.2 Variational inference for LDA	203
<u>22</u>		
<u>23</u>	29 State-space models	207
<u>24</u>	29.1 Continuous time SSMs	207
<u>25</u>	29.1.1 Ordinary differential equations	207
<u>26</u>	29.1.2 Example: Noiseless 1d spring-mass system	208
<u>27</u>	29.1.3 Example: tracking a moving object in continuous time	209
<u>28</u>	29.1.4 Example: tracking a particle in 2d	212
<u>29</u>	29.2 Structured State Space Sequence model (S4)	212
<u>30</u>		
<u>31</u>	30 Graph learning	215
<u>32</u>	30.1 Latent variable models for graphs	215
<u>33</u>	30.1.1 Stochastic block model	215
<u>34</u>	30.1.2 Mixed membership stochastic block model	217
<u>35</u>	30.1.3 Infinite relational model	219
<u>36</u>	30.2 Learning tree structures	220
<u>37</u>	30.2.1 Chow-Liu algorithm	221
<u>38</u>	30.2.2 Finding the MAP forest	223
<u>39</u>	30.2.3 Mixtures of trees	223
<u>40</u>	30.3 Learning DAG structures	224
<u>41</u>	30.3.1 Faithfulness	224
<u>42</u>	30.3.2 Markov equivalence	225
<u>43</u>	30.3.3 Bayesian model selection: statistical foundations	226
<u>44</u>	30.3.4 Bayesian model selection: algorithms	229
<u>45</u>	30.3.5 Constraint-based approach	231
<u>46</u>	30.3.6 Methods based on sparse optimization	234
<u>47</u>		

1	30.3.7	Consistent estimators	234
2	30.3.8	Handling latent variables	235
3	30.4	Learning undirected graph structures	243
4	30.4.1	Dependency networks	243
5	30.4.2	Graphical lasso for GGMs	245
6	30.4.3	Graphical lasso for discrete MRFs/CRFs	246
7	30.4.4	Bayesian inference for undirected graph structures	247
8	30.5	Learning causal DAGs	249
9	30.5.1	Learning cause-effect pairs	249
10	30.5.2	Learning causal DAGs from interventional data	252
11	30.5.3	Learning from low-level inputs	253
12	31	Non-parametric Bayesian models	255
13	31.1	Dirichlet processes	255
14	31.1.1	Definition of a DP	255
15	31.1.2	Stick breaking construction of the DP	257
16	31.1.3	The Chinese restaurant process (CRP)	259
17	31.2	Dirichlet process mixture models	260
18	31.2.1	Model definition	260
19	31.2.2	Fitting using collapsed Gibbs sampling	262
20	31.2.3	Fitting using variational Bayes	265
21	31.2.4	Other fitting algorithms	266
22	31.2.5	Choosing the hyper-parameters	267
23	31.3	Generalizations of the Dirichlet process	267
24	31.3.1	Pitman-Yor process	267
25	31.3.2	Dependent random probability measures	268
26	31.4	The Indian buffet process and the beta process	271
27	31.5	Small-variance asymptotics	274
28	31.6	Completely random measures	277
29	31.7	Lévy processes	278
30	31.8	Point processes with repulsion and reinforcement	280
31	31.8.1	Poisson process	280
32	31.8.2	Renewal process	281
33	31.8.3	Hawkes process	282
34	31.8.4	Gibbs point process	284
35	31.8.5	Determinantal point process	285
36	32	Representation learning	289
37	33	Interpretability	291
38	VI	Decision making	293
39	34	Multi-step decision problems	295

<u>1</u>	
<u>2</u>	35 Reinforcement learning 297
<u>3</u>	
<u>4</u>	36 Causality 299
<u>5</u>	
<u>6</u>	
<u>7</u>	
<u>8</u>	
<u>9</u>	
<u>10</u>	
<u>11</u>	
<u>12</u>	
<u>13</u>	
<u>14</u>	
<u>15</u>	
<u>16</u>	
<u>17</u>	
<u>18</u>	
<u>19</u>	
<u>20</u>	
<u>21</u>	
<u>22</u>	
<u>23</u>	
<u>24</u>	
<u>25</u>	
<u>26</u>	
<u>27</u>	
<u>28</u>	
<u>29</u>	
<u>30</u>	
<u>31</u>	
<u>32</u>	
<u>33</u>	
<u>34</u>	
<u>35</u>	
<u>36</u>	
<u>37</u>	
<u>38</u>	
<u>39</u>	
<u>40</u>	
<u>41</u>	
<u>42</u>	
<u>43</u>	
<u>44</u>	
<u>45</u>	
<u>46</u>	
<u>47</u>	

1 Introduction

This book contains supplementary material for [book2]. Some sections have not been checked as carefully as the main book, so *caveat lector*.

PART I

Fundamentals

2 Probability

2.1 More fun with MVNs

2.1.1 Inference in the presence of 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 Main Equation (2.119) is as follows:

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z} | \boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z) \quad (2.1)$$

$$p\left(\begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} | \mathbf{z}\right) = \mathcal{N}\left(\begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} | \begin{pmatrix} \mathbf{W}_1 \\ \mathbf{W}_2 \end{pmatrix} \mathbf{z} + \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix}, \begin{pmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix}\right) \quad (2.2)$$

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

$$p(\mathbf{z} | \mathbf{y}_1) = \mathcal{N}(\mathbf{z} | \boldsymbol{\mu}_{z|1}, \boldsymbol{\Sigma}_{z|1}) \quad (2.3)$$

$$\boldsymbol{\Sigma}_{z|1}^{-1} = \boldsymbol{\Sigma}_z^{-1} + \mathbf{W}_1^T \boldsymbol{\Sigma}_{11}^{-1} \mathbf{W}_1 \quad (2.4)$$

$$\boldsymbol{\mu}_{z|1} = \boldsymbol{\Sigma}_{z|1} [\mathbf{W}_1^T \boldsymbol{\Sigma}_{11}^{-1} (\mathbf{y}_1 - \mathbf{b}_1) + \boldsymbol{\Sigma}_z^{-1} \boldsymbol{\mu}_z] \quad (2.5)$$

2.1.2 Sensor fusion with unknown measurement noise

In this section, we extend the sensor fusion results from Main Section 2.3.2.3 to the case where the precision of each measurement device is unknown. This turns out to yield a potentially multi-modal posterior, as we will see, which is quite different from the Gaussian case. Our presentation is based on [**Minka01GaussVar**].

For simplicity, we assume the latent quantity is scalar, $z \in \mathbb{R}$, and that we just have two measurement devices, x and y . However, we allow these to have different precisions, so the data generating mechanism has the form $x_n | z \sim \mathcal{N}(z, \lambda_x^{-1})$ and $y_n | z \sim \mathcal{N}(z, \lambda_y^{-1})$. We will use a non-informative prior for z , $p(z) \propto 1$, which we can emulate using an infinitely broad Gaussian, $p(z) = \mathcal{N}(z | m_0 = 0, \lambda_0^{-1} = \infty)$. So the unknown parameters are the two measurement precisions, $\boldsymbol{\theta} = (\lambda_x, \lambda_y)$.

Suppose we make 2 independent measurements with each device, which turn out to be

$$x_1 = 1.1, x_2 = 1.9, y_1 = 2.9, y_2 = 4.1 \quad (2.6)$$

If the parameters θ were known, then the posterior would be Gaussian:

$$p(z|\mathcal{D}, \lambda_x, \lambda_y) = \mathcal{N}(z|m_N, \lambda_N^{-1}) \quad (2.7)$$

$$\lambda_N = \lambda_0 + N_x \lambda_x + N_y \lambda_y \quad (2.8)$$

$$m_N = \frac{\lambda_x N_x \bar{x} + \lambda_y N_y \bar{y}}{N_x \lambda_x + N_y \lambda_y} \quad (2.9)$$

where $N_x = 2$ is the number of x measurements, $N_y = 2$ is the number of y measurements, $\bar{x} = \frac{1}{N_x} \sum_{n=1}^{N_x} x_n = 1.5$ and $\bar{y} = \frac{1}{N_y} \sum_{n=1}^{N_y} y_n = 3.5$. This result follows because the posterior precision is the sum of the measurement precisions, and the posterior mean is a weighted sum of the prior mean (which is 0) and the data means.

However, the measurement precisions are not known. A simple solution is to estimate them by maximum likelihood. The log-likelihood is given by

$$\ell(z, \lambda_x, \lambda_y) = \frac{N_x}{2} \log \lambda_x - \frac{\lambda_x}{2} \sum_n (x_n - z)^2 + \frac{N_y}{2} \log \lambda_y - \frac{\lambda_y}{2} \sum_n (y_n - z)^2 \quad (2.10)$$

The MLE is obtained by solving the following simultaneous equations:

$$\frac{\partial \ell}{\partial z} = \lambda_x N_x (\bar{x} - z) + \lambda_y N_y (\bar{y} - z) = 0 \quad (2.11)$$

$$\frac{\partial \ell}{\partial \lambda_x} = \frac{1}{\lambda_x} - \frac{1}{N_x} \sum_{n=1}^{N_x} (x_n - z)^2 = 0 \quad (2.12)$$

$$\frac{\partial \ell}{\partial \lambda_y} = \frac{1}{\lambda_y} - \frac{1}{N_y} \sum_{n=1}^{N_y} (y_n - z)^2 = 0 \quad (2.13)$$

This gives

$$\hat{z} = \frac{N_x \hat{\lambda}_x \bar{x} + N_y \hat{\lambda}_y \bar{y}}{N_x \hat{\lambda}_x + N_y \hat{\lambda}_y} \quad (2.14)$$

$$1/\hat{\lambda}_x = \frac{1}{N_x} \sum_n (x_n - \hat{z})^2 \quad (2.15)$$

$$1/\hat{\lambda}_y = \frac{1}{N_y} \sum_n (y_n - \hat{z})^2 \quad (2.16)$$

We notice that the MLE for z has the same form as the posterior mean, m_N .

We can solve these equations by fixed point iteration. Let us initialize by estimating $\lambda_x = 1/s_x^2$ and $\lambda_y = 1/s_y^2$, where $s_x^2 = \frac{1}{N_x} \sum_{n=1}^{N_x} (x_n - \bar{x})^2 = 0.16$ and $s_y^2 = \frac{1}{N_y} \sum_{n=1}^{N_y} (y_n - \bar{y})^2 = 0.36$. Using this, we get $\hat{z} = 2.1154$, so $p(z|\mathcal{D}, \hat{\lambda}_x, \hat{\lambda}_y) = \mathcal{N}(z|2.1154, 0.0554)$. If we now iterate, we converge to $\hat{\lambda}_x = 1/0.1662$, $\hat{\lambda}_y = 1/4.0509$, $p(z|\mathcal{D}, \hat{\lambda}_x, \hat{\lambda}_y) = \mathcal{N}(z|1.5788, 0.0798)$.

The plug-in approximation to the posterior is plotted in Figure 2.1(a). This weights each sensor according to its estimated precision. Since sensor y was estimated to be much less reliable than sensor x , we have $\mathbb{E}[z|\mathcal{D}, \hat{\lambda}_x, \hat{\lambda}_y] \approx \bar{x}$, so we effectively ignore the y sensor.

Now we will adopt a Bayesian approach and integrate out the unknown precisions, following Main Section 3.4.3.3. That is, we compute

$$p(z|\mathcal{D}) \propto p(z) \left[\int p(\mathcal{D}_x|z, \lambda_x) p(\lambda_x|z) d\lambda_x \right] \left[\int p(\mathcal{D}_y|z, \lambda_y) p(\lambda_y|z) d\lambda_y \right] \quad (2.17)$$

We will use uninformative Jeffrey priors (Main Section 3.5.2) $p(z) \propto 1$, $p(\lambda_x|z) \propto 1/\lambda_x$ and $p(\lambda_y|z) \propto 1/\lambda_y$. Since the x and y terms are symmetric, we will just focus on one of them. The key integral is

$$I = \int p(\mathcal{D}_x|z, \lambda_x) p(\lambda_x|z) d\lambda_x \quad (2.18)$$

$$\propto \int \lambda_x^{-1} \lambda_x^{N_x/2} \exp\left(-\frac{N_x}{2} \lambda_x (\bar{x} - z)^2 - \frac{N_x}{2} s_x^2 \lambda_x\right) d\lambda_x \quad (2.19)$$

Exploiting the fact that $N_x = 2$ this simplifies to

$$I = \int \lambda_x^{-1} \lambda_x^1 \exp(-\lambda_x[(\bar{x} - z)^2 + s_x^2]) d\lambda_x \quad (2.20)$$

We recognize this as proportional to the integral of an unnormalized Gamma density

$$\text{Ga}(\lambda|a, b) \propto \lambda^{a-1} e^{-\lambda b} \quad (2.21)$$

where $a = 1$ and $b = (\bar{x} - z)^2 + s_x^2$. Hence the integral is proportional to the normalizing constant of the Gamma distribution, $\Gamma(a)b^{-a}$, so we get

$$I \propto \int p(\mathcal{D}_x|z, \lambda_x) p(\lambda_x|z) d\lambda_x \propto ((\bar{x} - z)^2 + s_x^2)^{-1} \quad (2.22)$$

and the posterior becomes

$$p(z|\mathcal{D}) \propto \frac{1}{(\bar{x} - z)^2 + s_x^2} \frac{1}{(\bar{y} - z)^2 + s_y^2} \quad (2.23)$$

The exact posterior is plotted in Figure 2.1(b). We see that it has two modes, one near $\bar{x} = 1.5$ and one near $\bar{y} = 3.5$. These correspond to the beliefs that the x sensor is more reliable than the y one, and vice versa. The weight of the first mode is larger, since the data from the x sensor agree more with each other, so it seems slightly more likely that the x sensor is the reliable one. (They obviously cannot both be reliable, since they disagree on the values that they are reporting.) However, the Bayesian solution keeps open the possibility that the y sensor is the more reliable one; from two measurements, we cannot tell, and choosing just the x sensor, as the plug-in approximation does, results in overconfidence (a posterior that is too narrow).

So far, we have assumed the prior is conjugate to the likelihood, so we have been able to compute the posterior analytically. However, this is rarely the case. A common alternative is to approximate the integral using Monte Carlo sampling, as follows:

$$p(\mathbf{z}|\mathcal{D}) \propto \int p(\mathbf{z}|\mathcal{D}, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta} \quad (2.24)$$

$$\approx \frac{1}{S} \sum_s p(\mathbf{z}|\mathcal{D}, \boldsymbol{\theta}^s) \quad (2.25)$$

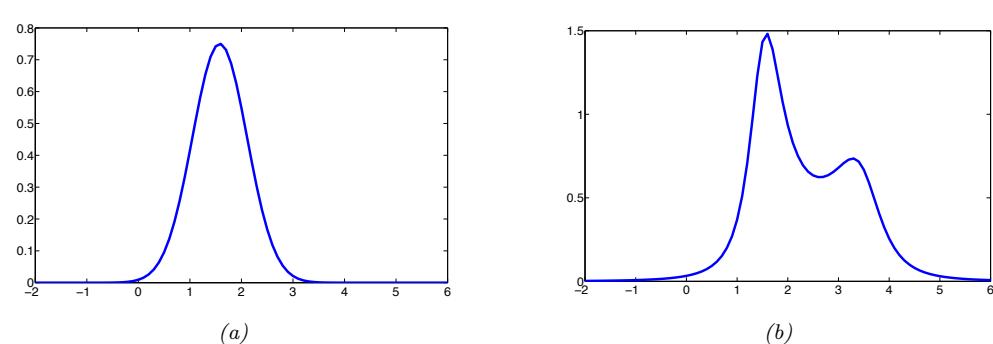


Figure 2.1: Posterior for z . (a) Plug-in approximation. (b) Exact posterior. Generated by [sensor_fusion_unknown_prec.ipynb](#).

where $\boldsymbol{\theta}^s \sim p(\boldsymbol{\theta}|\mathcal{D})$. Note that $p(z|\mathcal{D}, \boldsymbol{\theta}^s)$ is conditionally Gaussian, and is easy to compute. So we just need a way to draw samples from the parameter posterior, $p(\boldsymbol{\theta}|\mathcal{D})$. We discuss suitable methods for this in Main Chapter 11.

2.2 Google's PageRank algorithm

In this section, we discuss Google's **PageRank** algorithm, since it provides an interesting application of Markov chain theory. PageRanke is one of the components used for ranking web page search results. We sketch the basic idea below; see [Bryan06] for a more detailed explanation.

2.2.1 Retrieving relevant pages using inverted indices

We will treat the web as a giant directed graph, where nodes represent web pages (documents) and edges represent hyper-links.¹ We then perform a process called **web crawling**. We start at a few designated root nodes, such as [wikipedia.org](#), and then follows the links, storing all the pages that we encounter, until we run out of time.

Next, all of the words in each web page are entered into a data structure called an **inverted index**. That is, for each word, we store a list of the documents where this word occurs. At test time, when a user enters a query, we can find potentially relevant pages as follows: for each word in the query, look up all the documents containing each word, and intersect these lists. (We can get a more refined search by storing the location of each word in each document, and then testing if the words in a document occur in the same order as in the query.)

Let us give an example, from http://en.wikipedia.org/wiki/Inverted_index. Suppose we have 3 documents, D_0 = "it is what it is", D_1 = "what is it" and D_2 = "it is a banana". Then we can create the following inverted index, where each pair represents a document and word location:

```
42 "a":      {(2, 2)}
```

44 1. In 2008, Google said it had indexed 1 trillion (10^{12}) unique URLs. If we assume there are about 10 URLs per page
45 (on average), this means there were about 100 billion unique web pages. Estimates for 2010 are about 121 billion
46 unique web pages. Source: <https://bit.ly/2keQeyi>

```

1 "banana": {(2, 3)}
2 "is":      {(0, 1), (0, 4), (1, 1), (2, 1)}
3 "it":      {(0, 0), (0, 3), (1, 2), (2, 0)}
4 "what":    {(0, 2), (1, 0)}
5

```

For example, we see that the word “what” occurs in document 0 at location 2 (counting from 0), and in document 1 at location 0. Suppose we search for “what is it”. If we ignore word order, we retrieve the following documents:

$$10 \quad \{D_0, D_1\} \cap \{D_0, D_1, D_2\} \cap \{D_0, D_1, D_2\} = \{D_0, D_1\} \quad (2.26)$$

If we require that the word order matches, only document D_1 would be returned. More generally, we can allow out-of-order matches, but can give “bonus points” to documents whose word order matches the query’s word order, or to other features, such as if the words occur in the title of a document. We can then return the matching documents in decreasing order of their score/ relevance. This is called document **ranking**.

2.2.2 The PageRank score

So far, we have described the standard process of information retrieval. But the link structure of the web provides an additional source of information. The basic idea is that some web pages are more authoritative than others, so these should be ranked higher (assuming they match the query). A web page is considered an **authority** if it is linked to by many other pages. But to protect against the effect of so-called **link farms**, which are dummy pages which just link to a given site to boost its apparent relevance, we will weight each incoming link by the source’s authority. Thus we get the following recursive definition for the authoritativeness of page j , also called its **PageRank**:

$$27 \quad \pi_j = \sum_i A_{ij} \pi_i \quad (2.27)$$

where A_{ij} is the probability of following a link from i to j . (The term “PageRank” is named after Larry Page, one of Google’s co-founders.)

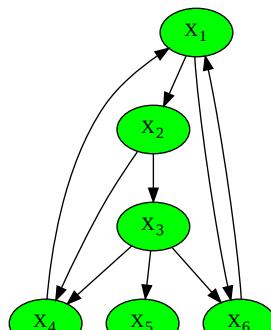
We recognize Equation (2.27) as the stationary distribution of a Markov chain. But how do we define the transition matrix? In the simplest setting, we define $A_{i,:}$ as a uniform distribution over all states that i is connected to. However, to ensure the distribution is unique, we need to make the chain into a regular chain. This can be done by allowing each state i to jump to any other state (including itself) with some small probability. This effectively makes the transition matrix aperiodic and fully connected (although the adjacency matrix G_{ij} of the web itself is highly sparse).

We discuss efficient methods for computing the leading eigenvector of this giant matrix below. Here we ignore computational issues, and just give some examples.

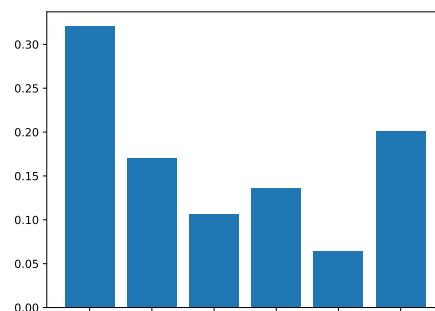
First, consider the small web in Figure 2.2. We find that the stationary distribution is

$$41 \quad \boldsymbol{\pi} = (0.3209, 0.1706, 0.1065, 0.1368, 0.0643, 0.2008) \quad (2.28)$$

43 So a random surfer will visit site 1 about 32% of the time. We see that node 1 has a higher PageRank
44 than nodes 4 or 6, even though they all have the same number of in-links. This is because being
45 linked to from an influential node helps increase your PageRank score more than being linked to by
46 a less influential node.

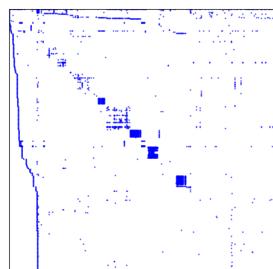


(a)

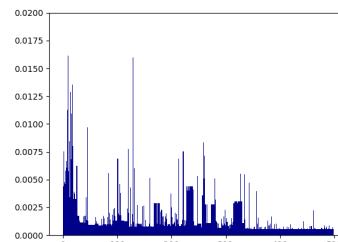


(b)

Figure 2.2: (a) A very small world wide web. Generated by `pagerank_small_plot_graph.ipynb` (b) The corresponding stationary distribution. Generated by `pagerank_demo_small.ipynb`.



(a)



(b)

Figure 2.3: (a) Web graph of 500 sites rooted at www.harvard.edu. (b) Corresponding page rank vector. Generated by `pagerank demo harvard.ipynb`.

As a slightly larger example, Figure 2.3(a) shows a web graph, derived from the root of `harvard.edu`. Figure 2.3(b) shows the corresponding PageRank vector.

2.2.3 Efficiently computing the PageRank vector

Let $G_{ij} = 1$ iff there is a link from j to i . Now imagine performing a random walk on this graph, where at every time step, with probability p you follow one of the outlinks uniformly at random, and with probability $1 - p$ you jump to a random node, again chosen uniformly at random. If there are no outlinks, you just jump to a random page. (These random jumps, including self-transitions, ensure the chain is irreducible (singly connected) and regular. Hence we can solve for its unique stationary distribution using eigenvector methods.) This defines the following transition matrix:

$$M_{ij} = \begin{cases} pG_{ij}/c_j + \delta & \text{if } c_j \neq 0 \\ 1/n & \text{if } c_i = 0 \end{cases} \quad (2.29)$$

where n is the number of nodes, $\delta = (1 - p)/n$ is the probability of jumping from one page to another without following a link and $c_j = \sum_i G_{ij}$ represents the out-degree of page j . (If $n = 4 \cdot 10^9$ and $p = 0.85$, then $\delta = 3.75 \cdot 10^{-11}$.) Here \mathbf{M} is a stochastic matrix in which *columns* sum to one. Note that $\mathbf{M} = \mathbf{A}^\top$ in our earlier notation.

We can represent the transition matrix compactly as follows. Define the diagonal matrix \mathbf{D} with entries

$$d_{jj} = \begin{cases} 1/c_j & \text{if } c_j \neq 0 \\ 0 & \text{if } c_j = 0 \end{cases} \quad (2.30)$$

Define the vector \mathbf{z} with components

$$z_j = \begin{cases} \delta & \text{if } c_j \neq 0 \\ 1/n & \text{if } c_j = 0 \end{cases} \quad (2.31)$$

Then we can rewrite Equation (2.29) as follows:

$$\mathbf{M} = p\mathbf{G}\mathbf{D} + \mathbf{1}\mathbf{z}^\top \quad (2.32)$$

The matrix \mathbf{M} is not sparse, but it is a rank one modification of a sparse matrix. Most of the elements of \mathbf{M} are equal to the small constant δ . Obviously these do not need to be stored explicitly.

Our goal is to solve $\mathbf{v} = \mathbf{M}\mathbf{v}$, where $\mathbf{v} = \pi^\top$. One efficient method to find the leading eigenvector of a large matrix is known as the **power method**. This simply consists of repeated matrix-vector multiplication, followed by normalization:

$$\mathbf{v} \propto \mathbf{M}\mathbf{v} = p\mathbf{G}\mathbf{D}\mathbf{v} + \mathbf{1}\mathbf{z}^\top\mathbf{v} \quad (2.33)$$

It is possible to implement the power method without using any matrix multiplications, by simply sampling from the transition matrix and counting how often you visit each state. This is essentially a Monte Carlo approximation to the sum implied by $\mathbf{v} = \mathbf{M}\mathbf{v}$. Applying this to the data in Figure 2.3(a) yields the stationary distribution in Figure 2.3(b). This took 13 iterations to converge, starting from a uniform distribution. To handle changing web structure, we can re-run this algorithm every day or every week, starting \mathbf{v} off at the old distribution; this is called warm starting [Langville06].

For details on how to perform this Monte Carlo power method in a parallel distributed computing environment, see e.g., [Rajaraman10].

2.2.4 Web spam

PageRank is not foolproof. For example, consider the strategy adopted by JC Penney, a department store in the USA. During the Christmas season of 2010, it planted many links to its home page on 1000s of irrelevant web pages, thus increasing its ranking on Google's search engine [Segal11]. Even though each of these source pages has low PageRank, there were so many of them that their effect added up. Businesses call this **search engine optimization**; Google calls it **web spam**. When Google was notified of this scam (by the *New York Times*), it manually downweighted JC Penney, since such behavior violates Google's code of conduct. The result was that JC Penney dropped from rank 1 to rank 65, essentially making it disappear from view. Automatically detecting such scams relies on various techniques which are beyond the scope of this chapter.

1
2 **2.2.5 Personalized PageRank**

3 The PageRank algorithm computes a single global notion of importance of each web page. In some
4 cases, it is useful for each user to define his own notion of importance. The **Personalized PageRank**
5 algorithm (aka **random walks with restart**) computes a stationary distribution relative to node
6 k , by returning with some probability to a specific starting node k rather than a random node. The
7 corresponding stationary distribution, π^k , gives a measure of how important each node is relative to
8 k . See [**Lofgren2015**] for details. (A similar system is used by Pinterest to infer the similarity of
9 one “pin” (bookmarked webpage) to another, as explained in [**Eksombatchai2018**]).
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

3 Bayesian statistics

3.1 Bayesian concept learning

In this section, we introduce Bayesian statistics using some simple examples inspired by Bayesian models of human learning. This will let us get familiar with the key ideas without getting bogged down by mathematical technicalities.

Consider how a child learns the meaning of a word, such as “dog”. Typically the child’s parents will point out **positive examples** of this concept, saying such things as, “look at the cute dog!”, or “mind the doggy”, etc. The core challenge is to figure out what we mean by the **concept** “dog”, based on a finite (and possibly quite small) number of such examples. Note that the parent is unlikely to provide **negative examples**; for example, people do not usually say “look at that non-dog”. Negative examples may be obtained during an active learning process (e.g., the child says “look at the dog” and the parent says “that’s a cat, dear, not a dog”), but psychological research has shown that people can learn concepts from positive examples alone [Xu07fei]. This means that standard supervised learning methods cannot be used.

We formulate the problem by assuming the data that we see are generated by some hidden concept $h \in \mathcal{H}$, where \mathcal{H} is called the **hypothesis space**. (We use the notation h rather than θ to be consistent with the concept learning literature.) We then focus on computing the posterior $p(h|\mathcal{D})$. In Section 3.1.1, we assume the hypothesis space consists of a finite number of alternative hypotheses; this will significantly simplify the computation of the posterior, allowing us to focus on the ideas and not get too distracted by the math. In Section 3.1.2, we will extend this to continuous hypothesis spaces. This will form the foundation for Bayesian inference of real-valued parameters for more familiar probability models, such as the Bernoulli and the Gaussian, logistic regression, and deep neural networks, that we discuss in later chapters. (See also [Jia2013] for an application of these ideas to the problem of concept learning from images.)

3.1.1 Learning a discrete concept: the number game

Suppose that we are trying to learn some mathematical concept from a teacher who provides examples of that concept. We assume that a concept is defined as the set of positive integers that belong to its **extension**; for example, the concept “even number” is defined by $h_{\text{even}} = \{2, 4, 6, \dots\}$, and the concept “powers of two” is defined by $h_{\text{two}} = \{2, 4, 8, 16, \dots\}$. For simplicity, we assume the range of numbers is between 1 and 100.

For example, suppose we see one example, $\mathcal{D} = \{16\}$. What other numbers do you think are examples of this concept? 17? 6? 32? 99? It’s hard to tell with only one example, so your predictions

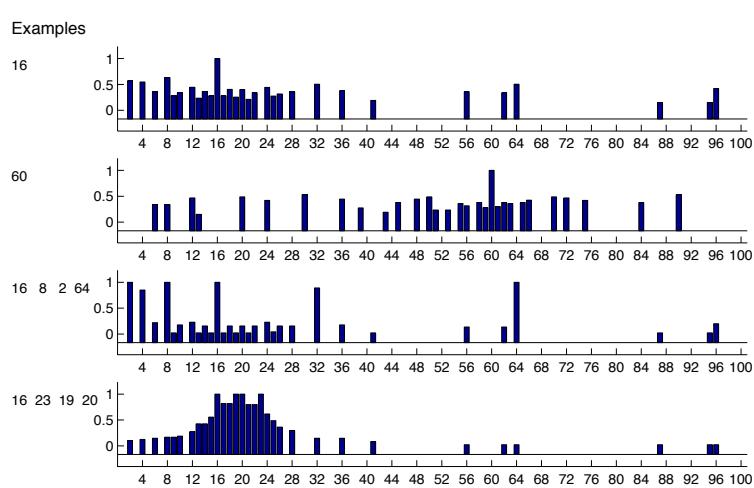


Figure 3.1: Empirical membership distribution in the numbers game, derived from predictions from 8 humans. First two rows: after seeing $\mathcal{D} = \{16\}$ and $\mathcal{D} = \{60\}$. This illustrates diffuse similarity. Third row: after seeing $\mathcal{D} = \{16, 8, 2, 64\}$. This illustrates rule-like behavior (powers of 2). Bottom row: after seeing $\mathcal{D} = \{16, 23, 19, 20\}$. This illustrates focussed similarity (numbers near 20). From Figure 5.5 of [Tenenbaum99]. Used with kind permission of Josh Tenenbaum.

will be quite vague. Presumably numbers that are similar in some sense to 16 are more likely. But similar in what way? 17 is similar, because it is “close by”, 6 is similar because it has a digit in common, 32 is similar because it is also even and a power of 2, but 99 does not seem similar. Thus some numbers are more likely than others.

Now suppose I tell you that $\mathcal{D} = \{2, 8, 16, 64\}$ are positive examples. You may guess that the hidden concept is “powers of two”. Given your beliefs about the true (but hidden) concept, you may confidently predict that $y \in \{2, 4, 8, 16, 32, 64\}$ may also be generated in the future by the teacher. This is an example of **generalization**, since we are making predictions about future data that we have not seen.

Figure 3.1 gives an example of how humans perform at this task. Given a single example, such as $\mathcal{D} = \{16\}$ or $\mathcal{D} = \{60\}$, humans make fairly diffuse predictions over the other numbers that are similar in magnitude. But when given several examples, such as $\mathcal{D} = \{2, 8, 16, 64\}$, humans often find an underlying **pattern**, and use this to make fairly precise predictions about which other numbers might be part of the same concept, even if those other numbers are “far away”.

How can we explain this behavior and emulate it in a machine? The classic approach to the problem of **induction** is to suppose we have a hypothesis space \mathcal{H} of concepts (such as even numbers, all numbers between 1 and 10, etc.), and then to identify the smallest subset of \mathcal{H} that is consistent with the observed data \mathcal{D} ; this is called the **version space**. As we see more examples, the version space shrinks and we become increasingly certain about the underlying hypothesis [Mitchell97].

However, the version space theory cannot explain the human behavior we saw in Figure 3.1. For example, after seeing $\mathcal{D} = \{16, 8, 2, 64\}$, why do people choose the rule “powers of two” and not, say, “all even numbers”, or “powers of two except for 32”, both of which are equally consistent with

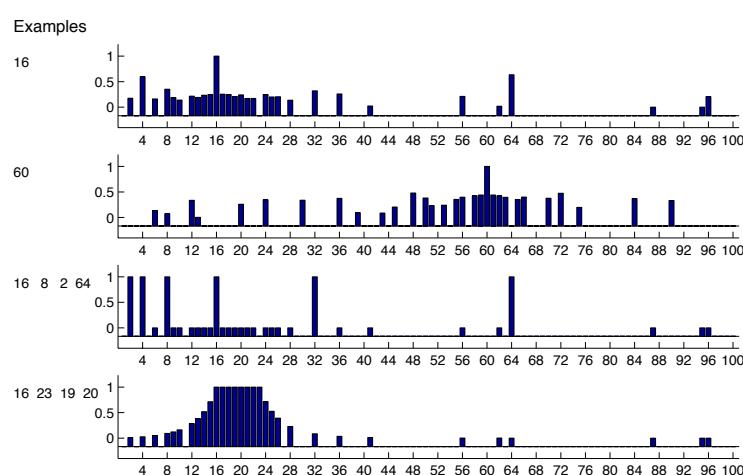


Figure 3.2: Posterior membership probabilities derived using the full hypothesis space. Compare to Figure 3.1. The predictions of the Bayesian model are only plotted for those values for which human data is available; this is why the top line looks sparser than Figure 3.4. From Figure 5.6 of [Tenenbaum99]. Used with kind permission of Josh Tenenbaum.

the evidence? We will now show how Bayesian inference can explain this behavior. The resulting predictions are shown in Figure 3.2.

3.1.1.1 Likelihood

We must explain why people chose h_{two} and not, say, h_{even} after seeing $\mathcal{D} = \{16, 8, 2, 64\}$, given that both hypotheses are consistent with the evidence. The key intuition is that we want to avoid **suspicious coincidences**. For example, if the true concept was even numbers, it would be surprising if we just happened to only see powers of two.

To formalize this, let us assume that the examples are sampled uniformly at random from the **extension** of the concept. (Tenenbaum calls this the **strong sampling assumption**.) Given this assumption, the probability of independently sampling N items (with replacement) from the unknown concept h is given by

$$p(\mathcal{D}|h) = \prod_{n=1}^N p(y_n|h) = \prod_{n=1}^N \frac{1}{\text{size}(h)} \mathbb{I}(y_n \in h) = \left[\frac{1}{\text{size}(h)} \right]^N \mathbb{I}(\mathcal{D} \in h) \quad (3.1)$$

where $\mathbb{I}(\mathcal{D} \in h)$ is non zero iff all the data points lie in the support of h . This crucial equation embodies what Tenenbaum calls the **size principle**, which means the model favors the simplest (smallest) hypothesis consistent with the data. This is more commonly known as **Occam's razor**.

To see how it works, let $\mathcal{D} = \{16\}$. Then $p(\mathcal{D}|h_{\text{two}}) = 1/6$, since there are only 6 powers of two less than 100, but $p(\mathcal{D}|h_{\text{even}}) = 1/50$, since there are 50 even numbers. So the likelihood that $h = h_{\text{two}}$ is higher than if $h = h_{\text{even}}$. After 4 examples, the likelihood of h_{two} is $(1/6)^4 = 7.7 \times 10^{-4}$, whereas the likelihood of h_{even} is $(1/50)^4 = 1.6 \times 10^{-7}$. This is a **likelihood ratio** of almost 5000:1 in

1 favor of h_{two} . This quantifies our earlier intuition that $D = \{16, 8, 2, 64\}$ would be a very suspicious
2 coincidence if generated by h_{even} .

3 3.1.1.2 Prior

4 In the Bayesian approach, we must specify a prior over unknowns, $p(h)$, as well as the likelihood,
5 $p(\mathcal{D}|h)$. To see why this is useful, suppose $D = \{16, 8, 2, 64\}$. Given this data, the concept $h' = \text{"powers}$
6 of two except 32" is more likely than $h = \text{"powers of two"}$, since h' does not need to explain the
7 coincidence that 32 is missing from the set of examples. However, the hypothesis $h' = \text{"powers of}$
8 two except 32" seems "conceptually unnatural". We can capture such intuition by assigning low
9 prior probability to unnatural concepts. Of course, your prior might be different than mine. This
10 subjective aspect of Bayesian reasoning is a source of much controversy, since it means, for example,
11 that a child and a math professor will reach different answers.¹

12 Although the subjectivity of the prior is controversial, it is actually quite useful. If you are told
13 the numbers are from some arithmetic rule, then given 1200, 1500, 900 and 1400, you may think 400
14 is likely but 1183 is unlikely. But if you are told that the numbers are examples of healthy cholesterol
15 levels, you would probably think 400 is unlikely and 1183 is likely, since you assume that healthy
16 levels lie within some range. Thus we see that the prior is the mechanism by which **background**
17 **knowledge** can be brought to bear on a problem. Without this, rapid learning (i.e., from small
18 samples sizes) is impossible.

19 So, what prior should we use? We will initially consider 30 simple arithmetical concepts, such
20 as "even numbers", "odd numbers", "prime numbers", or "numbers ending in 9". We could use a
21 uniform prior over these concepts; however, for illustration purposes, we make the concepts even and
22 odd more likely apriori, and use a uniform prior over the others. We also include two "unnatural"
23 concepts, namely "powers of 2, plus 37" and "powers of 2, except 32", but give them low prior weight.
24 See Figure 3.3a(bottom row) for a plot of this prior.

25 In addition to "rule-like" hypotheses, we consider the set of intervals between n and m for
26 $1 \leq n, m \leq 100$. This allows us to capture concepts based on being "close to" some number, rather
27 than satisfying some more abstract property. We put a uniform prior over the intervals.

28 We can combine these two priors by using a **mixture distribution**, as follows:

$$\text{32} \quad p(h) = \pi \text{Unif}(h|\text{rules}) + (1 - \pi) \text{Unif}(h|\text{intervals}) \quad (3.2)$$

33 where $0 < \pi < 1$ is the **mixture weight** assigned to the rules prior, and $\text{Unif}(h|S)$ is the uniform
34 distribution over the set S .

35 3.1.1.3 Posterior

36 The posterior is simply the likelihood times the prior, normalized: $p(h|\mathcal{D}) \propto p(\mathcal{D}|h)p(h)$. Figure 3.3a
37 plots the prior, likelihood and posterior after seeing $\mathcal{D} = \{16\}$. (In this figure, we only consider
38 rule-like hypotheses, not intervals, for simplicity.) We see that the posterior is a combination of
39 prior and likelihood. In the case of most of the concepts, the prior is uniform, so the posterior is

40 1. A child and a math professor presumably not only have different priors, but also different hypothesis spaces. However,
41 we can finesse that by defining the hypothesis space of the child and the math professor to be the same, and then
42 setting the child's prior weight to be zero on certain "advanced" concepts. Thus there is no sharp distinction between
43 the prior and the hypothesis space.

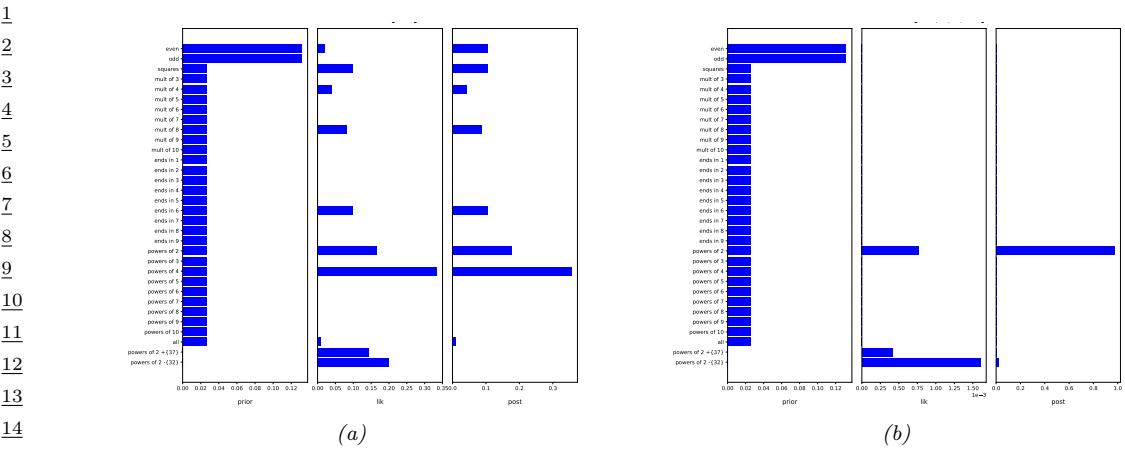


Figure 3.3: (a) Prior, likelihood and posterior for the model when the data is $\mathcal{D} = \{16\}$. (b) Results when $\mathcal{D} = \{2, 8, 16, 64\}$. Adapted from [Tenenbaum99]. Generated by [numbers_game.ipynb](#).

proportional to the likelihood. However, the “unnatural” concepts of “powers of 2, plus 37” and “powers of 2, except 32” have low posterior support, despite having high likelihood, due to the low prior. Conversely, the concept of odd numbers has low posterior support, despite having a high prior, due to the low likelihood.

Figure 3.3b plots the prior, likelihood and posterior after seeing $\mathcal{D} = \{16, 8, 2, 64\}$. Now the likelihood is much more peaked on the powers of two concept, so this dominates the posterior. Essentially the learner has an “aha” moment, and figures out the true concept.² This example also illustrates why we need the low prior on the unnatural concepts, otherwise we would have overfit the data and picked “powers of 2, except for 32”.

3.1.1.4 Posterior predictive

The posterior over hypotheses is our internal **belief state** about the world. The way to test if our beliefs are justified is to use them to predict objectively observable quantities (this is the basis of the scientific method). To do this, we compute the **posterior predictive distribution** over possible future observations:

$$p(y|\mathcal{D}) = \sum_h p(y|h)p(h|\mathcal{D}) \quad (3.3)$$

This is called **Bayes model averaging** [Hoeting99]. Each term is just a weighted average of the predictions of each individual hypothesis. This is illustrated in Figure 3.4. The dots at the bottom show the predictions from each hypothesis; the vertical curve on the right shows the weight associated with each hypothesis. If we multiply each row by its weight and add up, we get the distribution at the top.

2. Humans have a natural desire to figure things out; Alison Gopnik, in her paper “Explanation as orgasm” [Gopnik98], argued that evolution has ensured that we enjoy reducing our posterior uncertainty.

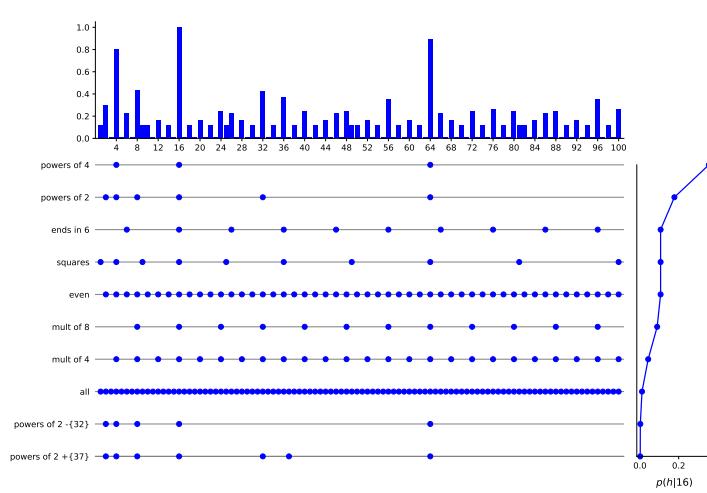


Figure 3.4: Posterior over hypotheses, and the induced posterior over membership, after seeing one example, $\mathcal{D} = \{16\}$. A dot means this number is consistent with this hypothesis. The graph $p(h|\mathcal{D})$ on the right is the weight given to hypothesis h . By taking a weighed sum of dots, we get $p(y \in h|\mathcal{D})$ (top). Adapted from Figure 2.9 of [Tenenbaum99]. Generated by [numbers_game.ipynb](#).

3.1.1.5 MAP, MLE, and the plugin approximation

As the amount of data increases, the posterior will (usually) become concentrated around a single point, namely the **posterior mode**, as we saw in Figure 3.3 (top right plot). The posterior mode is defined as the hypothesis with maximum posterior probability:

$$h_{\text{map}} \triangleq \underset{h}{\operatorname{argmax}} p(h|\mathcal{D}) \quad (3.4)$$

This is also called the **maximum a posterior** or **MAP** estimate.

We can compute the MAP estimate by solving the following **optimization problem**:

$$h_{\text{map}} = \underset{h}{\operatorname{argmax}} p(h|\mathcal{D}) = \underset{h}{\operatorname{argmax}} \log p(\mathcal{D}|h) + \log p(h) \quad (3.5)$$

The first term, $\log p(\mathcal{D}|h)$, is the log of the **likelihood**, $p(\mathcal{D}|h)$. The second term, $\log p(h)$, is the log of the prior. As the data set increases in size, the log likelihood grows in magnitude, but the log prior term remains constant. We thus say that the **likelihood overwhelms the prior**. In this context, a reasonable approximation to the MAP estimate is to ignore the prior term, and just pick the **maximum likelihood estimate** or **MLE**, which is defined as

$$h_{\text{mle}} \triangleq \underset{h}{\operatorname{argmax}} p(\mathcal{D}|h) = \underset{h}{\operatorname{argmax}} \log p(\mathcal{D}|h) = \underset{h}{\operatorname{argmax}} \sum_{n=1}^N \log p(y_n|h) \quad (3.6)$$

Suppose we approximate the posterior by a single **point estimate** \hat{h} , might be the MAP estimate or MLE. We can represent this degenerate distribution as a single point mass

$$p(h|\mathcal{D}) \approx \mathbb{I}(h = \hat{h}) \quad (3.7)$$

where $\mathbb{I}()$ is the indicator function. The corresponding posterior predictive distribution becomes

$$p(y|\mathcal{D}) \approx \sum_h p(y|h) \mathbb{I}(h = \hat{h}) = p(y|\hat{h}) \quad (3.8)$$

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

Although the plug-in approximation is simple, it behaves in a qualitatively inferior way than the fully Bayesian approach when the dataset is small. In the Bayesian approach, we start with broad predictions, and then become more precise in our forecasts as we see more data, which makes intuitive sense. For example, given $\mathcal{D} = \{16\}$, there are many hypotheses with non-negligible posterior mass, so the predicted support over the integers is broad. However, when we see $\mathcal{D} = \{16, 8, 2, 64\}$, the posterior concentrates its mass on one or two specific hypotheses, so the overall predicted support becomes more focused. By contrast, the MLE picks the minimal consistent hypothesis, and predicts the future using that single model. For example, if we see $\mathcal{D} = \{16\}$, we compute h_{mle} to be “all powers of 4” (or the interval hypothesis $h = \{16\}$), and the resulting plugin approximation only predicts $\{4, 16, 64\}$ as having non-zero probability. This is an example of **overfitting**, where we pay too much attention to the specific data that we saw in training, and fail to generalise correctly to novel examples. When we observe more data, the MLE will be forced to pick a broader hypothesis to explain all the data. For example, if we $\mathcal{D} = \{16, 8, 2, 64\}$, the MLE broadens to become “all powers of two”, similar to the Bayesian approach. Thus in the limit of infinite data, both approaches converge to the same predictions. However, in the small sample regime, the fully Bayesian approach, in which we consider multiple hypotheses, will give better (less over confident) predictions.

3.1.2 Learning a continuous concept: the healthy levels game

The number game involved observing a series of discrete variables, and inferring a distribution over another discrete variable from a finite hypothesis space. This made the computations particularly simple: we just needed to sum, multiply and divide. However, in many applications, the variables that we observe are real-valued continuous quantities. More importantly, the unknown parameters are also usually continuous, so the hypothesis space becomes (some subset) of \mathbb{R}^K , where K is the number of parameters. This complicates the mathematics, since we have to replace sums with integrals. However, the basic ideas are the same.

We illustrate these ideas by considering another example of concept learning called the **healthy levels game**, also due to Tenenbaum. The idea is this: we measure two continuous variables, representing the cholesterol and insulin levels of some randomly chosen healthy patients. We would like to know what range of values correspond to a healthy range. As in the numbers game, the challenge is to learn the concept from positive data alone.

Let our hypothesis space be **axis-parallel rectangles** in the plane, as in Figure 3.5. This is a classic example which has been widely studied in machine learning [Mitchell97]. It is also a reasonable assumption for the healthy levels game, since we know (from prior domain knowledge) that healthy levels of both insulin and cholesterol must fall between (unknown) lower and upper bounds. We can represent a rectangle hypothesis as $h = (\ell_1, \ell_2, s_1, s_2)$, where $\ell_j \in (-\infty, \infty)$ are the coordinates (locations) of the lower left corner, and $s_j \in [0, \infty)$ are the lengths of the two sides. Hence the hypothesis space is $\mathcal{H} = \mathbb{R}^2 \times \mathbb{R}_+^2$, where $\mathbb{R}_{\geq 0}$ is the set of non-negative reals.

More complex concepts might require discontinuous regions of space to represent them. Alternatively, we might want to use *latent* rectangular regions to represent more complex, high dimensional

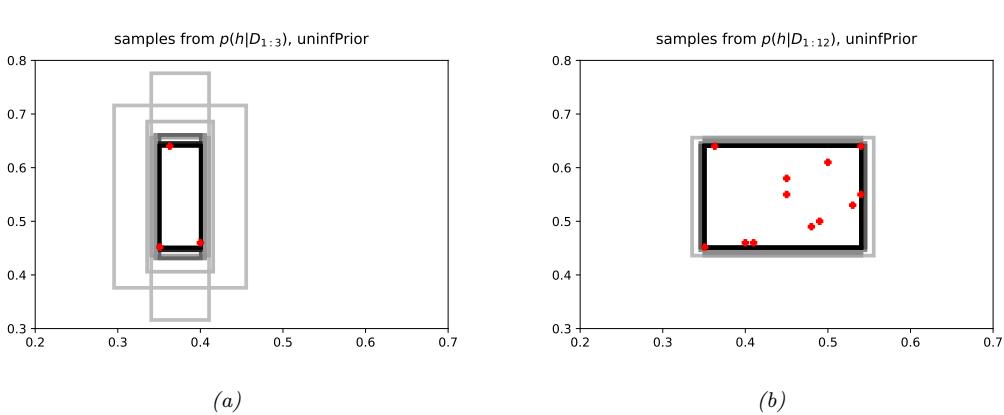


Figure 3.5: Samples from the “healthy levels” game. The axes represent “cholesterol level” and “insulin level”. (a) Given a small number of positive examples (represented by 3 red crosses), there is a lot of uncertainty about the true extent of the rectangle. (b) Given enough data, the smallest enclosing rectangle (which is the maximum likelihood hypothesis) becomes the most probable, although there are many other similar hypotheses that are almost as probable. Adapted from [Tenenbaum99]. Generated by healthy_levels_plots.ipynb.

concepts [Li2019]. The question of where the hypothesis space comes from is a very interesting one, but is beyond the scope of this chapter. (One approach is to use hierarchical Bayesian models, as discussed in [Tenenbaum11].)

3.1.2.1 Likelihood

We assume points are sampled uniformly at random from the support of the rectangle. To simplify the analysis, let us first consider the case of one-dimensional “rectangles”, i.e., lines. In the 1d case, the likelihood is $p(\mathcal{D}|\ell, s) = (1/s)^N$ if all points are inside the interval, otherwise it is 0. Hence

$$p(\mathcal{D}|\ell, s) = \begin{cases} s^{-N} & \text{if } \min(\mathcal{D}) \geq \ell \text{ and } \max(\mathcal{D}) \leq \ell + s \\ 0 & \text{otherwise} \end{cases} \quad (3.9)$$

To generalize this to 2d, we assume the observed features are conditionally independent given the hypothesis. Hence the 2d likelihood becomes

$$p(\mathcal{D}|h) = p(\mathcal{D}_1|\ell_1, s_1)p(\mathcal{D}_2|\ell_2, s_2) \quad (3.10)$$

where $\mathcal{D}_j = \{y_{nj} : n = 1 : N\}$ are the observations for dimension (feature) $j = 1, 2$.

3.1.2.2 Prior

For simplicity, let us assume the prior factorizes, i.e., $p(h) = p(\ell_1)p(\ell_2)p(s_1)p(s_2)$. We will use uninformative priors for each of these terms. As we explain in Main Section 3.5, this means we should use a prior of the form $p(h) \propto \frac{1}{s_1} \frac{1}{s_2}$.

1 **3.1.2.3 Posterior**

2 The posterior is given by

3
$$p(\ell_1, \ell_2, s_1, s_2 | \mathcal{D}) \propto p(\mathcal{D}_1 | \ell_1, s_1) p(\mathcal{D}_2 | \ell_2, s_2) \frac{1}{s_1} \frac{1}{s_2} \quad (3.11)$$

4 We can compute this numerically by discretizing \mathbb{R}^4 into a 4d grid, evaluating the numerator pointwise,
5 and normalizing.

6 Since visualizing a 4d distribution is difficult, we instead draw **posterior samples** from it,
7 $h^s \sim p(h|\mathcal{D})$, and visualize them as rectangles. In Figure 3.5(a), we show some samples when the
8 number N of observed data points is small — we are uncertain about the right hypothesis. In
9 Figure 3.5(b), we see that for larger N , the samples concentrate on the observed data.

10 **3.1.2.4 Posterior predictive distribution**

11 We now consider how to predict which data points we expect to see in the future, given the data we
12 have seen so far. In particular, we want to know how likely it is that we will see any point $\mathbf{y} \in \mathbb{R}^2$.

13 Let us define $y_j^{\min} = \min_n y_{nj}$, $y_j^{\max} = \max_n y_{nj}$, and $r_j = y_j^{\max} - y_j^{\min}$. Then one can show that
14 the posterior predictive distribution is given by

15
$$p(\mathbf{y}|\mathcal{D}) = \left[\frac{1}{(1 + d(y_1)/r_1)(1 + d(y_2)/r_2)} \right]^{N-1} \quad (3.12)$$

16 where $d(y_j) = 0$ if $y_j^{\min} \leq y_j \leq y_j^{\max}$, and otherwise $d(y_j)$ is the distance to the nearest data point
17 along dimension j . Thus $p(\mathbf{y}|\mathcal{D}) = 1$ if \mathbf{y} is inside the support of the training data; if \mathbf{y} is outside the
18 support, the probability density drops off, at a rate that depends on N .

19 Note that if $N = 1$, the predictive distribution is undefined. This is because we cannot infer the
20 extent of a 2d rectangle from just one data point (unless we use a stronger prior).

21 In Figure 3.6(a), we plot the posterior predictive distribution when we have just seen $N = 3$
22 examples; we see that there is a broad generalization gradient, which extends further along the vertical
23 dimension than the horizontal direction. This is because the data has a broader vertical spread than
24 horizontal. In other words, if we have seen a large range in one dimension, we have evidence that the
25 rectangle is quite large in that dimension, but otherwise we prefer compact hypotheses, as follows
26 from the size principle.

27 In Figure 3.6(b), we plot the distribution for $N = 12$. We see it is focused on the smallest consistent
28 hypothesis, since the size principle exponentially down-weights hypothesis which are larger than
29 necessary.

30 **3.1.2.5 Plugin approximation**

31 Now suppose we use a plug-in approximation to the posterior predictive, $p(\mathbf{y}|\mathcal{D}) \approx p(\mathbf{y}|\hat{\theta})$, where $\hat{\theta}$
32 is the MLE or MAP estimate, analogous to the discussion in Section 3.1.1.5. In Figure 3.6(c-d), we
33 show the behavior of this approximation. In both cases, it predicts the smallest enclosing rectangle,
34 since that is the one with maximum likelihood. However, this does not extrapolate beyond the range
35 of the observed data. We also see that initially the predictions are narrower, since very little data
36 has been observed, but that the predictions become broader with more data. By contrast, in the
37

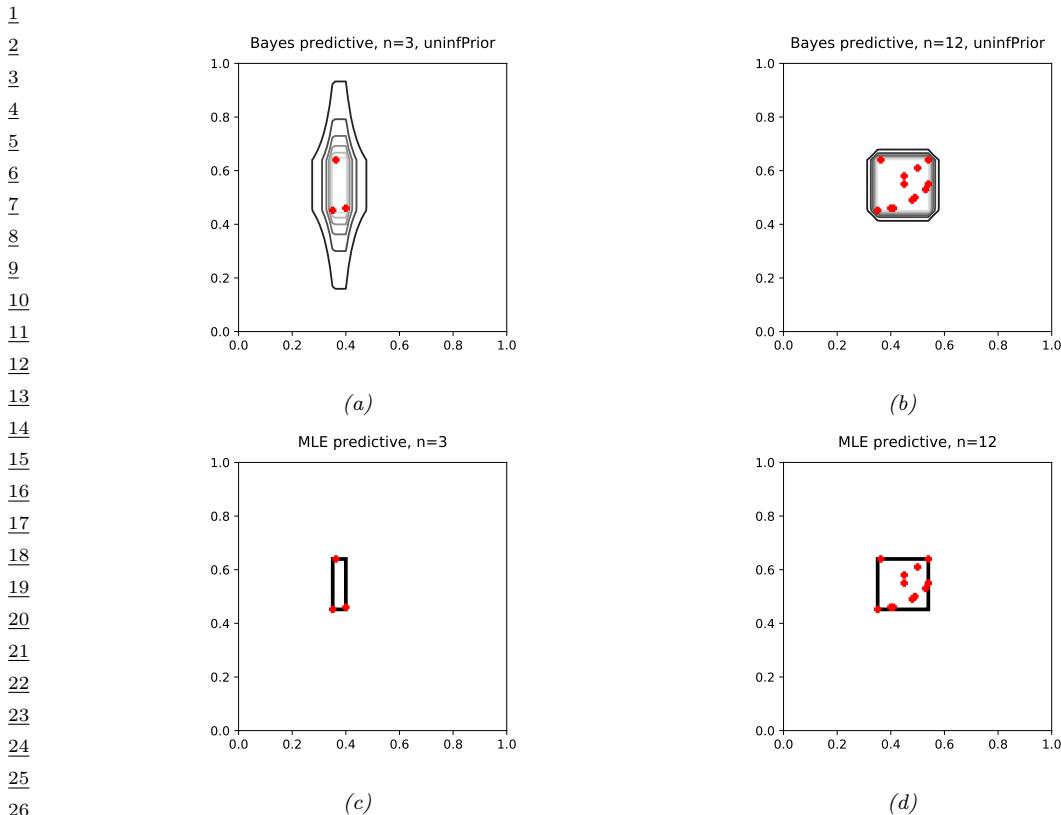


Figure 3.6: Posterior predictive distribution for the healthy levels game. Red crosses are observed data points. Left column: $N = 3$. Right column: $N = 12$. First row: Bayesian prediction. Second row: Plug-in prediction using MLE (smallest enclosing rectangle). We see that the Bayesian prediction goes from uncertain to certain as we learn more about the concept given more data, whereas the plug-in prediction goes from narrow to broad, as it is forced to generalize when it sees more data. However, both converge to the same answer. Adapted from [Tenenbaum99]. Generated by [healthy_levels_plot.ipynb](#).

Bayesian approach, the initial predictions are broad, since there is a lot of uncertainty, but become narrower with more data. In the limit of large data, both methods converge to the same predictions.

3.2 Informative priors

When we have very little data, it is important to choose an informative prior.

For example, consider the classic **taxicab problem** [Jaynes03]: you arrive in a new city, and see a taxi numbered $t = 27$, and you want to infer the total number T of taxis in the city. We will use a uniform likelihood, $p(t|T) = \frac{1}{T} \mathbb{I}(T \geq t)$, since we assume that we could have observed any taxi number up to the maximum value T . The MLE estimate of T is $\hat{T}_{\text{mle}} = t = 27$. But this does not seem reasonable. Instead, most people would guess that $T \sim 2 \times 27 = 54$, on the assumption that if the

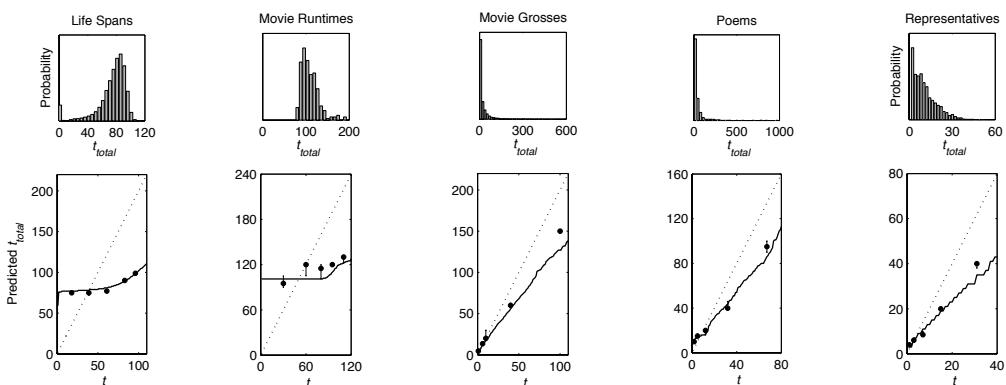


Figure 3.7: Top: empirical distribution of various durational quantities. Bottom: predicted total duration as a function of observed duration, $p(T|t)$. Dots are observed median responses of people. Solid line: Bayesian prediction using informed prior. Dotted line: Bayesian prediction using uninformative prior. From Figure 2a of [Griffiths06]. Used with kind permission of Tom Griffiths.

taxi you saw was a uniform random sample between 0 and T , then it would probably be close to the middle of the distribution.

In general, the conclusions we draw about T will depend strongly on our prior assumptions about what values of T are likely. In the sections below, we discuss different possible informative priors for different problem domains; our presentation is based on [Griffiths06].

Once we have chosen a prior, we can compute the posterior as follows:

$$p(T|t) = \frac{p(t|T)p(T)}{p(t)} \quad (3.13)$$

where

$$p(t) = \int_0^\infty p(t|T)p(T)dT = \int_t^\infty \frac{p(T)}{T}dT \quad (3.14)$$

We will use the posterior median as a point estimate for T . This is the value \hat{T} such that

$$p(T \geq \hat{T}|t) = \int_{\hat{T}}^\infty p(T|t)dT = 0.5 \quad (3.15)$$

Note that the posterior median is often a better summary of the posterior than the posterior mode, for reasons explained in Main Section 7.4.1.

3.2.1 Domain specific priors

At the top of Figure 3.7, we show some histograms representing the empirical distribution of various kinds of scalar quantities, specifically: the number of years people live, the number of minutes a movie lasts, the amount of money made (in 1000s of US dollars) by movies, the number of lines of a

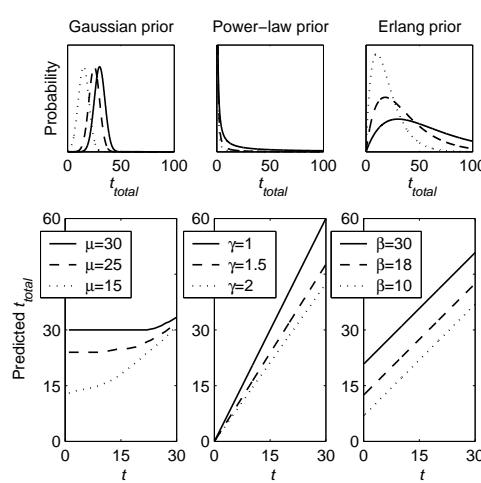


Figure 3.8: Top: three different prior distributions, for three different parameter values. Bottom: corresponding predictive distributions. From Figure 1 of [Griffiths06]. Used with kind permission of Tom Griffiths.

poem, and the number of years someone serves in the US house of Representatives. (The sources for these data is listed in [Griffiths06].)

At the bottom, we plot $p(T|t)$ as a function of t for each of these domains. The solid dots are the median responses of a group of people when asked to predict T from a single observation t . The solid line is the posterior median computed by a Bayesian model using a domain-appropriate prior (details below). The dotted line is the posterior median computed by a Bayesian model using an uninformative $1/T$ prior. We see a remarkable correspondence between people and the informed Bayesian model. This suggests that people can implicitly use an appropriate kind of prior for a wide range of problems, as argued in [Griffiths06]. In the sections below, we discuss some suitable parametric priors which capture this behavior. In [Griffiths06], they also consider some datasets that can only be well-modeled by a non-parametric prior. Bayesian inference works well in that case, too, but we omit this for simplicity.

3.2.2 Gaussian prior

Looking at Figure 3.7(a-b), it seems clear that life-spans and movie run-times can be well-modeled by a Gaussian, $\mathcal{N}(T|\mu, \sigma^2)$. Unfortunately, we cannot compute the posterior median in closed form if we use a Gaussian prior, but we can still evaluate it numerically, by solving a 1d integration problem. The resulting plot of $\hat{T}(t)$ vs t is shown in Figure 3.8 (bottom left). For values of t much less than the prior mean, μ , the predicted value of T is about equal to μ , so the left part of the curve is flat. For values of t much greater than μ , the predicted value converges to a line slightly above the diagonal, i.e., $\hat{T}(t) = t + \epsilon$ for some small (and decreasing) $\epsilon > 0$.

To see why this behavior makes intuitive sense, consider encountering a man at age 18, 39 or 51: in all cases, a reasonable prediction is that he will live to about $\mu = 75$ years. But now imagine meeting a man at age 80: we probably would not expect him to live much longer, so we predict

1 $\hat{T}(80) \approx 80 + \epsilon.$

3

4 3.2.3 Power-law prior

5

6 Looking at Figure 3.7(c-d), it seems clear that movie grosses and poem length can be modeled by
7 a **power law** distribution of the form $p(T) \propto T^{-\gamma}$ for $\gamma > 0$. (If $\gamma > 1$, this is called a Pareto
8 distribution, see Main Section 2.2.3.5.) Power-laws are characterized by having very **long tails**.
9 This captures the fact that most movies make very little money, but a few blockbusters make a lot.
10 The number of lines in various poems also has this shape, since there are a few epic poems, such
11 as Homer's *Odyssey*, but most are short, like haikus. Wealth has a similarly skewed distribution in
12 many countries, especially in plutocracies such as the USA (see e.g., [inequality.org](#)).

13 In the case of a power-law prior, $p(T) \propto T^{-\gamma}$, we can compute the posterior median analytically.
14 We have

15

$$\text{p}(t) \propto \int_t^\infty T^{-(\gamma+1)} dT = -\frac{1}{\gamma} T^{-\gamma}|_t^\infty = \frac{1}{\gamma} t^{-\gamma} \quad (3.16)$$

16

17 Hence the posterior becomes

19

20

$$\text{p}(T|t) = \frac{T^{-(\gamma+1)}}{\frac{1}{\gamma} t^{-\gamma}} = \frac{\gamma t^\gamma}{T^{\gamma+1}} \quad (3.17)$$

21

23 for values of $T \geq t$. We can derive the posterior median as follows:

24

25

$$\text{p}(T > T_M|t) = \int_{T_M}^\infty \frac{\gamma t^\gamma}{T^{\gamma+1}} dT = -\left(\frac{t^\gamma}{T}\right)|_{T_M}^\infty = \left(\frac{t}{T_M}\right)^\gamma \quad (3.18)$$

26

28 Solving for T_M such that $\text{p}(T > T_M|t) = 0.5$ gives $T_M = 2^{1/\gamma} t$.

29 This is plotted in Figure 3.8 (bottom middle). We see that the predicted duration is some constant
30 multiple of the observed duration. For the particular value of γ that best fits the empirical distribution
31 of movie grosses, the optimal prediction is about 50% larger than the observed quantity. So if we
32 observe that a movie has made \$40M to date, we predict that it will make \$60M in total.

33 As Griffiths and Tenenbaum point out, this rule is inappropriate for quantities that follow a
34 Gaussian prior, such as people's ages. As they write, "Upon meeting a 10-year-old girl and her
35 75-year-old grandfather, we would never predict that the girl will live a total of 15 years (1.5×10)
36 and that the grandfather will live to be 112 (1.5×75).". This shows that people implicitly know what
37 kind of prior to use when solving prediction problems of this kind.

38

39 3.2.4 Erlang prior

40

41 Looking at Figure 3.7(e), it seems clear that the number of years a US Representative is approximately
42 modeled by a gamma distribution (Main Section 2.2.3.1). Griffiths and Tenenbaum use a special
43 case of the Gamma distribution, where the shape parameter is $a = 2$; this is known as the **Erlang**
44 **distribution**:

45

$$\text{p}(T) = \text{Ga}(T|2, 1/\beta) \propto T e^{-T/\beta} \quad (3.19)$$

46

For the Erlang prior, we can also compute the posterior median analytically. We have

$$p(t) \propto \int_t^\infty \exp(-T/\beta) dT = -\beta \exp(-T/\beta)|_t^\infty = \beta \exp(-t/\beta) \quad (3.20)$$

so the posterior has the form

$$p(T|t) = \frac{\exp(-T/\beta)}{\beta \exp(-t/\beta)} = \frac{1}{\beta} \exp(-(T-t)/\beta) \quad (3.21)$$

for values of $T \geq t$. We can derive the posterior median as follows:

$$p(T > T_M|t) = \int_{T_M}^\infty \frac{1}{\beta} \exp(-(T-t)/\beta) dT = -\exp(-(T-t)/\beta)|_{T_M}^\infty = \exp(-(T_M-t)/\beta) \quad (3.22)$$

Solving for T_M such that $p(T > T_M|t) = 0.5$ gives $T_M = t + \beta \log 2$.

This is plotted in Figure 3.8 (bottom right). We see that the best guess is simply the observed value plus a constant, where the constant reflects the average term in office.

3.3 Tweedie's formula (Empirical Bayes without estimating the prior)

In this section, we present **Tweedie's formula** [Efron2011] which is a way to estimate the posterior of a quantity without knowing the prior. Instead, we replace the prior with an empirical estimate of the score function, which is the derivative of the log marginal density, as we explain below. This is useful since it allows us to estimate a latent quantity from noisy observations without ever observing the latent quantity itself.

Consider the case of a scalar natural parameter η with prior $g(\eta)$ and exponential family likelihood

$$y|\eta \sim f_\eta(y) = e^{\eta y - \psi(\eta)} f_0(y) \quad (3.23)$$

where $\psi(\eta)$ is the cumulant generating function (cgf) that makes $f_\eta(y)$ integrate to 1, and $f_0(y)$ is the density when $\eta = 0$. For example, in the case of a 1d Gaussian with fixed variance σ^2 , we have $\eta = \mu/\sigma^2$, $\psi(\eta) = \frac{1}{2}\sigma^2\eta^2$, and $f_0(y) = \mathcal{N}(y|0, \sigma^2)$.

Bayes rule gives the following posterior

$$g(\eta|y) = \frac{f_\eta(y)g(\eta)}{f(y)} \quad (3.24)$$

$$f(y) = \int_Y f_\eta(y)g(\eta)d\eta \quad (3.25)$$

Plugging in Equation (3.23) we get

$$g(\eta|y) = e^{y\eta - \lambda(y)}[g(\eta)e^{-\psi(y)}] \quad (3.26)$$

$$\lambda(y) = \log \left(\frac{f(y)}{f_0(y)} \right) \quad (3.27)$$

So we see that the posterior is an exponential family with canonical parameter y and cgf $\lambda(y)$. Letting

$$\ell(y) = \log f(y), \quad \ell_0(y) = \log f_0(y) \quad (3.28)$$

we can therefore derive the posterior moments as follows:

$$\mathbb{E}[\eta|y] = \lambda'(y) = \ell'(y) - \ell'_0(y), \quad \mathbb{V}[\eta|y] = \lambda''(y) = \ell''(y) - \ell''_0(y) \quad (3.29)$$

For the Gaussian case we have $\ell_0(y) = -y^2/(2\sigma^2)$, so

$$\mathbb{E}[\mu|y] = y + \sigma^2 \ell'(y), \quad \mathbb{V}[\mu|y] = \sigma^2(1 + \sigma^2 \ell''(y)) \quad (3.30)$$

If we plug in an empirical estimate of the score function $\ell'(y)$, we can compute an empirical Bayes estimate of the posterior mean without having to estimate the prior.

We can extend this to the multivariate Gaussian case as shown in [Raphan2007]. In particular, suppose the likelihood has the form

$$p(\mathbf{y}|\boldsymbol{\mu}) = \mathcal{N}(\mathbf{y}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \quad (3.31)$$

with marginal likelihood

$$p(\mathbf{y}) = \int p(\boldsymbol{\mu})p(\mathbf{y}|\boldsymbol{\mu})d\boldsymbol{\mu} \quad (3.32)$$

Since $\nabla_{\mathbf{y}}p(\mathbf{y}|\mathbf{x}) = p(\mathbf{y}|\mathbf{x})\nabla_{\mathbf{y}}\log p(\mathbf{y}|\mathbf{x})$ we have

$$\frac{\nabla_{\mathbf{y}}p(\mathbf{y})}{p(\mathbf{y})} = \frac{\int p(\boldsymbol{\mu})\nabla_{\mathbf{y}}p(\mathbf{y}|\boldsymbol{\mu})d\boldsymbol{\mu}}{p(\mathbf{y})} \quad (3.33)$$

$$= \frac{\int p(\boldsymbol{\mu})\boldsymbol{\Sigma}^{-1}p(\mathbf{y}|\boldsymbol{\mu})(\boldsymbol{\mu} - \mathbf{y})d\boldsymbol{\mu}}{p(\mathbf{y})} \quad (3.34)$$

$$= \boldsymbol{\Sigma}^{-1} \int p(\boldsymbol{\mu}|\mathbf{y})(\boldsymbol{\mu} - \mathbf{y})d\boldsymbol{\mu} \quad (3.35)$$

$$= \boldsymbol{\Sigma}^{-1}[\mathbb{E}[\boldsymbol{\mu}|\mathbf{y}] - \mathbf{y}] \quad (3.36)$$

and hence

$$\mathbb{E}[\boldsymbol{\mu}|\mathbf{y}] = \mathbf{y} + \boldsymbol{\Sigma}\nabla_{\mathbf{y}}\log p(\mathbf{y}) \quad (3.37)$$

For high dimensional signals, we can use neural networks to approximate the score function, as we discuss in Main Section 24.3.

4 Graphical models

4.1 More examples of DGMs

4.1.1 Water sprinkler

Suppose we want to model the dependencies between 5 random variables: C (whether it is cloudy season or not), R (whether it is raining or not), S (whether the water sprinkler is on or not), W (whether the grass is wet or not), and L (whether the grass is slippery or not). We know that the cloudy season makes rain more likely, so we add a $C \rightarrow R$ arc. We know that the cloudy season makes turning on a water sprinkler less likely, so we add a $C \rightarrow S$ arc. We know that either rain or sprinklers can cause the grass to get wet, so we add $S \rightarrow W$ and $R \rightarrow W$ edges. Finally, we know that wet grass can be slippery, so we add a $W \rightarrow L$ edge. See Figure 4.1 for the resulting DAG.

Formally, this defines the following joint distribution:

$$p(C, S, R, W, L) = p(C)p(S|C)p(R|C, \cancel{S})p(W|S, R, \cancel{C})p(L|W, \cancel{S}, \cancel{R}, \cancel{C}) \quad (4.1)$$

where we strike through terms that are not needed due to the conditional independence properties of the model.

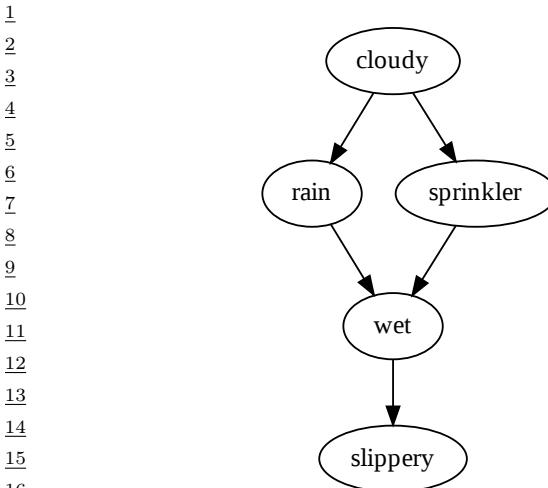
In addition to the graph structure, we need to specify the CPDs. For discrete random variables, we can represent the CPD as a table, which means we have a separate row (i.e., a separate categorical distribution) for each **conditioning case**, i.e., for each combination of parent values. This is known as a **conditional probability table** or **CPT**. We can represent the i 'th CPT as a tensor

$$\theta_{ijk} \triangleq p(x_i = k | \mathbf{x}_{\text{pa}(i)} = j) \quad (4.2)$$

Thus $\boldsymbol{\theta}_i$ 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, $i \in [N_G]$; 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 $J_i = \prod_{p \in \text{pa}(i)} K_p$. For example, consider the wet grass node in Figure 4.1. If all nodes are binary, we can represent its CPT by the table of numbers shown in Figure 4.1(right).

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.

Given the model, we can use it to answer probabilistic queries. For example, one can show (using the code at [sprinkler_pgm.ipynb](#)) that $p(R = 1) = 0.5$, which means the probability it rained (before we collect any data) is 50%. This is consistent with the CPT for that node. Now suppose



S	R	$p(W = 0)$	$p(W = 1)$
0	0	1.0	0.0
1	0	0.1	0.9
0	1	0.1	0.9
1	1	0.01	0.99

17 *Figure 4.1: Water sprinkler graphical model. (Left). The DAG. Generated by [sprinkler_pgm.ipynb](#). (Right).*
 18 *The CPT for the W node, assuming all variables are binary.*

20
 21
 22 we see that the grass is wet: our belief that it rained changes to $p(R = 1|W = 1) = 0.7079$. Now
 23 suppose we also notice the water sprinkler was turned on: our belief that it rained goes down to
 24 $p(R = 1|W = 1, S = 1) = 0.3204$. This negative mutual interaction between multiple causes of
 25 some observations is called the **explaining away** effect, also known as **Berkson's paradox** (see
 26 Main Section 4.2.4.2 for details).

27 In general, we can use our joint model to answer all kinds of probabilistic queries. This includes
 28 inferring latent quantities (such as whether the water sprinkler turned on or not given that the grass
 29 is wet), as well as predicting observed quantities, such as whether the grass will be slippery. It is this
 30 ability to answer arbitrary queries that makes PGMs so useful. See Main Chapter 9 for algorithmic
 31 details.

32 Note also that inference requires a fully-specified model. This means we need to know the
 33 graph structure G and the parameters of the CPDs θ . We discuss how to learn the parameters in
 34 Main Section 4.2.7 and the structure in Main Section 30.3.

35

36 4.1.2 Asia network

37 In this section, we consider a hypothetical medical model proposed in [Lauritzen88] which is known
 38 in the literature as the “**Asia network**”. (The name comes from the fact that was designed to
 39 diagnose various lung diseases in Western patients returning from a trip to Asia. Note that this
 40 example predates the COVID-19 pandemic by many years, and is a purely fictitious model.)

41 Figure 4.2a shows the model, as well as the prior marginal distributions over each node (assumed
 42 to be binary). Now suppose the patient reports that they have **Dyspnea**, aka shortness of breath.
 43 We can represent this fact “clamping” the distribution to be 100% probability that Dyspnea=Present,
 44 and 0% probability that Dyspnea=Absent. We then propagate this new information through the
 45 network to get the updated marginal distributions shown in Figure 4.2b. We see that the probability
 46

47

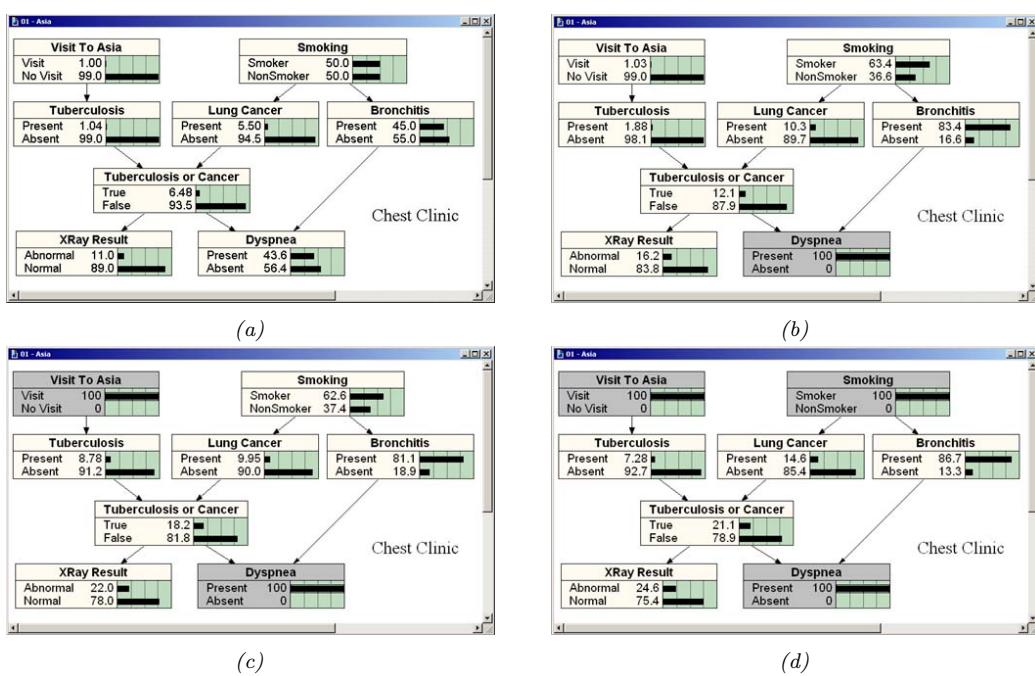


Figure 4.2: Illustration of belief updating in the “Asia” PGM. The histograms show the marginal distribution of each node. (a) Prior. (b) Posterior after conditioning on $\text{Dyspnea}=\text{Present}$. (c) Posterior after also conditioning on $\text{VisitToAsia}=\text{True}$. (d) Posterior after also conditioning on $\text{Smoking}=\text{True}$. Generated by [asia_pgm.ipynb](#).

of lung cancer has gone up from 5% to 10%, and probability of bronchitis has gone up from 45% to 83%.

However, it could also be a undiagnosed case of TB (tuberculosis), which may have been caused by exposure to an infectious lung disease that was prevalent in Asia at the time. So the doctor asks the patient if they have recently been to Asia, and they say yes. Figure 4.2c shows the new belief state of each node. We see that the probability of TB has increased from 2% to 9%. However, Bronchitis remains the most likely explanation of the symptoms.

To gain more information the doctor asks if the patient smokes, and they say yes. Figure 4.2d shows the new belief state of each node. Now the probability of cancer and bronchitis have both gone up. In addition, the posterior predicted probability that an X-ray will show an abnormal result has gone up to 24%, so the doctor may decide it is now worth ordering a test to verify this hypothesis.

This example illustrates the nature of recursive Bayesian updating, and how it can be useful for active learning and sequential decision making,

4.1.3 The QMR network

In this section, we describe the DPGM known as the **quick medical reference** or **QMR** network [Shwe91]. This is a model of infectious diseases and is shown (in simplified form) in Figure 4.3. (We

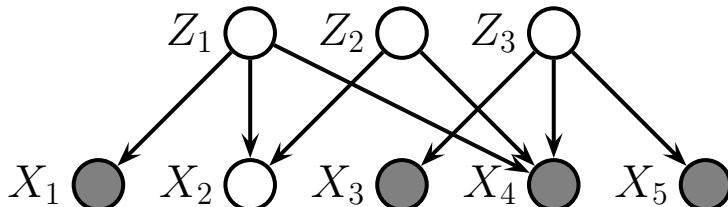


Figure 4.3: A small version of the QMR network. All nodes are binary. The hidden nodes z_k represent diseases, and the visible nodes x_d represent symptoms. In the full network, there are 570 hidden (disease) nodes and 4075 visible (symptom) nodes. The shaded (solid gray) leaf nodes are observed; in this example, symptom x_2 is not observed (i.e., we don't know if it is present or absent). Of course, the hidden diseases are never observed.

z_0	z_1	z_2	$P(x_d = 0 z_0, z_1, z_2)$	$P(x_d = 1 z_0, z_1, z_2)$
1	0	0	θ_0	$1 - \theta_0$
1	1	0	$\theta_0\theta_1$	$1 - \theta_0\theta_1$
1	0	1	$\theta_0\theta_2$	$1 - \theta_0\theta_2$
1	1	1	$\theta_0\theta_1\theta_2$	$1 - \theta_0\theta_1\theta_2$

Table 4.1: Noisy-OR CPD for $p(x_d|z_0, z_1, z_2)$, where $z_0 = 1$ is a leak node.

omit the parameters for clarity, so we don't use plate notation.) The QMR model is a **bipartite graph** structure, with hidden diseases (causes) at the top and visible symptoms or findings at the bottom. We can write the distribution as follows:

$$p(\mathbf{z}, \mathbf{x}) = \prod_{k=1}^K p(z_k) \prod_{d=1}^D p(x_d|\mathbf{x}_{\text{pa}(d)}) \quad (4.3)$$

where z_k represents the k 'th disease and x_d represents the d 'th symptom. This model can be used inside an inference engine to compute the posterior probability of each disease given the observed symptoms, i.e., $p(z_k|\mathbf{x}_v)$, where \mathbf{x}_v is the set of visible symptom nodes. (The symptoms which are not observed can be removed from the model, assuming they are missing at random (Main Section 3.11), because they contribute nothing to the likelihood; this is called **barren node removal**.)

We now discuss the parameterization of the model. For simplicity, we assume all nodes are binary. The CPD for the root nodes are just Bernoulli distributions, representing the prior probability of that disease. Representing the CPDs for the leaves (symptoms) using CPTs would require too many parameters, because the **fan-in** (number of parents) of many leaf nodes is very high. A natural alternative is to use logistic regression to model the CPD, $p(x_d|z_{\text{pa}(d)}) = \text{Ber}(x_d|\sigma(\mathbf{w}_d^\top \mathbf{z}_{\text{pa}(d)}))$. However, we use an alternative known as the **noisy-OR** model, which we explain below,

The noisy-OR model assumes that if a parent is on, then the child will usually also be on (since it is an or-gate), but occasionally the "links" from parents to child may fail, independently at random. If a failure occurs, the child will be off, even if the parent is on. To model this more precisely, let

$\theta_{kd} = 1 - q_{kd}$ be the probability that the $k \rightarrow d$ link fails. The only way for the child to be off is if all the links from all parents that are on fail independently at random. Thus

$$p(x_d = 0 | \mathbf{z}) = \prod_{k \in \text{pa}(d)} \theta_{kd}^{\mathbb{I}(z_k=1)} \quad (4.4)$$

Obviously, $p(x_d = 1 | \mathbf{z}) = 1 - p(x_d = 0 | \mathbf{z})$. In particular, let us define $q_{kd} = 1 - \theta_{kd} = p(x_d = 1 | z_k = 1, \mathbf{z}_{-k} = 0)$; this is the probability that k can activate d “on its own”; this is sometimes called its “**causal power**” (see e.g., [Korb2011]).

If we observe that $x_d = 1$ but all its parents are off, then this contradicts the model. Such a data case would get probability zero under the model, which is problematic, because it is possible that someone exhibits a symptom but does not have any of the specified diseases. To handle this, we add a dummy **leak node** z_0 , which is always on; this represents “all other causes”. The parameter q_{0d} represents the probability that the background leak can cause symptom d on its own. The modified CPD becomes

$$p(x_d = 0 | \mathbf{z}) = \theta_{0d} \prod_{k \in \text{pa}(d)} \theta_{kd}^{z_k} \quad (4.5)$$

See Table 4.1 for a numerical example.

If we define $w_{kd} \triangleq \log(\theta_{kd})$, we can rewrite the CPD as

$$p(x_d = 1 | \mathbf{z}) = 1 - \exp \left(w_{0d} + \sum_k z_k w_{kd} \right) \quad (4.6)$$

We see that this is similar to a logistic regression model.

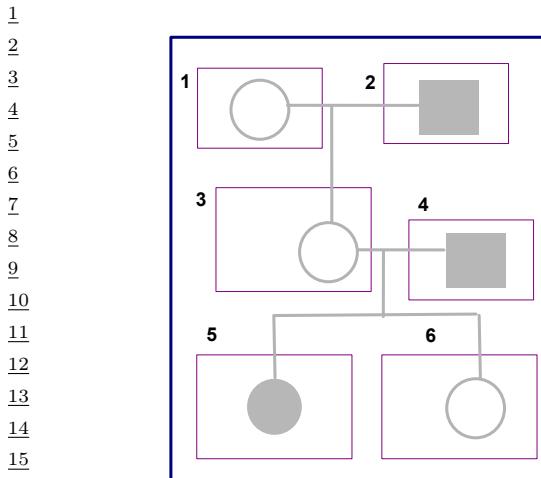
It is relatively easy to set the θ_{kd} parameters by hand, based on domain expertise, as was done with QMR. Such a model is called a **probabilistic expert system**. In this book, we focus on learning parameters from data; we discuss how to do this in Main Section 4.2.7.2 (see also [Neal92; Meek97]).

4.1.4 Genetic linkage analysis

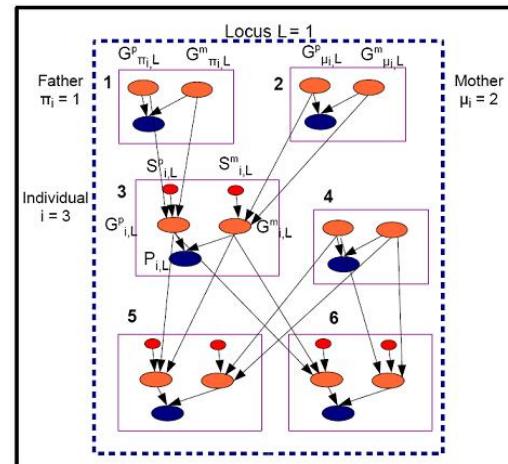
DPGM’s are widely used in statistical genetics. In this section, we discuss the problem of **genetic linkage analysis**, in which we try to infer which genes cause a given disease. We explain the method below.

4.1.4.1 Single locus

We start with a **pedigree graph**, which is a DAG that representing the relationship between parents and children, as shown in Figure 4.4(a). Next we construct the DGM. For each person (or animal) i and location or locus j along the genome, we create three nodes: the observed **phenotype** P_{ij} (which can be a property such as blood type, or just a fragment of DNA that can be measured), and two hidden **alleles** (genes), G_{ij}^m and G_{ij}^p , one inherited from i ’s mother (maternal allele) and the other from i ’s father (paternal allele). Together, the ordered pair $\mathbf{G}_{ij} = (G_{ij}^m, G_{ij}^p)$ constitutes i ’s hidden **genotype** at locus j .



(a)



(b)

Figure 4.4: Left: family tree, circles are females, squares are males. Individuals with the disease of interest are highlighted. Right: DGM for locus $j = L$. Blue node P_{ij} is the phenotype for individual i at locus j . Orange nodes $G_{ij}^{p/m}$ is the paternal/ maternal allele. Small red nodes $S_{ij}^{p/m}$ are the paternal/ maternal selection switching variables. The founder (root) nodes do not have any parents, and hence do no need switching variables. All nodes are hidden except the blue phenotypes. Adapted from Figure 3 from [Friedman00linkage].

G^p	G^m	$p(P = a)$	$p(P = b)$	$p(P = o)$	$p(P = ab)$
a	a	1	0	0	0
a	b	0	0	0	1
a	o	1	0	0	0
b	a	0	0	0	1
b	b	0	1	0	0
b	o	0	1	0	0
o	a	1	0	0	0
o	b	0	1	0	0
o	o	0	0	1	0

Table 4.2: CPT which encodes a mapping from genotype to phenotype (bloodtype). This is a deterministic, but many-to-one, mapping.

Obviously we must add $G_{ij}^m \rightarrow X_{ij}$ and $G_{ij}^p \rightarrow P_{ij}$ arcs representing the fact that genotypes cause phenotypes. The CPD $p(P_{ij}|G_{ij}^p, G_{ij}^m)$ is called the **penetrance model**. As a very simple example, suppose $P_{ij} \in \{A, B, O, AB\}$ represents person i 's observed bloodtype, and $G_{ij}^m, G_{ij}^p \in \{A, B, O\}$ is their genotype. We can represent the penetrance model using the deterministic CPD shown in Table 4.2. For example, A dominates O, so if a person has genotype AO or OA, their phenotype will be A.

In addition, we add arcs from i 's mother and father into $G_{ij}^{m/p}$, reflecting the **Mendelian inheritance** of genetic material from one's parents. More precisely, let $\mu_i = k$ be i 's mother. For example, in Figure 4.4(b), for individual $i = 3$, we have $\mu_i = 2$, since 2 is the mother of 3. The gene G_{ij}^m could

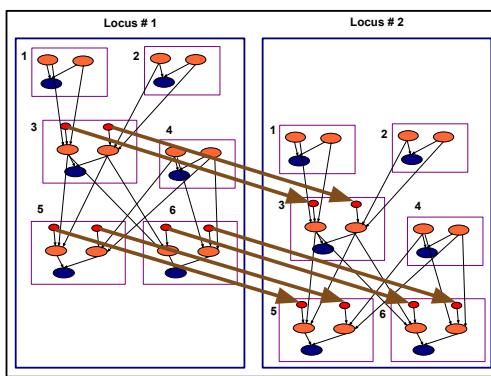


Figure 4.5: Extension of Figure 4.4 to two loci, showing how the switching variables are spatially correlated. This is indicated by the $S_{ij}^m \rightarrow S_{i,j+1}^m$ and $S_{ij}^p \rightarrow S_{i,j+1}^p$ edges. Adapted from Figure 3 from [Friedman00linkage].

either be equal to G_{kj}^m or G_{kj}^p , that is, i 's maternal allele is a copy of one of its mother's two alleles. Let S_{ij}^m be a hidden switching variable that specifies the choice. Then we can use the following CPD, known as the **inheritance model**:

$$p(G_{ij}^m | G_{kj}^m, G_{kj}^p, S_{ij}^m) = \begin{cases} \mathbb{I}(G_{ij}^m = G_{kj}^m) & \text{if } S_{ij}^m = m \\ \mathbb{I}(G_{ij}^m = G_{kj}^p) & \text{if } S_{ij}^m = p \end{cases} \quad (4.7)$$

We can define $p(G_{ij}^p | G_{kj}^m, G_{kj}^p, S_{ij}^p)$ similarly, where $\pi = p_i$ is i 's father. The values of the S_{ij} are said to specify the **phase** of the genotype. The values of $G_{i,j}^p$, $G_{i,j}^m$, $S_{i,j}^p$ and $S_{i,j}^m$ constitute the **haplotype** of person i at locus j . (The genotype $G_{i,j}^p$ and $G_{i,j}^m$ without the switching variables $S_{i,j}^p$ and $S_{i,j}^m$ is called the “unphased” genotype.)

Next, we need to specify the prior for the root nodes, $p(G_{ij}^m)$ and $p(G_{ij}^p)$. This is called the **founder model**, and represents the overall prevalence of different kinds of alleles in the population. We usually assume independence between the loci for these founder alleles, and give these root nodes uniform priors. Finally, we need to specify priors for the switch variables that control the inheritance process. For now, we will assume there is just a single locus, so we can assume uniform priors for the switches. The resulting DGM is shown in Figure 4.4(b).

4.1.4.2 Multiple loci

We get more statistical power if we can measure multiple phenotypes and genotypes. In this case, we must model spatial correlation amongst the genes, since genes that are close on the genome are likely to be coinherited, since there is less likely to be a crossover event between them. We can model this by imposing a two-state Markov chain on the switching variables S 's, where the probability of switching state at locus j is given by $\theta_j = \frac{1}{2}(1 - e^{-2d_j})$, where d_j is the distance between loci j and $j + 1$. This is called the **recombination model**. The resulting DGM for two linked loci in Figure 4.5.

We can now use this model to determine where along the genome a given disease-causing gene is assumed to lie — this is the genetic linkage analysis task. The method works as follows. First, suppose all the parameters of the model, including the distance between all the marker loci, are known. The only unknown is the location of the disease-causing gene. If there are L marker loci, we construct $L + 1$ models: in model ℓ , we postulate that the disease gene comes after marker ℓ , for $0 < \ell < L + 1$. We can estimate the Markov switching parameter $\hat{\theta}_\ell$, and hence the distance d_ℓ between the disease gene and its nearest known locus. We measure the quality of that model using its likelihood, $p(\mathcal{D}|\hat{\theta}_\ell)$. We then can then pick the model with highest likelihood.

Note, however, that computing the likelihood requires marginalizing out all the hidden S and G variables. See [Fishelson02] and the references therein for some exact methods for this task; these are based on the variable elimination algorithm, which we discuss in Main Section 9.5. Unfortunately, for reasons we explain in Main Section 9.5.4, exact methods can be computationally intractable if the number of individuals and/or loci is large. See [Albers06] for an approximate method for computing the likelihood based on the “cluster variation method”.

Note that it is possible to extend the above model in multiple ways. For example, we can model evolution amongst phylogenies using a **phylogenetic HMM** [Siepel03].

4.2 More examples of UGMs

4.3 Restricted Boltzmann machines (RBMs) in more detail

In this section, we discuss RBMs in more detail.

4.3.1 Binary RBMs

The most common form of RBM has binary hidden nodes and binary visible nodes. The joint distribution then has the following form:

$$p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp(-\mathcal{E}(\mathbf{x}, \mathbf{z}; \boldsymbol{\theta})) \quad (4.8)$$

$$\begin{aligned} \mathcal{E}(\mathbf{x}, \mathbf{z}; \boldsymbol{\theta}) &\triangleq - \sum_{d=1}^D \sum_{k=1}^K x_d z_k W_{dk} - \sum_{d=1}^D x_d b_d - \sum_{k=1}^K z_k c_k \\ &= -(\mathbf{x}^\top \mathbf{W} \mathbf{z} + \mathbf{x}^\top \mathbf{b} + \mathbf{z}^\top \mathbf{c}) \end{aligned} \quad (4.9)$$

$$Z(\boldsymbol{\theta}) = \sum_{\mathbf{x}} \sum_{\mathbf{z}} \exp(-\mathcal{E}(\mathbf{x}, \mathbf{z}; \boldsymbol{\theta})) \quad (4.11)$$

where \mathcal{E} is the energy function, \mathbf{W} is a $D \times K$ weight matrix, \mathbf{b} are the visible bias terms, \mathbf{c} are the hidden bias terms, and $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b}, \mathbf{c})$ are all the parameters. For notational simplicity, we will absorb the bias terms into the weight matrix by adding dummy units $x_0 = 1$ and $z_0 = 1$ and setting $\mathbf{w}_{0,:} = \mathbf{c}$ and $\mathbf{w}_{:,0} = \mathbf{b}$. Note that naively computing $Z(\boldsymbol{\theta})$ takes $O(2^D 2^K)$ time but we can reduce this to $O(\min\{D 2^K, K 2^D\})$ time using the structure of the graph.

When using a binary RBM, the posterior can be computed as follows:

$$p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta}) = \prod_{k=1}^K p(z_k|\mathbf{x}, \boldsymbol{\theta}) = \prod_k \text{Ber}(z_k|\sigma(\mathbf{w}_{:,k}^\top \mathbf{x})) \quad (4.12)$$

By symmetry, one can show that we can generate data given the hidden variables as follows:

$$p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) = \prod_d p(x_d|\mathbf{z}, \boldsymbol{\theta}) = \prod_d \text{Ber}(x_d|\sigma(\mathbf{w}_{d,:}^\top \mathbf{z})) \quad (4.13)$$

We can write this in matrix-vector notation as follows:

$$\mathbb{E}[\mathbf{z}|\mathbf{x}, \boldsymbol{\theta}] = \sigma(\mathbf{W}^\top \mathbf{x}) \quad (4.14)$$

$$\mathbb{E}[\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}] = \sigma(\mathbf{W}\mathbf{z}) \quad (4.15)$$

The weights in \mathbf{W} are called the **generative weights**, since they are used to generate the observations, and the weights in \mathbf{W}^\top are called the **recognition weights**, since they are used to recognize the input.

From Equation 4.12, we see that we activate hidden node k in proportion to how much the input vector \mathbf{x} “looks like” the weight vector $\mathbf{w}_{:,k}$ (up to scaling factors). Thus each hidden node captures certain features of the input, as encoded in its weight vector, similar to a feedforward neural network.

For example, consider an RBM for text models, where \mathbf{x} is a bag of words (i.e., a bit vector over the vocabulary). Let $z_k = 1$ if “topic” k is present in the document. Suppose a document has the topics “sports” and “drugs”. If we “multiply” the predictions of each topic together, the model may give very high probability to the word “doping”, which satisfies both constraints. By contrast, adding together experts can only make the distribution broader. In particular, if we mix together the predictions from “sports” and “drugs”, we might generate words like “cricket” and “addiction”, which come from the union of the two topics, not their intersection.

4.3.2 Categorical RBMs

We can extend the binary RBM to categorical visible variables by using a 1-of- C encoding, where C is the number of states for each x_d . We define a new energy function as follows [Salak07; Salak10softmax]:

$$\mathcal{E}(\mathbf{x}, \mathbf{z}; \boldsymbol{\theta}) \triangleq -\sum_{d=1}^D \sum_{k=1}^K \sum_{c=1}^C x_d^c z_k w_{dk}^c - \sum_{d=1}^D \sum_{c=1}^C x_d^c b_d^c - \sum_{k=1}^K z_k c_k \quad (4.16)$$

The full conditionals are given by

$$p(x_d = c^*|\mathbf{z}, \boldsymbol{\theta}) = \text{softmax}(\{b_d^c + \sum_k z_k w_{dk}^c\}_{c=1}^C)[c^*] \quad (4.17)$$

$$p(z_k = 1|\mathbf{x}, \boldsymbol{\theta}) = \sigma(c_k + \sum_d \sum_c x_d^c w_{dk}^c) \quad (4.18)$$

4.3.3 Gaussian RBMs

We can generalize the model to handle real-valued data. In particular, a **Gaussian RBM** has the following energy function:

$$\mathcal{E}(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) = -\sum_{d=1}^D \sum_{k=1}^K w_{dk} z_k x_d - \frac{1}{2} \sum_{d=1}^D (x_d - b_d)^2 - \sum_{k=1}^K a_k z_k \quad (4.19)$$

The parameters of the model are $\boldsymbol{\theta} = (w_{dk}, a_k, b_d)$. (We have assumed the data is standardized, so we fix the variance to $\sigma^2 = 1$.) Compare this to a Gaussian in canonical or information form (see Main Section 2.3.1.4):

$$\mathcal{N}_c(\mathbf{x}|\boldsymbol{\eta}, \boldsymbol{\Lambda}) \propto \exp(\boldsymbol{\eta}^\top \mathbf{x} - \frac{1}{2} \mathbf{x}^\top \boldsymbol{\Lambda} \mathbf{x}) \quad (4.20)$$

where $\boldsymbol{\eta} = \boldsymbol{\Lambda} \boldsymbol{\mu}$. We see that we have set $\boldsymbol{\Lambda} = \mathbf{I}$, and $\boldsymbol{\eta} = \sum_k z_k \mathbf{w}_{:,k}$. Thus the mean is given by $\boldsymbol{\mu} = \boldsymbol{\Lambda}^{-1} \boldsymbol{\eta} = \sum_k z_k \mathbf{w}_{:,k}$, which is a weighted combination of prototypes. The full conditionals, which are needed for inference and learning, are given by

$$p(x_d|\mathbf{z}, \boldsymbol{\theta}) = \mathcal{N}(x_d|b_d + \sum_k w_{dk} z_k, 1) \quad (4.21)$$

$$p(z_k = 1|\mathbf{x}, \boldsymbol{\theta}) = \sigma \left(c_k + \sum_d w_{dk} x_d \right) \quad (4.22)$$

More powerful models, which make the (co)variance depend on the hidden states, can also be developed [Ranzato10].

4.3.4 RBMs with Gaussian hidden units

If we use Gaussian latent variables and Gaussian visible variables, we get an undirected version of factor analysis (Main Section 28.3.1). Interestingly, this is mathematically equivalent to the standard directed version [Marks01].

If we use Gaussian latent variables and categorical observed variables, we get an undirected version of categorical PCA (Main Section 28.3.5). In [Salak07], this was applied to the Netflix collaborative filtering problem, but was found to be significantly inferior to using binary latent variables, which have more expressive power.

5 Information theory

5.1 Minimizing KL between two Gaussians

5.1.1 Moment projection

In moment projection, we want to solve

$$q = \underset{q}{\operatorname{argmin}} D_{\text{KL}}(p \parallel q) \quad (5.1)$$

We just need to equate the moments of q to the moments of p . For example, suppose $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $q(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mathbf{m}, \text{diag}(\mathbf{v}))$. We have $\mathbf{m} = \boldsymbol{\mu}$ and $v_i = \boldsymbol{\Sigma}_{ii}$. This is the “mode covering” solution.

5.1.2 Information projection

In information projection, we want to solve

$$q = \underset{q}{\operatorname{argmin}} D_{\text{KL}}(q \parallel p) \quad (5.2)$$

For example, suppose

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1}) \quad (5.3)$$

and

$$q(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mathbf{m}, \mathbf{V}) = \mathcal{N}(\mathbf{x}|\mathbf{m}, \text{diag}(\mathbf{v})) \quad (5.4)$$

Below we show that the optimal solution is to set $\mathbf{m} = \boldsymbol{\mu}$ and $v_i = \boldsymbol{\Lambda}_{ii}^{-1}$. This is the “mode seeking” solution.

To derive this result, we write out the KL as follows:

$$D_{\text{KL}}(q \parallel p) = \frac{1}{2} \left(\log \frac{|\boldsymbol{\Sigma}|}{|\mathbf{V}|} - D + (\boldsymbol{\mu} - \mathbf{m})^\top \boldsymbol{\Sigma}(\boldsymbol{\mu} - \mathbf{m}) + \text{tr}(\boldsymbol{\Lambda}\mathbf{V}) \right) \quad (5.5)$$

where D is the dimensionality. To find the optimal \mathbf{m} , we set the derivative to 0 and solve. This gives $\mathbf{m} = \boldsymbol{\mu}$. Plugging in this solution, the objective then becomes

$$J(q) = \frac{1}{2} \left(-\log |\mathbf{V}| - \text{const} + \sum_j [\boldsymbol{\Lambda}_{jj} \mathbf{V}_{jj}] \right) \quad (5.6)$$

1 Defining $\mathbf{V} = \text{diag}(\sigma_i^2)$ we get
2

3

$$4 J(q) = \frac{1}{2} \left(-\sum_j \log \sigma_j^2 - \text{const} + \sum_j [\sigma_j^2 \mathbf{\Lambda}_{jj}] \right) \quad (5.7)$$

5

6

7 Setting $\frac{\partial J(q)}{\partial \sigma_i} = 0$ we get
8

9

$$10 \frac{\partial J(q)}{\partial \sigma_i} = -\frac{1}{\sigma_i} + \sigma_i \mathbf{\Lambda}_{ii} = 0 \quad (5.8)$$

11

12 which gives
13

14 $\sigma_i^{-2} = \mathbf{\Lambda}_{ii}$ (5.9)
15

16 which says that we should match the marginal precisions of the two distributions.
17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

6 Optimization

6.1 Proximal methods

In this section, we discuss a class of optimization algorithms called **proximal methods** that use as their basic subroutine the **proximal operator** of a function, as opposed to its gradient or Hessian. We define this operator below, but essentially it involves solving a convex subproblem.

Compared to gradient methods, proximal method are easier to apply to nonsmooth problems (e.g., with ℓ_1 terms), as well as large scale problems that need to be decomposed and solved in parallel. These methods are widely used in signal and image processing, and in some applications in deep learning (e.g., [Bai2019] uses proximal methods for training quantized DNNs, [Yao2020] uses proximal methods for efficient neural architecture search, [PPO; Wang2019PPO] uses proximal methods for policy gradient optimization, etc.).

Our presentation is based in part on the tutorial in [proximal]. For another good review, see [Polson2015].

6.1.1 Proximal operators

Let $f : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{+\infty\}$ be a convex function, where $f(\mathbf{x}) = \infty$ means the point is infeasible. Let the effective domain of f be the set of feasible points:

$$\text{dom}(f) = \{\mathbf{x} \in \mathbb{R}^n : f(\mathbf{x}) < \infty\} \quad (6.1)$$

The **proximal operator** (also called a **proximal mapping**) of f , denoted $\text{prox}_f(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^n$, is defined by

$$\text{prox}_f(\mathbf{x}) = \underset{\mathbf{z}}{\operatorname{argmin}} \left(f(\mathbf{z}) + \frac{1}{2} \|\mathbf{z} - \mathbf{x}\|_2^2 \right) \quad (6.2)$$

This is a strongly convex function and hence has a unique minimizer. This operator is sketched in Figure 6.1a. We see that points inside the domain move towards the minimum of the function, whereas points outside the domain move to the boundary and then towards the minimum.

For example, suppose f is the indicator function for the convex set \mathcal{C} , i.e.,

$$f(\mathbf{x}) = I_{\mathcal{C}}(\mathbf{x}) = \begin{cases} 0 & \text{if } \mathbf{x} \in \mathcal{C} \\ \infty & \text{if } \mathbf{x} \notin \mathcal{C} \end{cases} \quad (6.3)$$

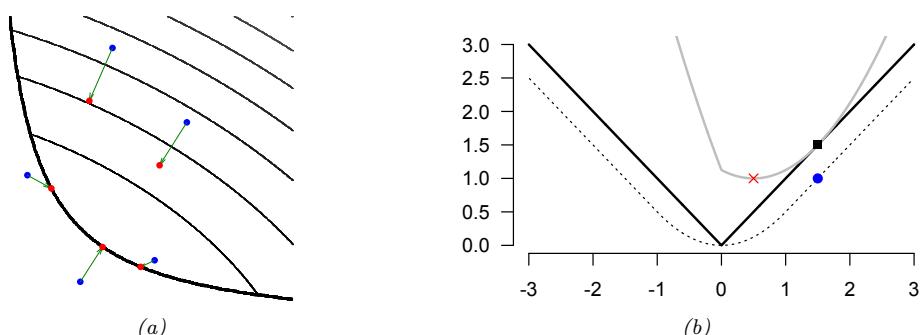


Figure 6.1: (a) Evaluating a proximal operator at various points. The thin lines represent level sets of a convex function; the minimum is at the bottom left. The black line represents the boundary of its domain. Blue points get mapped to red points by the prox operator, so points outside the feasible set get mapped to the boundary, and points inside the feasible set get mapped to closer to the minimum. From Figure 1 of [proximal]. Used with kind permission of Stephen Boyd. (b) Illustration of the Moreau envelope with $\eta = 1$ (dotted line) of the absolute value function (solid black line). See text for details. From Figure 1 of [Polson2015]. Used with kind permission of Nicholas Polson.

In this case, the proximal operator is equivalent to projection onto the set \mathcal{C} :

$$\text{proj}_{\mathcal{C}}(\mathbf{x}) = \underset{\mathbf{z} \in \mathcal{C}}{\operatorname{argmin}} \|\mathbf{z} - \mathbf{x}\|_2 \quad (6.4)$$

We can therefore think of the prox operator as generalized projection.

We will often want to compute the prox operator for a scaled function ηf , for $\eta > 0$, which can be written as

$$\text{prox}_{\eta f}(\mathbf{x}) = \underset{\mathbf{z}}{\operatorname{argmin}} \left(f(\mathbf{z}) + \frac{1}{2\eta} \|\mathbf{z} - \mathbf{x}\|_2^2 \right) \quad (6.5)$$

The solution to the problem in Equation (6.5) the same as the solution to the trust region optimization problem of the form

$$\underset{\mathbf{z}}{\operatorname{argmin}} f(\mathbf{z}) \text{ s.t. } \|\mathbf{z} - \mathbf{x}\|_2 \leq \rho \quad (6.6)$$

for appropriate choices of η and ρ . This the proximal projection minimizes the function while staying close to the current iterate. We give other interpretations of the proximal operator below.

We can generalize the operator by replacing the Euclidean distance with Mahalanobis distance:

$$\text{prox}_{\eta f, \mathbf{A}}(\mathbf{x}) = \underset{\mathbf{z}}{\operatorname{argmin}} \left(f(\mathbf{z}) + \frac{1}{2\eta} (\mathbf{z} - \mathbf{x})^\top \mathbf{A} (\mathbf{z} - \mathbf{x}) \right) \quad (6.7)$$

where \mathbf{A} is a psd matrix.

1

6.1.1.1 Moreau envelope

2 Let us define the following quadratic approximation to the function f as a function of \mathbf{z} , requiring
3 that it touch f at \mathbf{x} :

4

$$f_{\mathbf{x}}^{\eta}(\mathbf{z}) = f(\mathbf{z}) + \frac{1}{2\eta} \|\mathbf{z} - \mathbf{x}\|_2^2 \quad (6.8)$$

5

6 By definition, the location of the minimum of this function is $\mathbf{z}^*(\mathbf{x}) = \operatorname{argmin}_{\mathbf{z}} f_{\mathbf{x}}^{\eta}(\mathbf{z}) = \operatorname{prox}_{\eta f}(\mathbf{x})$.

7 For example, consider approximating the function $f(x) = |x|$ at $x_0 = 1.5$ using $f_{x_0}^1(z) = |z| + \frac{1}{2}(z - x_0)^2$. This is shown in Figure 6.1b: the solid black line is $f(x)$, $x_0 = 1.5$ is the black square, and the light gray line is $f_{x_0}^1(z)$. The proximal projection of x_0 onto f is $\mathbf{z}^*(x_0) = \operatorname{argmin}_z f_{x_0}^1(z) = 0.5$, which is the minimum of the quadratic, shown by the red cross. This proximal point is closer to the minimum of $f(x)$ than the starting point, x_0 .

8 Now let us evaluate the approximation at this proximal point:

9

$$f_{\mathbf{x}}^{\eta}(\mathbf{z}^*(\mathbf{x})) = f(\mathbf{z}^*) + \frac{1}{2\eta} \|\mathbf{z}^* - \mathbf{x}\|_2^2 = \min_{\mathbf{z}} f(\mathbf{z}) + \frac{1}{2\eta} \|\mathbf{z} - \mathbf{x}\|_2^2 \triangleq f^{\eta}(\mathbf{x}) \quad (6.9)$$

10

11 where $f^{\eta}(\mathbf{x})$ is called the **Moreau envelope** of f .

12 For example, in Figure 6.1b, we see that $f_{x_0}^1(z^*) = f_{x_0}^1(0.5) = 1.0$, so $f^1(x_0) = 1.0$. This is shown
13 by the blue circle. The dotted line is the locus of blue points as we vary x_0 , i.e., the Moreau envelope
14 of f .

15 We see that the Moreau envelope is a smooth lower bound on f , and has the same minimum location
16 as f . Furthermore, it has domain \mathbb{R}^n , even when f does not, and it is continuously differentiable,
17 even when f is not. This makes it easier to optimize. For example, the Moreau envelope of $f(r) = |r|$
18 is the **Huber loss** function, which is used in robust regression.

19

6.1.1.2 Prox operator on a linear approximation yields gradient update

20 Suppose we make a linear approximation of f at the current iterate \mathbf{x}_t :

21

$$\hat{f}(\mathbf{x}) = f(\mathbf{x}_t) + \mathbf{g}_t^T(\mathbf{x} - \mathbf{x}_t) \quad (6.10)$$

22

23 where $\mathbf{g}_t = \nabla f(\mathbf{x}_t)$. To compute the prox operator, note that

24

$$\nabla_{\mathbf{z}} \hat{f}_{\mathbf{x}}^{\eta}(\mathbf{z}) = \nabla_{\mathbf{z}} \left[f(\mathbf{x}_t) + \mathbf{g}_t^T(\mathbf{z} - \mathbf{x}) + \frac{1}{2\eta} \|\mathbf{z} - \mathbf{x}_t\|_2^2 \right] = \mathbf{g}_t + \frac{1}{\eta}(\mathbf{z} - \mathbf{x}_t) \quad (6.11)$$

25

26 Solving $\nabla_{\mathbf{z}} \hat{f}_{\mathbf{x}}^{\eta}(\mathbf{z}) = \mathbf{0}$ yields the standard gradient update:

27

$$\operatorname{prox}_{\eta \hat{f}}(\mathbf{x}) = \mathbf{x} - \eta \mathbf{g}_t \quad (6.12)$$

28

29 Thus a prox step is equivalent to a gradient step on a linearized objective.

30

6.1.1.3 Prox operator on a quadratic approximation yields regularized Newton update

31 Now suppose we use a second order approximation at \mathbf{x}_t :

32

$$\hat{f}(\mathbf{x}) = f(\mathbf{x}_t) + \nabla \mathbf{g}_t(\mathbf{x} - \mathbf{x}_t) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_t)^T \mathbf{H}_t(\mathbf{x} - \mathbf{x}_t) \quad (6.13)$$

33

The prox operator for this is

$$\text{prox}_{\eta f}(\mathbf{x}) = \mathbf{x} - (\mathbf{H}_t + \frac{1}{\eta} \mathbf{I})^{-1} \mathbf{g}_t \quad (6.14)$$

6.1.1.4 Prox operator as gradient descent on a smoothed objective

Prox operators are arguably most useful for nonsmooth functions for which we cannot make a Taylor series approximation. Instead, we will optimize the Moreau envelope, which is a smooth approximation.

In particular, from Equation (6.9), we have

$$f^\eta(\mathbf{x}) = f(\text{prox}_{\eta f}(\mathbf{x})) + \frac{1}{2\eta} \|\mathbf{x} - \text{prox}_{\eta f}(\mathbf{x})\|_2^2 \quad (6.15)$$

Hence the gradient of the Moreau envelope is given by

$$\nabla_{\mathbf{x}} f^\eta(\mathbf{x}) = \frac{1}{\eta}(\mathbf{x} - \text{prox}_{\eta f}(\mathbf{x})) \quad (6.16)$$

Thus we can rewrite the prox operator as

$$\text{prox}_{\eta f}(\mathbf{x}) = \mathbf{x} - \eta \nabla f^\eta(\mathbf{x}) \quad (6.17)$$

Thus a prox step is equivalent to a gradient step on the smoothed objective.

6.1.2 Computing proximal operators

In this section, we briefly discuss how to compute proximal operators for various functions that are useful in ML, either as regularizers or constraints. More examples can be found in [proximal; Polson2015].

6.1.2.1 Moreau decomposition

A useful technique for computing some kinds of proximal operators leverages a result known as **Moreau decomposition**, which states that

$$\mathbf{x} = \text{prox}_f(\mathbf{x}) + \text{prox}_{f^*}(\mathbf{x}) \quad (6.18)$$

where f^* is the convex conjugate of f (see Section 6.3).

For example, suppose $f = \|\cdot\|$ is a general norm on \mathbb{R}^D . It can be shown that $f^* = I_{\mathcal{B}}$, where

$$\mathcal{B} = \{\mathbf{x} : \|\mathbf{x}\|^* \leq 1\} \quad (6.19)$$

is the **unit ball** for the **dual norm** $\|\cdot\|^*$, defined by

$$\|\mathbf{z}\|^* = \sup\{\mathbf{z}^\top \mathbf{x} : \|\mathbf{x}\| \leq 1\} \quad (6.20)$$

Hence

$$\text{prox}_{\lambda f}(\mathbf{x}) = \mathbf{x} - \lambda \text{prox}_{f^*/\lambda}(\mathbf{x}/\lambda) = \mathbf{x} - \lambda \text{proj}_{\mathcal{B}}(\mathbf{x}/\lambda) \quad (6.21)$$

Thus there is a close connection between proximal operators of norms and projections onto norm balls that we will leverage below.

6.1.2.2 Projection onto box constraints

Let $\mathcal{C} = \{\mathbf{x} : \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\}$ be a box or hyper-rectangle, imposing lower and upper bounds on each element. (These bounds can be infinite for certain elements if we don't want to constrain values along that dimension.) The projection operator is easy to compute elementwise by simply thresholding at the boundaries:

$$\text{proj}_{\mathcal{C}}(\mathbf{x})_d = \begin{cases} l_d & \text{if } x_k \leq l_k \\ x_d & \text{if } l_k \leq x_k \leq u_k \\ u_d & \text{if } x_k \geq u_k \end{cases} \quad (6.22)$$

For example, if we want to ensure all elements are non-negative, we can use

$$\text{proj}_{\mathcal{C}}(\mathbf{x}) = \mathbf{x}_+ = [\max(x_1, 0), \dots, \max(x_D, 0)] \quad (6.23)$$

6.1.2.3 ℓ_1 norm

Consider the 1-norm $f(\mathbf{x}) = \|\mathbf{x}\|_1$. The proximal projection can be computed componentwise. We can solve each 1d problem as follows:

$$\text{prox}_{\lambda f}(x) = \underset{z}{\operatorname{argmin}} \lambda |z| + \frac{1}{2}(z - x)^2 \quad (6.24)$$

One can show that the solution to this is given by

$$\text{prox}_{\lambda f}(x) = \begin{cases} x - \lambda & \text{if } x \geq \lambda \\ 0 & \text{if } |x| \geq \lambda \\ x + \lambda & \text{if } x \leq \lambda \end{cases} \quad (6.25)$$

This is known as the **soft thresholding operator**, since values less than λ in absolute value are set to 0 (thresholded), but in a differentiable way. This is useful for enforcing sparsity. Note that soft thresholding can be written more compactly as

$$\text{SoftThreshold}_{\lambda}(x) = \text{sign}(x) (|x| - \lambda)_+ \quad (6.26)$$

where $x_+ = \max(x, 0)$ is the positive part of x . In the vector case, we define $\text{SoftThreshold}_{\lambda}(\mathbf{x})$ to be elementwise soft thresholding.

6.1.2.4 ℓ_2 norm

Now consider the ℓ_2 norm $f(\mathbf{x}) = \|\mathbf{x}\|_2 = \sqrt{\sum_{d=1}^D x_d^2}$. The dual norm for this is also the ℓ_2 norm. Projecting onto the corresponding unit ball \mathcal{B} can be done by simply scaling vectors that lie outside the unit sphere:

$$\text{proj}_{\mathcal{B}}(\mathbf{x}) = \begin{cases} \frac{\mathbf{x}}{\|\mathbf{x}\|_2} & \|\mathbf{x}\|_2 > 1 \\ \mathbf{x} & \|\mathbf{x}\|_2 \leq 1 \end{cases} \quad (6.27)$$

Hence by the Moreau decomposition we have

$$\text{prox}_{\lambda f}(\mathbf{x}) = (1 - \lambda/\|\mathbf{x}\|_2)_+ \mathbf{x} = \begin{cases} (1 - \frac{\lambda}{\|\mathbf{x}\|_2}) \mathbf{x} & \text{if } \|\mathbf{x}\|_2 \geq \lambda \\ 0 & \text{otherwise} \end{cases} \quad (6.28)$$

This will set the whole vector to zero if its ℓ_2 norm is less than λ . This is therefore called **block soft thresholding**.

6.1.2.5 Squared ℓ_2 norm

Now consider using the *squared* ℓ_2 norm (scaled by 0.5), $f(\mathbf{x}) = \frac{1}{2}\|\mathbf{x}\|_2^2 = \frac{1}{2}\sum_{d=1}^D x_d^2$. One can show that

$$\text{prox}_{\lambda f}(\mathbf{x}) = \frac{1}{1 + \lambda} \mathbf{x} \quad (6.29)$$

This reduces the magnitude of the \mathbf{x} vector, but does not enforce sparsity. It is therefore called the **shrinkage operator**.

More generally, if $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x} + \mathbf{b}^\top \mathbf{x} + c$ is a quadratic, with \mathbf{A} being positive definite, then

$$\text{prox}_{\lambda f}(\mathbf{x}) = (\mathbf{I} + \lambda \mathbf{A})^{-1}(\mathbf{x} - \lambda \mathbf{b}) \quad (6.30)$$

A special case of this is if f is affine, $f(\mathbf{x}) = \mathbf{b}^\top \mathbf{x} + c$. Then we have $\text{prox}_{\lambda f}(\mathbf{x}) = \mathbf{x} - \lambda \mathbf{b}$. We saw an example of this in Equation (6.12).

6.1.2.6 Nuclear norm

The **nuclear norm**, also called the **trace norm**, of an $m \times n$ matrix \mathbf{A} is the ℓ_1 norm of its singular values: $f(\mathbf{A}) = \|\mathbf{A}\|_* = \|\boldsymbol{\sigma}\|_1$. Using this as a regularizer can result in a low rank matrix. The proximal operator for this is defined by

$$\text{prox}_{\lambda f}(\mathbf{A}) = \sum_i (\sigma_i - \lambda)_+ \mathbf{u}_i \mathbf{v}_i^\top \quad (6.31)$$

where $\mathbf{A} = \sum_i \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$ is the SVD of \mathbf{A} . This operation is called **singular value thresholding**.

6.1.2.7 Projection onto positive definite cone

Consider the cone of positive semidefinite matrices \mathcal{C} , and let $f(\mathbf{A}) = I_{\mathcal{C}}(\mathbf{A})$ be the indicator function. The proximal operator corresponds to projecting \mathbf{A} onto the cone. This can be computed using

$$\text{proj}_{\mathcal{C}}(\mathbf{A}) = \sum_i (\lambda_i)_+ \mathbf{u}_i \mathbf{u}_i^\top \quad (6.32)$$

where $\sum_i \lambda_i \mathbf{u}_i \mathbf{u}_i^\top$ is the eigenvalue decomposition of \mathbf{A} . This is useful for optimizing psd matrices.

6.1.2.8 Projection onto probability simplex

Let $\mathcal{C} = \{\mathbf{x} : \mathbf{x} \geq 0, \sum_{d=1}^D x_d = 1\} = \mathbb{S}_D$ be the probability simplex in D dimensions. We can project onto this using

$$\text{proj}_{\mathcal{C}}(\mathbf{x}) = (\mathbf{x} - \nu \mathbf{1})_+ \quad (6.33)$$

The value $\nu \in \mathbb{R}$ must be found using bisection search. See [proximal] for details. This is useful for optimizing over discrete probability distributions.

6.1.3 Proximal point methods (PPM)

A **proximal point method** (PPM), also called a **proximal minimization algorithm**, iteratively applies the following update:

$$\boldsymbol{\theta}_{t+1} = \text{prox}_{\eta_t \mathcal{L}}(\boldsymbol{\theta}_t) = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \mathcal{L}(\boldsymbol{\theta}) + \frac{1}{2\eta_t} \|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|_2^2 \quad (6.34)$$

where we assume $\mathcal{L} : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{+\infty\}$ is a closed proper convex function. The advantage of this method over minimizing \mathcal{L} directly is that sometimes adding quadratic regularization can improve the conditioning of the problem, and hence speed convergence.

6.1.3.1 Gradient descent is PPM on a linearized objective

We now show that SGD is PPM on a linearized objective. To see this, let the approximation at the current iterate be

$$\hat{\mathcal{L}}_t(\boldsymbol{\theta}) = \tilde{\mathcal{L}}_t(\boldsymbol{\theta}_t) + \mathbf{g}_t^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_t) \quad (6.35)$$

where $\mathbf{g}_t = \nabla_{\boldsymbol{\theta}} \tilde{\mathcal{L}}_t(\boldsymbol{\theta}_t)$. Now we compute a proximal update to this approximate objective:

$$\boldsymbol{\theta}_{t+1} = \text{prox}_{\eta_t \hat{\mathcal{L}}_t}(\boldsymbol{\theta}_t) = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \hat{\mathcal{L}}_t(\boldsymbol{\theta}) + \frac{1}{2\eta_t} \|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|_2^2 \quad (6.36)$$

We have

$$\nabla_{\boldsymbol{\theta}} \left[\tilde{\mathcal{L}}_t(\boldsymbol{\theta}_t) + \mathbf{g}_t^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_t) + \frac{1}{2\eta_t} \|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|_2^2 \right] = \mathbf{g}_t + \frac{1}{\eta_t} (\boldsymbol{\theta} - \boldsymbol{\theta}_t) \quad (6.37)$$

Setting the gradient to zero yields the SGD step $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \mathbf{g}_t$.

6.1.3.2 Beyond linear approximations (truncated ADAGRAD)

Sometimes we can do better than just using PPM with a linear approximation to the objective, at essentially no extra cost, as pointed out in [Asi2019siam; Asi2019aistats; Asi2019pnas]. For example, suppose we know a lower bound on the loss, $\tilde{\mathcal{L}}_t^{\min} = \min_{\boldsymbol{\theta}} \tilde{\mathcal{L}}_t(\boldsymbol{\theta})$. For example, when using squared error, or cross-entropy loss for discrete labels, we have $\tilde{\mathcal{L}}_t(\boldsymbol{\theta}) \geq 0$. Let us therefore define the **truncated model**

$$\hat{\tilde{\mathcal{L}}}_t(\boldsymbol{\theta}) = \max \left(\tilde{\mathcal{L}}_t(\boldsymbol{\theta}) + \mathbf{g}_t^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_t), \tilde{\mathcal{L}}_t^{\min} \right) \quad (6.38)$$

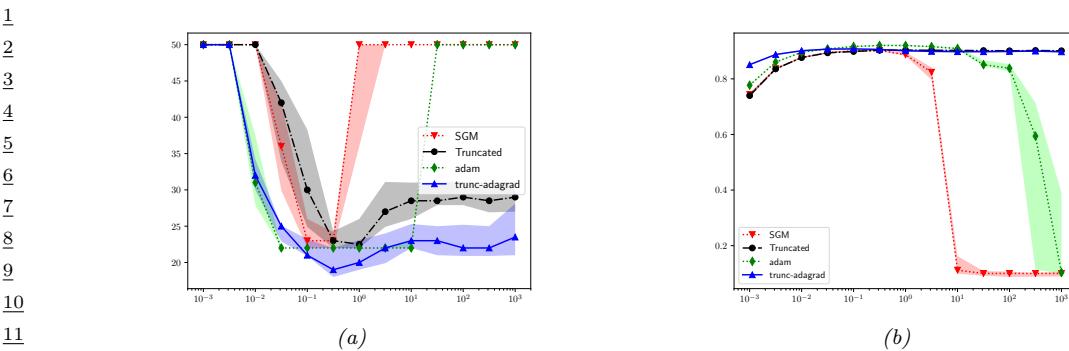


Figure 6.2: Illustration of the benefits of using a lower-bounded loss function when training a resnet-128 CNN on the CIFAR10 image classification dataset. The curves are as follows: SGM (stochastic gradient method, i.e., SGD), ADAM, truncated SGD and truncated ADAGRAD. (a) Time to reach an error that satisfies $\mathcal{L}(\theta_t) - \mathcal{L}(\theta^*) \leq \epsilon$ vs initial learning rate η_0 . (b) Top-1 accuracy after 50 epochs vs η_0 . The lines represent median performance across 50 random restarts, and shading represents 90% confidence intervals. From Figure 4 of [Asi2019pnas]. Used with kind permission of Hilal Asi.

We can further improve things by replacing the Euclidean norm with a scaled Euclidean norm, where the diagonal scaling matrix is given by $\mathbf{A}_t = \text{diag}(\sum_{i=1}^t \mathbf{g}_i \mathbf{g}_i^\top)^{\frac{1}{2}}$, as in ADAGRAD [adagrad]. If $\tilde{\mathcal{L}}_t^{\min} = 0$, the resulting proximal update becomes

$$\theta_{t+1} = \operatorname{argmin}_{\theta} \left[\tilde{\mathcal{L}}_t(\theta_t) + \mathbf{g}_t^\top (\theta - \theta_t) \right]_+ + \frac{1}{2\eta_t} (\theta - \theta_t)^\top \mathbf{A}_t (\theta - \theta_t) \quad (6.39)$$

$$= \theta_t - \min(\eta_t, \frac{\tilde{\mathcal{L}}_t(\theta_t)}{\mathbf{g}_t^\top \mathbf{A}_t^{-1} \mathbf{g}_t}) \mathbf{g}_t \quad (6.40)$$

Thus the update is like a standard SGD update, but we truncate the learning rate if it is too big.¹

[Asi2019pnas] call this **truncated AdaGrad**. Furthermore, they prove optimizing this truncated linear approximation (with or without AdaGrad weighting), instead of the standard linear approximation used by gradient descent, can result in significant benefits. In particular, it is guaranteed to be stable (under certain technical conditions) for any learning rate, whereas standard GD can “blow up”, even for convex problems.

Figure 6.2 shows the benefits of this approach when training a resnet-128 CNN (Main Section 16.2.4) on the CIFAR10 image classification dataset. For SGD and the truncated proximal method, the learning rate is decayed using $\eta_t = \eta_0 t^{-\beta}$ with $\beta = 0.6$. For ADAM and truncated ADAGRAD, the learning rate is set to $\eta_t = \eta_0$, since we use diagonal scaling. We see that both truncated methods (regular and ADAGRAD version) have good performance for a much broader range of initial learning rate η_0 compared to SGD or Adam.

1. One way to derive this update (suggested by Hilal Asi) is to do case analysis on the value of $\hat{\mathcal{L}}_t(\theta_{t+1})$, where $\hat{\mathcal{L}}_t$ is the truncated linear model. If $\hat{\mathcal{L}}_t(\theta_{t+1}) > 0$, then setting the gradient to zero yields the usual SGD update, $\theta_{t+1} = \theta_t - \eta_t \mathbf{g}_t$. (We assume $\mathbf{A}_t = \mathbf{I}$ for simplicity.) Otherwise we must have $\hat{\mathcal{L}}_t(\theta_{t+1}) = 0$. But we know that $\theta_{t+1} = \theta_t - \lambda \mathbf{g}_t$ for some λ , so we solve $\hat{\mathcal{L}}_t(\theta_t - \lambda \mathbf{g}_t) = 0$ to get $\lambda = \hat{\mathcal{L}}_t(\theta_t) / \|\mathbf{g}_t\|_2^2$.

6.1.3.3 Stochastic and incremental PPM

PPM can be extended to the stochastic setting, where the goal is to optimize $\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{q(\mathbf{z})} [\tilde{\mathcal{L}}(\boldsymbol{\theta}, \mathbf{z})]$, by using the following stochastic update:

$$\boldsymbol{\theta}_{t+1} = \text{prox}_{\eta_t \tilde{\mathcal{L}}_t}(\boldsymbol{\theta}_t) = \operatorname{argmin}_{\boldsymbol{\theta}} \tilde{\mathcal{L}}_t(\boldsymbol{\theta}) + \frac{1}{2\eta_t} \|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|_2^2 \quad (6.41)$$

where $\tilde{\mathcal{L}}_t(\boldsymbol{\theta}) = \tilde{\mathcal{L}}(\boldsymbol{\theta}, \mathbf{z}_t)$ and $\mathbf{z}_t \sim q$. The resulting method is known as **stochastic PPM** (see e.g., [Patrascu2018]). If q is the empirical distribution associated with a finite-sum objective, this is called the **incremental proximal point method** [Bertsekas2015]. It is often more stable than SGD.

In the case where the cost function is a linear least squares problem, one can show [Akyildiz2018] that the IPPM is equivalent to the Kalman filter (Main Section 8.2.2), where the posterior mean is equal to the current parameter estimate, $\boldsymbol{\theta}_t$. The advantage of this probabilistic perspective is that it also gives us the posterior covariance, which can be used to define a variable-metric distance function inside the prox operator, as in Equation (6.7). We can extend this to nonlinear problems using the extended KF (Main Section 8.3.2).

6.1.4 Mirror descent

In this section, we discuss **mirror descent** [Nemirovski1983; Beck03], which is like gradient descent, but can leverage non-Euclidean geometry to potentially speed up convergence, or enforce certain constraints.

Suppose we replace the Euclidean distance term $\|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|_2^2$ in PPM with a Bregman divergence (Main Section 5.1.10), defined as

$$D_h(\mathbf{x}, \mathbf{y}) = h(\mathbf{x}) - [h(\mathbf{y}) + \nabla h(\mathbf{y})^\top (\mathbf{x} - \mathbf{y})] \quad (6.42)$$

where $h(\mathbf{x})$ is a strongly convex function. Combined with our linear approximation to the objective, this gives the following update:

$$\boldsymbol{\theta}_{t+1} = \operatorname{argmin}_{\boldsymbol{\theta}} \hat{\mathcal{L}}(\boldsymbol{\theta}) + \frac{1}{\eta_t} D_h(\boldsymbol{\theta}, \boldsymbol{\theta}_t) \quad (6.43)$$

$$= \operatorname{argmin}_{\boldsymbol{\theta}} \eta_t \mathbf{g}_t^\top \boldsymbol{\theta} + D_h(\boldsymbol{\theta}, \boldsymbol{\theta}_t) \quad (6.44)$$

This is known as **mirror descent** [Nemirovski1983; Beck03]. This can easily be extended to the stochastic setting in the obvious way.

One can show that natural gradient descent (Main Section 6.4) is a form of mirror descent [Raskutti2015]. More precisely, mirror descent in the mean parameter space is equivalent to natural gradient descent in the canonical parameter space.

6.1.5 Proximal gradient method

We are often interested in optimizing a composite objective of the form

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_s(\boldsymbol{\theta}) + \mathcal{L}_r(\boldsymbol{\theta}) \quad (6.45)$$

where \mathcal{L}_s is convex and differentiable (smooth), and \mathcal{L}_r is convex but not necessarily differentiable (i.e., it may be non-smooth or “rough”). For example, \mathcal{L}_r might be an ℓ_1 norm regularization term, and \mathcal{L}_s might be the NLL for linear regression (see Section 6.1.5.1).

The **proximal gradient method** is the following update:

$$\theta_{t+1} = \text{prox}_{\eta_t \mathcal{L}_r}(\theta_t - \eta_t \nabla \mathcal{L}_s(\theta_t)) \quad (6.46)$$

If $\mathcal{L}_r = I_C$, this is equivalent to **projected gradient descent**. If $\mathcal{L}_r = 0$, this is equivalent to gradient descent. If $\mathcal{L}_s = 0$, this is equivalent to a proximal point method.

We can create a version of the proximal gradient method with **Nesterov acceleration** as follows:

$$\tilde{\theta}_{t+1} = \theta_t + \beta_t(\theta_t - \theta_{t-1}) \quad (6.47)$$

$$\theta_{t+1} = \text{prox}_{\eta_t \mathcal{L}_r}(\tilde{\theta}_{t+1} - \eta_t \nabla \mathcal{L}_s(\tilde{\theta}_{t+1})) \quad (6.48)$$

See e.g., [Tseng2008].

Now we consider the stochastic case, where $\mathcal{L}_s(\theta) = \mathbb{E}[\mathcal{L}_s(\theta, z)]$. (We assume \mathcal{L}_r is deterministic.)

In this setting, we can use the following stochastic update:

$$\theta_{t+1} = \text{prox}_{\eta_t \mathcal{L}_r}(\theta_t - \eta_t \nabla \mathcal{L}_s(\theta_t, z_t)) \quad (6.49)$$

where $z_t \sim q$. This is called the **stochastic proximal gradient method**. If q is the empirical distribution, this is called the **incremental proximal gradient method** [Bertsekas2015]. Both methods can also be accelerated (see e.g., [Nitanda2014]).

If \mathcal{L}_s is not convex, we can compute a locally convex approximation, as in Section 6.1.3.2. (We assume \mathcal{L}_r remains convex.) The accelerated version of this is studied in [Li2015apgm]. In the stochastic case, we can similarly make a locally convex approximation to $\mathcal{L}_s(\theta, z)$. This is studied in [Reddi2016; Li2018prox]. An EKF interpretation in the incremental case (where $q = p_D$) is given in [Akyildiz2019].

6.1.5.1 Example: Iterative soft-thresholding algorithm (ISTA) for sparse linear regression

Suppose we are interested in fitting a linear regression model with a sparsity-promoting prior on the weights, as in the lasso model (Main Section 15.2.6). One way to implement this is to add the ℓ_1 -norm of the parameters as a (non-smooth) penalty term, $\mathcal{L}_r(\theta) = \|\theta\|_1 = \sum_{d=1}^D |\theta_d|$. Thus the objective is

$$\mathcal{L}(\theta) = \mathcal{L}_s(\theta) + \mathcal{L}_r(\theta) = \frac{1}{2} \|\mathbf{X}\theta - \mathbf{y}\|_2^2 + \lambda \|\theta\|_1 \quad (6.50)$$

The proximal gradient descent update can be written as

$$\theta_{t+1} = \text{SoftThreshold}_{\eta_t \lambda}(\theta_t - \eta_t \nabla \mathcal{L}_s(\theta_t)) \quad (6.51)$$

where the soft thresholding operator (Equation (6.26)) is applied elementwise, and $\nabla \mathcal{L}_s(\theta) = \mathbf{X}^\top(\mathbf{X}\theta - \mathbf{y})$. This is called the **iterative soft thresholding algorithm** or **ISTA** [Daubechies2004; Donoho1995]. If we combine this with Nesterov acceleration, we get the method known as “fast ISTA” or **FISTA** [Beck2009], which is widely used to fit sparse linear models.

6.1.6 Alternating direction method of multipliers (ADMM)

Consider the problem of optimizing $\mathcal{L}(\mathbf{x}) = \mathcal{L}_s(\mathbf{x}) + \mathcal{L}_r(\mathbf{x})$ where now both \mathcal{L}_s and \mathcal{L}_r may be non-smooth (but we assume both are convex). We may want to optimize these problems independently (e.g., so we can do it in parallel), but need to ensure the solutions are consistent.

One way to do this is by using the **variable splitting trick** combined with constrained optimization:

$$\text{minimize } \mathcal{L}_s(\mathbf{x}) + \mathcal{L}_r(\mathbf{z}) \text{ s.t. } \mathbf{x} - \mathbf{z} = \mathbf{0} \quad (6.52)$$

This is called **consensus form**.

The corresponding **augmented Langragian** is given by

$$L_\rho(\mathbf{x}, \mathbf{z}, \mathbf{y}) = \mathcal{L}_s(\mathbf{x}) + \mathcal{L}_r(\mathbf{z}) + \mathbf{y}^\top (\mathbf{x} - \mathbf{z}) + \frac{\rho}{2} \|\mathbf{x} - \mathbf{z}\|_2^2 \quad (6.53)$$

where $\rho > 0$ is the penalty strength, and $\mathbf{y} \in \mathbb{R}^n$ are the dual variables associated with the consistency constraint. We can now perform the following block coordinate descent updates:

$$\mathbf{x}_{t+1} = \operatorname{argmin}_{\mathbf{x}} L_\rho(\mathbf{x}, \mathbf{z}_t, \mathbf{y}_t) \quad (6.54)$$

$$\mathbf{z}_{t+1} = \operatorname{argmin}_{\mathbf{z}} L_\rho(\mathbf{x}_{t+1}, \mathbf{z}, \mathbf{y}_t) \quad (6.55)$$

$$\mathbf{y}_{t+1} = \mathbf{y}_t + \rho(\mathbf{x}_{t+1} - \mathbf{z}_{t+1}) \quad (6.56)$$

We see that the dual variable is the (scaled) running average of the consensus errors.

Inserting the definition of $L_\rho(\mathbf{x}, \mathbf{z}, \mathbf{y})$ gives us the following more explicit update equations:

$$\mathbf{x}_{t+1} = \operatorname{argmin}_{\mathbf{x}} \left(\mathcal{L}_s(\mathbf{x}) + \mathbf{y}_t^\top \mathbf{x} + \frac{\rho}{2} \|\mathbf{x} - \mathbf{z}_t\|_2^2 \right) \quad (6.57)$$

$$\mathbf{z}_{t+1} = \operatorname{argmin}_{\mathbf{z}} \left(\mathcal{L}_r(\mathbf{z}) - \mathbf{y}_t^\top \mathbf{z} + \frac{\rho}{2} \|\mathbf{x}_{t+1} - \mathbf{z}\|_2^2 \right) \quad (6.58)$$

If we combine the linear and quadratic terms, we get

$$\mathbf{x}_{t+1} = \operatorname{argmin}_{\mathbf{x}} \left(\mathcal{L}_s(\mathbf{x}) + \frac{\rho}{2} \|\mathbf{x} - \mathbf{z}_t + (1/\rho)\mathbf{y}_t\|_2^2 \right) \quad (6.59)$$

$$\mathbf{z}_{t+1} = \operatorname{argmin}_{\mathbf{z}} \left(\mathcal{L}_r(\mathbf{z}) + \frac{\rho}{2} \|\mathbf{x}_{t+1} - \mathbf{z} - (1/\rho)\mathbf{y}_t\|_2^2 \right) \quad (6.60)$$

Finally, if we define $\mathbf{u}_t = (1/\rho)\mathbf{y}_t$ and $\lambda = 1/\rho$, we can now write this in a more general way:

$$\mathbf{x}_{t+1} = \operatorname{prox}_{\lambda \mathcal{L}_s}(\mathbf{z}_t - \mathbf{u}_t) \quad (6.61)$$

$$\mathbf{z}_{t+1} = \operatorname{prox}_{\lambda \mathcal{L}_r}(\mathbf{x}_{t+1} + \mathbf{u}_t) \quad (6.62)$$

$$\mathbf{u}_{t+1} = \mathbf{u}_t + \mathbf{x}_{t+1} - \mathbf{z}_{t+1} \quad (6.63)$$

This is called the **alternating direction method of multipliers** or **ADMM** algorithm. The advantage of this method is that the different terms in the objective (along with any constraints they may have) are handled completely independently, allowing different solvers to be used. Furthermore, the method can be extended to the stochastic setting as shown in [Zhong2014].

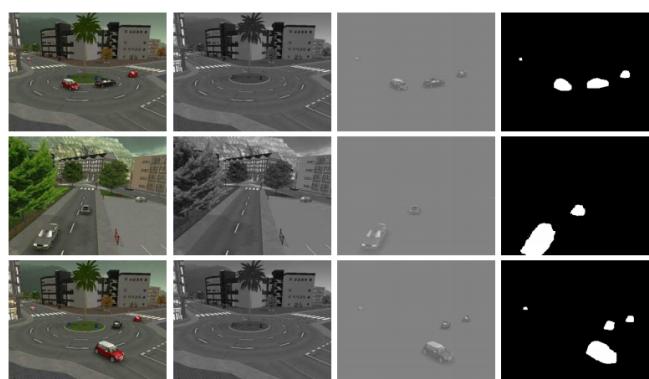


Figure 6.3: Robust PCA applied to some frames from a surveillance video. First column is input image. Second column is low-rank background model. Third model is sparse foreground model. Last column is derived foreground mask. From Figure 1 of [Bouwmans2017]. Used with kind permission of Thierry Bouwmans.

6.1.6.1 Example: robust PCA

In this section, we give an example of ADMM from [proximal].

Consider the following **matrix decomposition problem**:

$$\underset{\mathbf{X}_{1:J}}{\text{minimize}} \sum_{j=1}^J \gamma_j \phi_j(\mathbf{X}_j) \quad \text{s.t.} \quad \sum_{j=1}^J \mathbf{X}_j = \mathbf{A} \quad (6.64)$$

where $\mathbf{A} \in \mathbb{R}^{m \times n}$ is a given data matrix, $\mathbf{X}_j \in \mathbb{R}^{m \times n}$ are the optimization variables, and $\gamma_j > 0$ are trade-off parameters.

For example, suppose we want to find a good least squares approximation to \mathbf{A} as a sum of a low rank matrix plus a sparse matrix. This is called **robust PCA** [Candes2011], since the sparse matrix can handle the small number of outliers that might otherwise cause the rank of the approximation to be high. The method is often used to decompose surveillance videos into a low rank model for the static background, and a sparse model for the dynamic foreground objects, such as moving cars or people, as illustrated in Figure 6.3. (See e.g., [Bouwmans2017] for a review.) RPCA can also be used to remove small “outliers”, such as specularities and shadows, from images of faces, to improve face recognition.

We can formulate robust PCA as the following optimization problem:

$$\underset{\mathbf{L}, \mathbf{S}}{\text{minimize}} \|\mathbf{A} - (\mathbf{L} + \mathbf{S})\|_F^2 + \gamma_L \|\mathbf{L}\|_* + \gamma_S \|\mathbf{S}\|_1 \quad (6.65)$$

which is a sparse plus low rank decomposition of the observed data matrix. We can reformulate this to match the form of a canonical matrix decomposition problem by defining $\mathbf{X}_1 = \mathbf{L}$, $\mathbf{X}_2 = \mathbf{S}$ and $\mathbf{X}_3 = \mathbf{A} - (\mathbf{X}_1 + \mathbf{X}_2)$, and then using these loss functions:

$$\phi_1(\mathbf{X}_1) = \|\mathbf{X}_1\|_*, \quad \phi_2(\mathbf{X}_2) = \|\mathbf{X}_2\|_1, \quad \phi_3(\mathbf{X}_3) = \|\mathbf{X}_3\|_F^2 \quad (6.66)$$

We can tackle such matrix decomposition problems using ADMM, where we use the split $\mathcal{L}_s(\mathbf{X}) = \sum_j \gamma_j \phi_j(\mathbf{X}_j)$ and $\mathcal{L}_r(\mathbf{X}) = I_C(\mathbf{X})$, where $\mathbf{X} = (\mathbf{X}_1, \dots, \mathbf{X}_J)$ and $C = \{\mathbf{X}_{1:J} : \sum_{j=1}^J \mathbf{X}_j = \mathbf{A}\}$. The

overall algorithm becomes

$$\mathbf{X}_{j,t+1} = \text{prox}_{\eta_t \phi_j}(\mathbf{X}_{j,t} - \bar{\mathbf{X}}_t + \frac{1}{N} \mathbf{A} - \mathbf{U}_t) \quad (6.67)$$

$$\mathbf{U}_{t+1} = \mathbf{U}_t + \bar{\mathbf{X}}_{t+1} - \frac{1}{J} \mathbf{A} \quad (6.68)$$

where $\bar{\mathbf{X}}$ is the elementwise average of $\mathbf{X}_1, \dots, \mathbf{X}_J$. Note that the \mathbf{X}_j can be updated in parallel.

Projection onto the ℓ_1 norm is discussed in Section 6.1.2.3, projection onto the nuclear norm is discussed in Section 6.1.2.6. projection onto the squared Frobenius norm is the same as projection onto the squared Euclidean norm discussed in Section 6.1.2.5, and projection onto the constraint set $\sum_j \mathbf{X}_j = \mathbf{A}$ can be done using the averaging operator:

$$\text{proj}_{\mathcal{C}}(\mathbf{X}_1, \dots, \mathbf{X}_J) = (\mathbf{X}_1, \dots, \mathbf{X}_J) - \bar{\mathbf{X}} + \frac{1}{J} \mathbf{A} \quad (6.69)$$

An alternative to using ℓ_1 minimization in the inner loop is to use hard thresholding [Cherapanamjeri2017]. Although not convex, this method can be shown to converge to the global optimum, and is much faster.

It is also possible to formulate a non-negative version of robust PCA. Even though NRPCA is not a convex problem, it is possible to find the globally optimal solution [Fattah2018jmlr; Anderson2019].

6.2 Dynamic programming

Dynamic programming is a way to efficiently find the globally optimal solution to certain kinds of optimization problems. The key requirement is that the optimal solution be expressed in terms of the optimal solution to smaller subproblems, which can be reused many times. Note that DP is more of an algorithm “family” rather than a specific algorithm. We give some examples below.

6.2.1 Example: computing Fibonacci numbers

Consider the problem of computing **Fibonacci numbers**, defined via the recursive equation

$$F_i = F_{i-1} + F_{i-2} \quad (6.70)$$

with base cases $F_0 = F_1 = 1$. Thus we have that $F_2 = 2$, $F_3 = 3$, $F_4 = 5$, $F_5 = 8$, etc. A simple **recursive** algorithm to compute the first n Fibonacci numbers is shown in Algorithm 6.1. Unfortunately, this takes exponential time. For example, evaluating $\text{fib}(5)$ proceeds as follows:

$$F_5 = F_4 + F_3 \quad (6.71)$$

$$= (F_3 + F_2) + (F_2 + F_1) \quad (6.72)$$

$$= ((F_2 + F_1) + (F_1 + F_0)) + ((F_1 + F_0) + F_1) \quad (6.73)$$

$$= (((F_1 + F_0) + F_1) + (F_1 + F_0))((F_1 + F_0) + F_1) \quad (6.74)$$

We see that there is a lot of repeated computation. For example, $\text{fib}(2)$ is computed 3 times. One way to improve the efficiency is to use **memoization**, which means memorizing each function value that is computed. This will result in a linear time algorithm. However, the overhead involved can be high.

It is usually preferable to try to solve the problem **bottom up**, solving small subproblems first, and then using their results to help solve larger problems later. A simple way to do this is shown in Algorithm 6.2.

Algorithm 6.1: Fibonacci numbers, top down

```

1 function fib(n)
2 if n = 0 or n = 1 then
3   return 1
4 else
5   return (fib(n - 1) + fib(n - 2))

```

Algorithm 6.2: Fibonacci numbers, bottom up

```

1 function fib(n)
2 F0 := 1, F1 := 2
3 for i = 2, ..., n do
4   Fi := Fi-1 + Fi-2
5 return Fn

```

6.2.2 ML examples

There are many applications of DP to ML problems, which we discuss elsewhere in this book. These include the forwards-backwards algorithm for inference in HMMs (Main Section 9.2.3), the Viterbi algorithm for MAP sequence estimation in HMMs (Main Section 9.2.6), inference in more general graphical models (Section 9.2), reinforcement learning (Main Section 34.6), etc.

6.3 Conjugate duality

In this section, we briefly discuss **conjugate duality**, which is a useful way to construct linear lower bounds on non-convex functions. We follow the presentation of [BishopBook].

6.3.1 Introduction

Consider an arbitrary continuous function $f(\mathbf{x})$, and suppose we create a linear lower bound on it of the form

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) \triangleq \boldsymbol{\lambda}^\top \mathbf{x} - f^*(\boldsymbol{\lambda}) \leq f(\mathbf{x}) \quad (6.75)$$

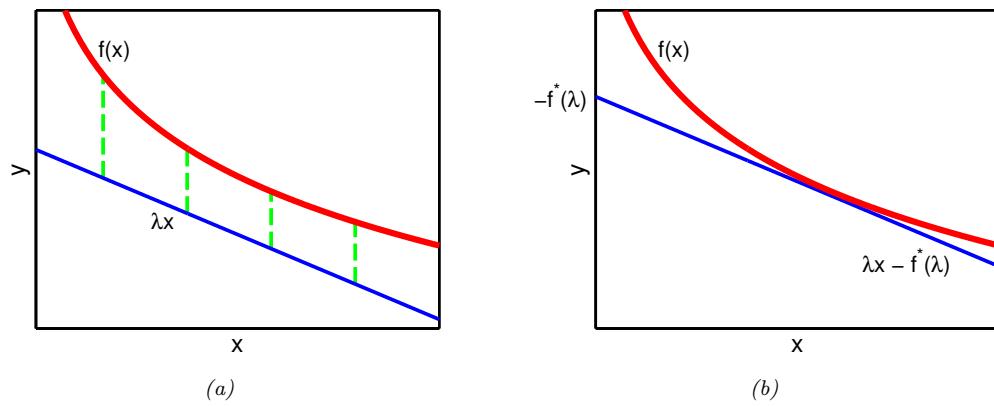


Figure 6.4: Illustration of a conjugate function. Red line is original function $f(x)$, and the blue line is a linear lower bound λx . To make the bound tight, we find the x where $\nabla f(x)$ is parallel to λ , and slide the line up to touch there; the amount we slide up is given by $f^*(\lambda)$. Adapted from Figure 10.11 of [BishopBook].

where λ is the slope, which we choose, and $f^*(\lambda)$ is the intercept, which we solve for below. See Figure 6.4(a) for an illustration.

For a fixed λ , we can find the point \mathbf{x}_λ where the lower bound is tight by “sliding” the line upwards until it touches the curve at \mathbf{x}_λ , as shown in Figure 6.4(b). At \mathbf{x}_λ , we minimize the distance between the function and the lower bound:

$$\mathbf{x}_\lambda \triangleq \underset{\mathbf{x}}{\operatorname{argmin}} f(\mathbf{x}) - \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = \underset{\mathbf{x}}{\operatorname{argmin}} f(\mathbf{x}) - \boldsymbol{\lambda}^\top \mathbf{x} \quad (6.76)$$

Since the bound is tight at this point, we have

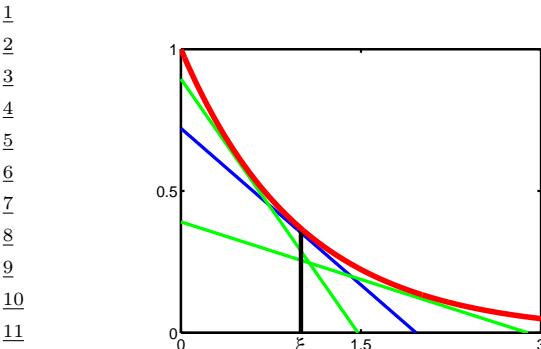
$$f(\mathbf{x}_\lambda) = \mathcal{L}(\mathbf{x}_\lambda, \boldsymbol{\lambda}) = \boldsymbol{\lambda}^\top \mathbf{x}_\lambda - f^*(\boldsymbol{\lambda}) \quad (6.77)$$

and hence

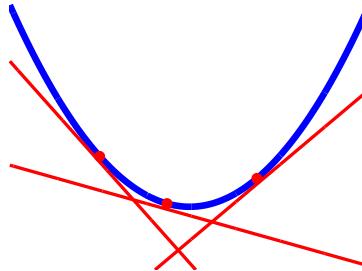
$$f^*(\boldsymbol{\lambda}) = \boldsymbol{\lambda}^\top \mathbf{x}_\lambda - f(\mathbf{x}_\lambda) = \max_{\mathbf{x}} \boldsymbol{\lambda}^\top \mathbf{x} - f(\mathbf{x}) \quad (6.78)$$

The function f^* is called the **conjugate** of f , also known as the **Fenchel transform** of f . For the special case of differentiable f , f^* is called the **Legendre transform** of f .

One reason conjugate functions are useful is that they can be used to create convex lower bounds to non-convex functions. That is, we have $\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) \leq f(\mathbf{x})$, with equality at $\mathbf{x} = \mathbf{x}_\lambda$, for any function $f : \mathbb{R}^D \rightarrow \mathbb{R}$. For any given \mathbf{x} , we can optimize over $\boldsymbol{\lambda}$ to make the bound as tight as possible, giving us a fixed function $\mathcal{L}(\mathbf{x})$; this is called a **variational approximation**. We can then try to maximize this lower bound wrt \mathbf{x} instead of maximizing $f(\mathbf{x})$. This method is used extensively in approximate Bayesian inference, as we discuss in Main Chapter 10.



(a)



(b)

Figure 6.5: (a) The red curve is $f(x) = e^{-x}$ and the colored lines are linear lower bounds. Each lower bound of slope λ is tangent to the curve at the point $x_\lambda = -\log(-\lambda)$, where $f(x_\lambda) = e^{\log(-\lambda)} = -\lambda$. For the blue curve, this occurs at $x_\lambda = \xi$. Adapted from Figure 10.10 of [BishopBook]. Generated by `opt_lower_bound.ipynb`. (b) For a convex function $f(x)$, its epigraph can be represented as the intersection of half-spaces defined by linear lower bounds of the form $f^\dagger(\lambda)$. Adapted from Figure 13 of [Jaakkola99].

6.3.2 Example: exponential function

Let us consider an example. Suppose $f(x) = e^{-x}$, which is convex. Consider a linear lower bound of the form

$$\mathcal{L}(x, \lambda) = \lambda x - f^\dagger(\lambda) \quad (6.79)$$

where the conjugate function is given by

$$f^\dagger(\lambda) = \max_x \lambda x - f(x) = -\lambda \log(-\lambda) + \lambda \quad (6.80)$$

as illustrated in Figure 6.5(a).

To see this, define

$$J(x, \lambda) = \lambda x - f(x) \quad (6.81)$$

We have

$$\frac{\partial J}{\partial x} = \lambda x - f'(x) = \lambda + e^{-x} \quad (6.82)$$

Setting the derivative to zero gives

$$x_\lambda = \arg \max_x J(x, \lambda) = -\log(-\lambda) \quad (6.83)$$

Hence

$$f^\dagger(\lambda) = J(x_\lambda, \lambda) = \lambda(-\log(-\lambda)) - e^{\log(-\lambda)} = -\lambda \log(-\lambda) + \lambda \quad (6.84)$$

1 **6.3.3 Conjugate of a conjugate**

2 It is interesting to see what happens if we take the conjugate of the conjugate:

3

$$\underline{5} \quad f^{**}(\mathbf{x}) = \max_{\boldsymbol{\lambda}} \boldsymbol{\lambda}^T \mathbf{x} - f^\dagger(\boldsymbol{\lambda}) \quad (6.85)$$

6

7 If f is convex, then $f^{**} = f$, so f and f^\dagger are called **conjugate duals**. To see why, note that

8

$$\underline{9} \quad f^{**}(\mathbf{x}) = \max_{\boldsymbol{\lambda}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) \leq f(\mathbf{x}) \quad (6.86)$$

10

11 Since we are free to modify $\boldsymbol{\lambda}$ for each \mathbf{x} , we can make the lower bound tight at each \mathbf{x} . This perfectly
12 characterizes f , since the epigraph of a convex function is an intersection of half-planes defined by
13 linear lower bounds, as shown in Figure 6.5(b).

14 Let us demonstrate this using the example from Section 6.3.2. We have

15

$$\underline{16} \quad f^{**}(x) = \max_{\lambda} \lambda x - f^\dagger(\lambda) = \max_{\lambda} \lambda x + \lambda \log(-\lambda) - \lambda \quad (6.87)$$

17

18 Define

19

$$\underline{20} \quad J^*(x, \lambda) = \lambda x - f^\dagger(x) = \lambda x + \lambda \log(-\lambda) - \lambda \quad (6.88)$$

21 We have

22

$$\underline{23} \quad \frac{\partial}{\partial \lambda} J^*(x, \lambda) = x + \log(-\lambda) + \lambda \left(\frac{-1}{-\lambda} \right) - 1 = 0 \quad (6.89)$$

24

25

$$x = -\log(-\lambda) \quad (6.90)$$

26

$$\lambda_x = -e^{-x} \quad (6.91)$$

27

28 Substituting back we find

29

$$\underline{30} \quad f^{**}(x) = J^*(x, \lambda_x) = (-e^{-x})x + (-e^{-x})(-x) - (-e^{-x}) = e^{-x} = f(x) \quad (6.92)$$

31

32 **6.3.4 Bounds for the logistic (sigmoid) function**

33 In this section, we use the results on conjugate duality to derive upper and lower bounds to the
34 logistic function, $\sigma(x) = \frac{1}{1+e^{-x}}$.

36 **6.3.4.1 Exponential upper bound**

38 The sigmoid function is neither convex nor concave. However, it is easy to show that $f(x) =$
39 $\log \sigma(x) = -\log(1 + e^{-x})$ is concave, by showing that its second derivative is negative. Now, any
40 convex function $f(x)$ can be represented by

42

$$\underline{43} \quad f(x) = \min_{\eta} \eta x - f^\dagger(\eta) \quad (6.93)$$

44 where

45

$$\underline{46} \quad f^\dagger(\eta) = \min_x \eta x - f(x) \quad (6.94)$$

47

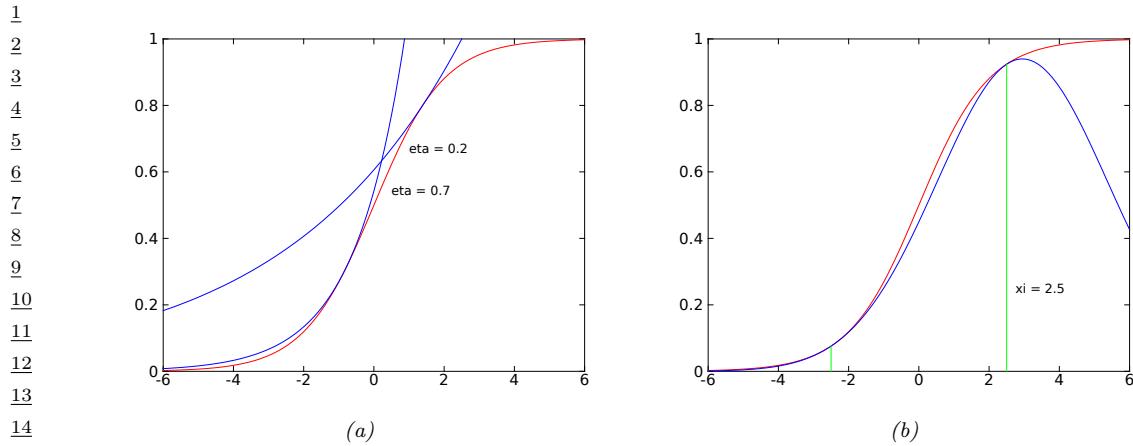


Figure 6.6: Illustration of (a) exponential upper bound and (b) quadratic lower bound to the sigmoid function. Generated by [sigmoid_upper_bounds.ipynb](#) and [sigmoid_lower_bounds.ipynb](#).

One can show that if $f(x) = \log \sigma(x)$, then

$$f^\dagger(\eta) = -\eta \ln \eta - (1-\eta) \ln(1-\eta) \quad (6.95)$$

which is the binary entropy function. Hence

$$\log \sigma(x) \leq \eta x - f^\dagger(\eta) \quad (6.96)$$

$$\sigma(x) \leq \exp(\eta x - f^\dagger(\eta)) \quad (6.97)$$

This exponential upper bound on $\sigma(x)$ is illustrated in Figure 6.6(a).

6.3.4.2 Quadratic lower bound

It is also useful to compute a lower bound on $\sigma(x)$. If we make this a quadratic lower bound, it will “play nicely” with Gaussian priors, which simplifies the analysis of several models. This approach was first suggested in [Jaakkola96b].

First we write

$$\log \sigma(x) = -\log(1 + e^{-x}) = -\log(e^{-x/2}(e^{x/2} + e^{-x/2})) \quad (6.98)$$

$$= x/2 - \log(e^{x/2} + e^{-x/2}) \quad (6.99)$$

The function $f(x) = -\log(e^{x/2} + e^{-x/2})$ is a convex function of $y = x^2$, as can be verified by showing $\frac{d}{dx^2}f(x) > 0$. Hence we can create a linear lower bound on f , using the conjugate function

$$f^\dagger(\eta) = \max_{x^2} \eta x^2 - f(\sqrt{x^2}) \quad (6.100)$$

We have

$$0 = \eta - \frac{dx}{dx^2} \frac{d}{dx} f(x) = \eta + \frac{1}{4x} \tanh(\frac{x}{2}) \quad (6.101)$$

The lower bound is tangent at the point $x_\eta = \xi$, where

$$\eta = -\frac{1}{4\xi} \tanh\left(\frac{\xi}{2}\right) = -\frac{1}{2\xi} \left[\sigma(\xi) - \frac{1}{2} \right] = -\lambda(\xi) \quad (6.102)$$

The conjugate function can be rewritten as

$$f^\dagger(\lambda(\xi)) = -\lambda(\xi)\xi^2 - f(\xi) = \lambda(\xi)\xi^2 + \log(e^{\xi/2} + e^{-\xi/2}) \quad (6.103)$$

So the lower bound on f becomes

$$f(x) \geq -\lambda(\xi)x^2 - g(\lambda(\xi)) = -\lambda(\xi)x^2 + \lambda(\xi)\xi^2 - \log(e^{\xi/2} + e^{-\xi/2}) \quad (6.104)$$

and the lower bound on the sigmoid function becomes

$$\sigma(x) \geq \sigma(\xi) \exp\left[(x - \xi)/2 - \lambda(\xi)(x^2 - \xi^2)\right] \quad (6.105)$$

This is illustrated in Figure 6.6(b).

Although a quadratic is not a good representation for the overall shape of a sigmoid, it turns out that when we use the sigmoid as a likelihood function and combine it with a Gaussian prior, we get a Gaussian-like posterior; in this context, the quadratic lower bound works quite well (since a quadratic likelihood times a Gaussian prior will yield an exact Gaussian posterior). See Section 15.1.1 for an example, where we use this bound for Bayesian logistic regression.

6.4 The Bayesian learning rule

In this section, we discuss the “**Bayesian learning rule**” [BLR], 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

$$\bar{\ell}(\theta) = \sum_{n=1}^N \ell(y_n, f_\theta(x_n)) + R(\theta) \quad (6.106)$$

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 6.7, 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:

$$\mathcal{L}(q) = \mathbb{E}_{q(\theta)} \left[\sum_{n=1}^N \ell(y_n, f_\theta(x_n)) \right] + D_{\text{KL}}(q(\theta) \parallel \pi_0(\theta)) \quad (6.107)$$

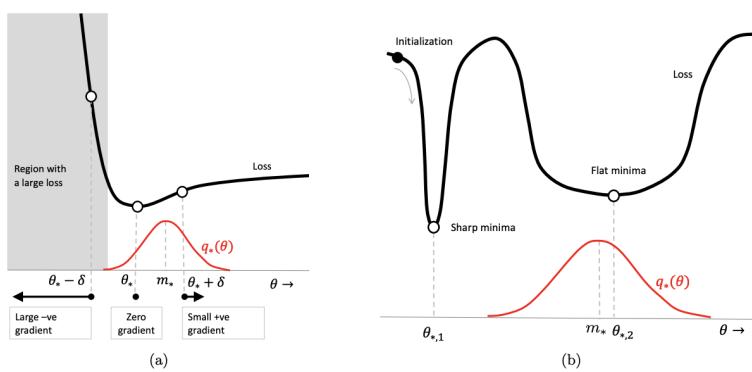


Figure 6.7: Illustration of the robustness obtained by using a Bayesian approach to parameter estimation. (a) When the minimum θ_* lies next to a “wall”, the Bayesian solution shifts away from the boundary to avoid large losses due to perturbations of the parameters. (b) The Bayesian solution prefers flat minima over sharp minima, to avoid large losses due to perturbations of the parameters. From Figure 1 of [BLR]. Used with kind permission of Emtiyaz Khan.

We can rewrite the KL term as

$$D_{\text{KL}}(q(\boldsymbol{\theta}) \parallel \pi_0(\boldsymbol{\theta})) = \mathbb{E}_{q(\boldsymbol{\theta})}[R(\boldsymbol{\theta})] - \mathbb{H}(q(\boldsymbol{\theta})) \quad (6.108)$$

and hence can rewrite the BLR objective as follows:

$$\mathcal{L}(q) = \mathbb{E}_{q(\boldsymbol{\theta})}[\bar{\ell}(\boldsymbol{\theta})] - \mathbb{H}(q(\boldsymbol{\theta})) \quad (6.109)$$

Below we show that different approximations to this objective recover a variety of different methods in the literature.

6.4.1 Deriving inference algorithms from BLR

In this section we show how to derive several different inference algorithms from BLR. (We discuss such algorithms in more detail in ??.)

6.4.1.1 Bayesian inference as optimization

The BLR objective includes standard exact Bayesian inference as a special case, as first shown in [Zellner1988]. To see this, let us assume the loss function is derived from a log-likelihood:

$$\ell(y, f_{\boldsymbol{\theta}}(\mathbf{x})) = -\log p(\mathbf{y}|f_{\boldsymbol{\theta}}(\mathbf{x})) \quad (6.110)$$

Let $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n) : n = 1 : N\}$ be the data we condition on. The Bayesian posterior can be written as

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{1}{Z(\mathcal{D})} \pi_0(\boldsymbol{\theta}) \prod_{n=1}^N p(\mathbf{y}_n|f_{\boldsymbol{\theta}}(\mathbf{x}_n)) \quad (6.111)$$

This can be derived by minimizing the BLR, since

$$\mathcal{L}(q) = -\mathbb{E}_{q(\boldsymbol{\theta})} \left[\sum_{n=1}^N \log p(\mathbf{y}_n | f_{\boldsymbol{\theta}}(\mathbf{x}_n)) \right] + D_{\text{KL}}(q(\boldsymbol{\theta}) \| \pi_0(\boldsymbol{\theta})) \quad (6.112)$$

$$= \mathbb{E}_{q(\boldsymbol{\theta})} \left[\log \frac{q(\boldsymbol{\theta})}{\frac{\pi_0(\boldsymbol{\theta})}{Z(\mathcal{D})} \prod_{n=1}^N p(\mathbf{y}_n | f_{\boldsymbol{\theta}}(\mathbf{x}_n))} \right] - \log Z(\mathcal{D}) \quad (6.113)$$

$$= D_{\text{KL}}(q(\boldsymbol{\theta}) \| p(\boldsymbol{\theta} | \mathcal{D})) - \log Z(\mathcal{D}) \quad (6.114)$$

Since $Z(\mathcal{D})$ is a constant, we can minimize the loss by setting $q(\boldsymbol{\theta}) = p(\boldsymbol{\theta} | \mathcal{D})$.

Of course, we can use other kinds of loss, not just log likelihoods. This results in a framework known as **generalized Bayesian inference** [Bissiri2016; Knoblauch2019; Knoblauch2021]. See ?? for more discussion.

6.4.1.2 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:

$$q(\boldsymbol{\theta}) = h(\boldsymbol{\theta}) \exp[\boldsymbol{\lambda}^T \mathcal{T}(\boldsymbol{\theta}) - A(\boldsymbol{\lambda})] \quad (6.115)$$

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

$$\mathcal{L}(\boldsymbol{\lambda}) = \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] - \mathbb{H}(q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})) \quad (6.116)$$

We can optimize this using natural gradient descent (??). The update becomes

$$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t - \eta_t \tilde{\nabla}_{\boldsymbol{\lambda}} \left[\mathbb{E}_{q_{\boldsymbol{\lambda}_t}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] - \mathbb{H}(q_{\boldsymbol{\lambda}_t}) \right] \quad (6.117)$$

where $\tilde{\nabla}_{\boldsymbol{\lambda}}$ denotes the natural gradient. We discuss how to compute these natural gradients in ???. In particular, we can convert it to regular gradients wrt the moment parameters $\boldsymbol{\mu}_t = \boldsymbol{\mu}(\boldsymbol{\lambda}_t)$. This gives

$$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t - \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}_t}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] + \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{H}(q_{\boldsymbol{\mu}_t}) \quad (6.118)$$

From ?? we have

$$\nabla_{\boldsymbol{\mu}} \mathbb{H}(q) = -\boldsymbol{\lambda} - \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\log h(\boldsymbol{\theta})] \quad (6.119)$$

Hence the update becomes

$$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t - \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}_t}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] - \eta_t \boldsymbol{\lambda}_t - \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\log h(\boldsymbol{\theta})] \quad (6.120)$$

$$= (1 - \eta_t) \boldsymbol{\lambda}_t - \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta}) + \log h(\boldsymbol{\theta})] \quad (6.121)$$

For distributions q with constant base measure $h(\boldsymbol{\theta})$, this simplifies to

$$\boldsymbol{\lambda}_{t+1} = (1 - \eta_t) \boldsymbol{\lambda}_t - \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] \quad (6.122)$$

Hence at the fixed point we have

$$\boldsymbol{\lambda}_* = (1 - \eta) \boldsymbol{\lambda}_* - \eta \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] \quad (6.123)$$

$$\boldsymbol{\lambda}_* = \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [-\bar{\ell}(\boldsymbol{\theta})] = \tilde{\nabla}_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})} [-\bar{\ell}(\boldsymbol{\theta})] \quad (6.124)$$

6.4.1.3 Conjugate variational inference

In ?? we show how to do exact inference in conjugate models. We can derive ?? from the BLR by using the fixed point condition in Equation (6.124) to write

$$\boldsymbol{\lambda}_* = \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_*} [-\bar{\ell}(\boldsymbol{\theta})] = \boldsymbol{\lambda}_0 + \sum_{i=1}^N \underbrace{\nabla_{\boldsymbol{\mu}} \mathbb{E}_{q_*} [\log p(\mathbf{y}_i | \boldsymbol{\theta})]}_{\tilde{\boldsymbol{\lambda}}_i(\mathbf{y}_i)} \quad (6.125)$$

where $\tilde{\boldsymbol{\lambda}}_i(\mathbf{y}_i)$ are the sufficient statistics for the i 'th likelihood term.

For models where the joint distribution over the latents factorizes (using a graphical model), we can further decompose this update into a series of local terms. This gives rise to the variational message passing scheme discussed in ??.

6.4.1.4 Partially conjugate variational inference

In Supplementary ??, we discuss CVI, which performs variational inference for partially conjugate models, using gradient updates for the non-conjugate parts, and exact Bayesian inference for the conjugate parts.

6.4.2 Deriving optimization algorithms from BLR

In this section we show how to derive several different optimization algorithms from BLR. Recall that in BLR, instead of directly minimizing the loss

$$\bar{\ell}(\boldsymbol{\theta}) = \sum_{n=1}^N \ell(\mathbf{y}_n, f_{\boldsymbol{\theta}}(\mathbf{x}_n)) + R(\boldsymbol{\theta}) \quad (6.126)$$

we will instead minimize

$$\mathcal{L}(\boldsymbol{\lambda}) = \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\lambda})} [\bar{\ell}(\boldsymbol{\theta})] - \mathbb{H}(q(\boldsymbol{\theta}|\boldsymbol{\lambda})) \quad (6.127)$$

Below we show that different approximations to this objective recover a variety of different optimization methods that are used in the literature.

1

6.4.2.1 Gradient descent

2
3 In this section, we show how to derive gradient descent as a special case of BLR. We use as our
4 approximate posterior $q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta} | \mathbf{m}, \mathbf{I})$. In this case the natural and moment parameters are equal,
5 $\boldsymbol{\mu} = \boldsymbol{\lambda} = \mathbf{m}$. The base measure satisfies the following (from ??):
6

7 $2 \log h(\boldsymbol{\theta}) = -D \log(2\pi) - \boldsymbol{\theta}^\top \boldsymbol{\theta}$ (6.128)
8

9 Hence

10 $\nabla_{\boldsymbol{\mu}} \mathbb{E}_q [\log h(\boldsymbol{\theta})] = \nabla_{\boldsymbol{\mu}} (-D \log(2\pi) - \boldsymbol{\mu}^\top \boldsymbol{\mu} - D) = -\boldsymbol{\mu} = -\boldsymbol{\lambda} = -\mathbf{m}$ (6.129)

12 Thus from Equation (6.121) the BLR update becomes

14 $\mathbf{m}_{t+1} = (1 - \eta_t) \mathbf{m}_t + \eta_t \mathbf{m}_t - \eta_t \nabla_{\mathbf{m}} \mathbb{E}_{q_{\mathbf{m}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})]$ (6.130)

16 We can remove the expectation using the **first order delta method** [verHoef2012] to get

17 $\nabla_{\mathbf{m}} \mathbb{E}_{q_{\mathbf{m}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] \approx \nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}}$ (6.131)

19 Putting these together gives the gradient descent update:

21 $\mathbf{m}_{t+1} = \mathbf{m}_t - \eta_t \nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}_t}$ (6.132)

23

6.4.2.2 Newton's method

24 In this section, we show how to derive Newton's second order optimization method as a special case
25 of BLR, as first shown in [Khan2018icml].

26 Suppose we assume $q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta} | \mathbf{m}, \mathbf{S}^{-1})$. The natural parameters are

28 $\boldsymbol{\lambda}^{(1)} = \mathbf{S}\mathbf{m}, \boldsymbol{\lambda}^{(2)} = -\frac{1}{2}\mathbf{S}$ (6.133)

30 The mean (moment) parameters are

32 $\boldsymbol{\mu}^{(1)} = \mathbf{m}, \boldsymbol{\mu}^{(2)} = \mathbf{S}^{-1} + \mathbf{m}\mathbf{m}^\top$ (6.134)

34 Since the base measure is constant (see ??), from Equation (6.122) we have

36 $\mathbf{S}_{t+1} \mathbf{m}_{t+1} = (1 - \eta_t) \mathbf{S}_t \mathbf{m}_t - \eta_t \nabla_{\boldsymbol{\mu}^{(1)}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})]$ (6.135)

37 $\mathbf{S}_{t+1} = (1 - \eta_t) \mathbf{S}_t + 2\eta_t \nabla_{\boldsymbol{\mu}^{(2)}} \mathbb{E}_{q_{\boldsymbol{\mu}}(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})]$ (6.136)

39 In ?? we show that

40 $\nabla_{\boldsymbol{\mu}^{(1)}} \mathbb{E}_{q(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] = \mathbb{E}_{q(\boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta})] - \mathbb{E}_{q(\boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta})] \mathbf{m}$ (6.137)

42 $\nabla_{\boldsymbol{\mu}^{(2)}} \mathbb{E}_{q(\boldsymbol{\theta})} [\bar{\ell}(\boldsymbol{\theta})] = \frac{1}{2} \mathbb{E}_{q(\boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta})]$ (6.138)

44 Hence the update for the precision matrix becomes

46 $\mathbf{S}_{t+1} = (1 - \eta_t) \mathbf{S}_t + \eta_t \mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta})]$ (6.139)

For the precision weighted mean, we have

$$\mathbf{S}_{t+1}\mathbf{m}_{t+1} = (1 - \eta_t)\mathbf{S}_t\mathbf{m}_t - \eta_t\mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}}\bar{\ell}(\boldsymbol{\theta})] + \eta_t\mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}}^2\bar{\ell}(\boldsymbol{\theta})]\mathbf{m}_t \quad (6.140)$$

$$= \mathbf{S}_{t+1}\mathbf{m}_t - \eta_t\mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}}\bar{\ell}(\boldsymbol{\theta})] \quad (6.141)$$

Hence

$$\mathbf{m}_{t+1} = \mathbf{m}_t - \eta_t\mathbf{S}_{t+1}^{-1}\mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}}\bar{\ell}(\boldsymbol{\theta})] \quad (6.142)$$

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

$$\mathbf{S}_{t+1} = \mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}}^2\bar{\ell}(\boldsymbol{\theta})] \approx \nabla_{\boldsymbol{\theta}}^2\bar{\ell}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}_t} \quad (6.143)$$

and

$$\mathbb{E}_q [\nabla_{\boldsymbol{\theta}}\bar{\ell}(\boldsymbol{\theta})] \approx \nabla_{\boldsymbol{\theta}}\bar{\ell}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\mathbf{m}_t} \quad (6.144)$$

This gives Newton's update:

$$\mathbf{m}_{t+1} = \mathbf{m}_t - [\nabla_{\mathbf{m}}^2\bar{\ell}(\mathbf{m}_t)]^{-1}[\nabla_{\mathbf{m}}\bar{\ell}(\mathbf{m}_t)] \quad (6.145)$$

6.4.2.3 Variational online Gauss-Newton

In this section, we describe various second order optimization methods that can be derived from the BLR using a series of simplifications.

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 6.4.2.2, we get the following updates:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \frac{1}{\mathbf{s}_{t+1}} \odot \mathbb{E}_{q_t} [\nabla_{\boldsymbol{\theta}}\bar{\ell}(\boldsymbol{\theta})] \quad (6.146)$$

$$\mathbf{s}_{t+1} = (1 - \eta_t)\mathbf{s}_t + \eta_t\mathbb{E}_{q_t} [\text{diag}(\nabla_{\boldsymbol{\theta}}^2\bar{\ell}(\boldsymbol{\theta}))] \quad (6.147)$$

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:

$$\hat{\nabla}_{\boldsymbol{\theta}}\bar{\ell}(\boldsymbol{\theta}) = \frac{N}{M} \sum_{i \in \mathcal{M}} \nabla_{\boldsymbol{\theta}}\ell(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i)) + \nabla_{\boldsymbol{\theta}}R(\boldsymbol{\theta}) \quad (6.148)$$

$$\hat{\nabla}_{\boldsymbol{\theta}}^2\bar{\ell}(\boldsymbol{\theta}) = \frac{N}{M} \sum_{i \in \mathcal{M}} \nabla_{\boldsymbol{\theta}_j}^2\ell(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i)) + \nabla_{\boldsymbol{\theta}_j}^2R(\boldsymbol{\theta}) \quad (6.149)$$

where M is the minibatch size.

For some non-convex problems, such as DNNs, the Hessian may be not be positive definite, so we can get better results using a Gauss-Newton approximation, based on the squared gradients instead of the Hessian:

$$\hat{\nabla}_{\boldsymbol{\theta}_j}^2\bar{\ell}(\boldsymbol{\theta}) \approx \frac{N}{M} \sum_{i \in \mathcal{M}} [\nabla_{\boldsymbol{\theta}_j}\ell(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i))]^2 + \nabla_{\boldsymbol{\theta}_j}^2R(\boldsymbol{\theta}) \quad (6.150)$$

This is also faster to compute.

Putting all this together gives rise to the **online Gauss-Newton** or **OGN** method of [Osawa2019nips].

If we drop the delta approximation, and work with expectations, we get the **variational pline Gauss-Newton** or **VOGN** method of [Khan2018icml]. We can approximate the expectations by sampling. In particular, VOGN uses the following **weight perturbation** method

$$\mathbb{E}_{q_t} \left[\hat{\nabla}_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta}) \right] \approx \hat{\nabla}_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta}_t + \boldsymbol{\epsilon}_t) \quad (6.151)$$

where $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \text{diag}(\mathbf{s}_t))$. It is also possible to approximate the Fisher information matrix directly; this results in the **variational online generalized Gauss-Newton** or **VOGNN** method of [Osawa2019nips].

6.4.2.4 Adaptive learning rate SGD

In this section, we show how to derive an update rule which is very similar to the **RMSprop** [**RMSprop**] method, which is widely used in deep learning. The approach we take is similar to that VOGN in Section 6.4.2.3. We use the same diagonal Gaussian approximation, $q_t(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta} | \boldsymbol{\theta}_t, \mathbf{S}_t^{-1})$, where $\mathbf{S}_t = \text{diag}(\mathbf{s}_t)$ is a vector of precisions. We then use the delta method to eliminate expectations:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \frac{1}{\mathbf{s}_{t+1}} \odot \nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta}_t) \quad (6.152)$$

$$\mathbf{s}_{t+1} = (1 - \eta_t) \mathbf{s}_t + \eta_t \text{diag}(\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta}_t)) \quad (6.153)$$

where \odot is elementwise multiplication. If we allow for different learning rates we get

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha_t \frac{1}{\mathbf{s}_{t+1}} \odot \nabla_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta}_t) \quad (6.154)$$

$$\mathbf{s}_{t+1} = (1 - \beta_t) \mathbf{s}_t + \beta_t \text{diag}(\hat{\nabla}_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta}_t)) \quad (6.155)$$

Now suppose we replace the diagonal Hessian approximation with the sum of the squares per-sample gradients:

$$\text{diag}(\nabla_{\boldsymbol{\theta}}^2 \bar{\ell}(\boldsymbol{\theta}_t)) \approx \hat{\nabla} \bar{\ell}(\boldsymbol{\theta}_t) \odot \hat{\nabla} \bar{\ell}(\boldsymbol{\theta}_t) \quad (6.156)$$

If we also change some scaling factors we can get the RMSprop updates:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \frac{1}{\sqrt{\mathbf{v}_{t+1}} + c\mathbf{1}} \odot \hat{\nabla}_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta}_t) \quad (6.157)$$

$$\mathbf{v}_{t+1} = (1 - \beta) \mathbf{v}_t + \beta [\hat{\nabla} \bar{\ell}(\boldsymbol{\theta}_t) \odot \hat{\nabla} \bar{\ell}(\boldsymbol{\theta}_t)] \quad (6.158)$$

This allows us to use standard deep learning optimizers to get a Gaussian approximation to the posterior for the parameters [Osawa2019nips].

It is also possible to derive the Adam optimizer [adam] from BLR by adding a momentum term to RMSprop. See [BLR; Aitchison2018] for details.

6.4.3 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 distribution, and such expected loss objectives are usually differentiable and smooth. This is called “**variational optimization**” [BarberVarOpt], since we are optimizing over a probability distribution.

For example, consider the case of a **binary neural network** where $\theta_d \in \{0, 1\}$ indicates if weight d is used or not. We can optimize over the parameters of a Bernoulli distribution, $q(\boldsymbol{\theta}|\boldsymbol{\lambda}) = \prod_{d=1}^D \text{Ber}(\theta_d|p_d)$, where $p_d \in [0, 1]$ and $\lambda_d = \log(p_d/(1 - p_d))$ is the log odds. This is the basis of the **BayesBiNN** approach [Meng2020icml].

If we ignore the entropy and regularizer term, we get the following simplified objective:

$$\mathcal{L}(\boldsymbol{\lambda}) = \int \bar{\ell}(\boldsymbol{\theta}) q(\boldsymbol{\theta}|\boldsymbol{\lambda}) d\boldsymbol{\theta} \quad (6.159)$$

This method has various names: **stochastic relaxation** [Staines2012; Staines2013; Malago2013], **stochastic approximation** [Hu2012ac; Hu2012survey], etc. It is closely related to **evolutionary strategies**, which we discuss in ??.

In the case of functions with continuous domains, we can use a Gaussian for $q(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The resulting integral in Equation (6.159) can then sometimes be solved in closed form, as explained in [Mobahi2016]. By starting with a broad variance, and gradually reducing it, we hope the method can avoid poor local optima, similar to simulated annealing (??). However, we generally get better results by including the entropy term, because then we can automatically learn to adapt the variance. In addition, we can often work with natural gradients, which results in faster convergence.

PART II

Inference

7

Inference algorithms: an overview

8 Inference for state-space models

8.1 More Kalman filtering

In this section, we discuss various variants and extensions of Kalman filtering.

8.1.1 Example: tracking an object with spiral dynamics

Consider a variant of the 2d tracking problem in Main Section 29.7.1, where the hidden state is the position and velocity of the object, $\mathbf{z}_t = (u_t \ v_t \ \dot{u}_t \ \dot{v}_t)$. (We use u and v for the two coordinates, to avoid confusion with the state and observation variables.) We use the following dynamics matrix:

$$\mathbf{F} = \begin{pmatrix} 0.1 & 1.1 & \Delta & 0 \\ -1 & 1 & 0 & \Delta \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{pmatrix} \quad (8.1)$$

The eigenvectors of the top left block of this transition matrix are complex, resulting in cyclical behavior, as explained in [Strogatz2015]. Furthermore, since the velocities are shrinking at each step by a factor of 0.1, the cycling behavior becomes a spiral inwards, as illustrated by the line in Figure 8.1(a). The crosses correspond to noisy measurements of the location, as before. In Figure 8.1(b-c), we show the results of Kalman filtering and smoothing.

8.1.2 Derivation of RLS

In this section, we explicitly derive the recursive least squares equations.

Recall from Main Section 8.2.2 that the Kalman filter equations are as follows:

$$\Sigma_{t|t-1} = \mathbf{F}_t \Sigma_{t-1} \mathbf{F}_t^\top + \mathbf{Q}_t \quad (8.2)$$

$$\mathbf{S}_t = \mathbf{H}_t \Sigma_{t|t-1} \mathbf{H}_t^\top + \mathbf{R}_t \quad (8.3)$$

$$\mathbf{K}_t = \Sigma_{t|t-1} \mathbf{H}_t^\top \mathbf{S}_t^{-1} \quad (8.4)$$

$$\boldsymbol{\mu}_t = \mathbf{F}_{t-1} \boldsymbol{\mu}_{t-1} + \mathbf{K}_t (y_t - \mathbf{H}_t \mathbf{F}_{t-1} \boldsymbol{\mu}_{t-1}) \quad (8.5)$$

$$\Sigma_t = (\mathbf{I} - \mathbf{K}_t \mathbf{H}_t) \Sigma_{t|t-1} \quad (8.6)$$

In the case of RLS we have $\mathbf{H}_t = \mathbf{u}_t^\top$, $\mathbf{F}_t = \mathbf{I}$, $\mathbf{Q}_t = 0$ and $\mathbf{R}_t = \sigma^2$. Thus $\Sigma_{t|t-1} = \Sigma_{t-1}$, and the

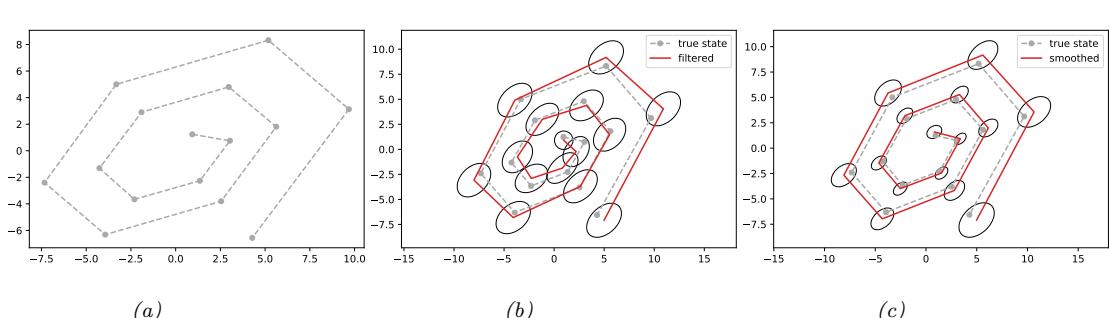


Figure 8.1: Illustration of Kalman filtering and smoothing for a linear dynamical system. (a) Observed data. (b) Filtering. (c) Smoothing. Generated by `kf_spiral.ipynb`.

remaining equations simplify as follows:

$$s_t = \mathbf{u}_t^\top \boldsymbol{\Sigma}_{t-1} \mathbf{u}_t + \sigma^2 \quad (8.7)$$

$$k_t = \frac{1}{s_t} \Sigma_{t-1} u_t \quad (8.8)$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + \mathbf{k}_t(\mathbf{y}_t - \mathbf{u}_t^\top \boldsymbol{\mu}_{t-1}) = \boldsymbol{\mu}_{t-1} + \frac{1}{s_t} \boldsymbol{\Sigma}_{t-1} \mathbf{u}_t (\mathbf{y}_t - \mathbf{u}_t^\top \boldsymbol{\mu}_{t-1}) \quad (8.9)$$

$$\Sigma_t = (\mathbf{I} - k_t \mathbf{u}_t^\top) \Sigma_{t-1} = \Sigma_{t-1} - \frac{1}{s_t} \Sigma_{t-1} \mathbf{u}_t \mathbf{u}_t^\top \Sigma_{t-1} \quad (8.10)$$

Note that from Main Equation (8.32), we can also write the Kalman gain as

$$\mathbf{k}_t = \frac{1}{\sigma^2} (\boldsymbol{\Sigma}_{t-1}^{-1} + \frac{1}{\sigma^2} \mathbf{u}_t \mathbf{u}_t^\top)^{-1} \mathbf{u}_t \quad (8.11)$$

Also, from Main Equation (8.30), we can also write the posterior covariance as

$$\Sigma_{t+1} = \Sigma_{t+1} - s_t k_t k_t^\top \quad (8.12)$$

If we let $\mathbf{V}_t = \Sigma_t / \sigma^2$, we can further simplify the equations, as follows [Borodachev2016].

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + \frac{\sigma^2 \mathbf{V}_{t-1} \boldsymbol{u}_t (\boldsymbol{y}_t - \boldsymbol{u}_t^\top \boldsymbol{\mu}_{t-1})}{\sigma^2 (\boldsymbol{u}_t^\top \mathbf{V}_{t-1} \boldsymbol{u}_t + 1)} = \boldsymbol{\mu}_{t-1} + \frac{\mathbf{V}_{t-1} \boldsymbol{u}_t (\boldsymbol{y}_t - \boldsymbol{u}_t^\top \boldsymbol{\mu}_{t-1})}{\boldsymbol{u}_t^\top \mathbf{V}_{t-1} \boldsymbol{u}_t + 1} \quad (8.13)$$

$$\mathbf{V}_t = \mathbf{V}_{t-1} - \frac{\mathbf{V}_{t-1} u_t u_t^\top \mathbf{V}_{t-1}}{u_t^\top \mathbf{V}_{t-1} u_t + 1} \quad (8.14)$$

We can initialize these recursions using a vague prior, $\mu_0 = \mathbf{0}$, $\Sigma_0 = \infty \mathbf{I}$. 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.

8.1.3 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 [West97]. The model is defined as follows:

$$p(\mathbf{z}_t | \mathbf{z}_{t-1}) = \mathcal{N}(\mathbf{z}_t | \mathbf{F}_t \mathbf{z}_{t-1}, V \mathbf{Q}_t^*) \quad (8.15)$$

$$p(y_t | \mathbf{z}_t) = \mathcal{N}(y_t | \mathbf{h}_t^\top \mathbf{z}_t, V) \quad (8.16)$$

where \mathbf{Q}_t^* is the unscaled system noise, and we define $\mathbf{H}_t = \mathbf{h}_t^\top$ 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:

$$p_0(\lambda) = \text{Ga}\left(\frac{\nu_0}{2}, \frac{\nu_0 \tau_0}{2}\right) \quad (8.17)$$

$$p_0(\mathbf{z} | \lambda) = \mathcal{N}(\boldsymbol{\mu}_0, V \boldsymbol{\Sigma}_0^*) \quad (8.18)$$

where τ_0 is the prior mean for σ^2 , and $\nu_0 > 0$ is the strength of this prior.

We now discuss the belief updating step. We assume that the prior belief state at time $t - 1$ is

$$\mathcal{N}(\mathbf{z}_{t-1}, \lambda | \mathcal{D}_{1:t-1}) = \mathcal{N}(\mathbf{z}_{t-1} | \boldsymbol{\mu}_{t-1}, V \boldsymbol{\Sigma}_{t-1}^*) \text{Ga}(\lambda | \frac{\nu_{t-1}}{2}, \frac{\nu_{t-1} \tau_{t-1}}{2}) \quad (8.19)$$

The posterior is given by

$$\mathcal{N}(\mathbf{z}_t, \lambda | \mathcal{D}_{1:t}) = \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_t, V \boldsymbol{\Sigma}_t^*) \text{Ga}(\lambda | \frac{\nu_t}{2}, \frac{\nu_t \tau_t}{2}) \quad (8.20)$$

where

$$\boldsymbol{\mu}_{t|t-1} = \mathbf{F}_t \boldsymbol{\mu}_{t-1} \quad (8.21)$$

$$\boldsymbol{\Sigma}_{t|t-1}^* = \mathbf{F}_t \boldsymbol{\Sigma}_{t-1}^* \mathbf{F}_t + \mathbf{Q}_t^* \quad (8.22)$$

$$e_t = y_t - \mathbf{h}_t^\top \boldsymbol{\mu}_{t|t-1} \quad (8.23)$$

$$s_t^* = \mathbf{h}_t^\top \boldsymbol{\Sigma}_{t|t-1}^* \mathbf{h}_t + 1 \quad (8.24)$$

$$\mathbf{k}_t = \boldsymbol{\Sigma}_{t|t-1}^* \mathbf{h}_t / s_t^* \quad (8.25)$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + \mathbf{k}_t e_t \quad (8.26)$$

$$\boldsymbol{\Sigma}_t^* = \boldsymbol{\Sigma}_{t|t-1}^* - \mathbf{k}_t \mathbf{k}_t^\top s_t^* \quad (8.27)$$

$$\nu_t = \nu_{t-1} + 1 \quad (8.28)$$

$$\nu_t \tau_t = \nu_{t-1} \tau_{t-1} + e_t^2 / s_t^* \quad (8.29)$$

If we marginalize out V , the marginal distribution for \mathbf{z}_t is a Student distribution:

$$p(\mathbf{z}_t | \mathcal{D}_{1:t}) = \mathcal{T}_{\nu_t}(\mathbf{z}_t | \boldsymbol{\mu}_t, \tau_t \boldsymbol{\Sigma}_t^*) \quad (8.30)$$

The one-step-ahead posterior predictive density for the observations is given by

$$p(y_t | \mathbf{y}_{1:t-1}) = \mathcal{T}_{\nu_{t-1}}(y_t | \hat{y}_t, \tau_{t-1} s_t^*) \quad (8.31)$$

These equations only differs from the standard KF equations by the scaling term τ_t (or τ_{t-1} for the predictive), and the use of a Student distribution instead of a Gaussian. However, as ν_t increases over time, the Student distribution will rapidly converge to a Gaussian.

8.1.4 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., [Rao1999; Friston2003; Spratling2017; Millidge2021pc; Marino2021]). 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. 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 [Millidge2021phd].

To see this, we adopt the framework of inference as optimization, as used in variational inference. 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 Main Section 10.1.1.1, the variational free energy (VFE) is given by $\mathcal{F}(\boldsymbol{\psi}) = \sum_{t=1}^T \mathcal{F}_t(\boldsymbol{\psi}_t)$, where

$$\mathcal{F}_t(\boldsymbol{\psi}_t) = \mathbb{E}_{q(\mathbf{z}_{t-1})} [D_{\text{KL}}(q(\mathbf{z}_t) \parallel p(\mathbf{z}_t, \mathbf{y}_t | \mathbf{z}_{t-1}))] \quad (8.32)$$

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

$$\mathcal{F}_t(\boldsymbol{\mu}_t) = -\mathbb{E}_{q(\mathbf{z}_t | \boldsymbol{\mu}_t)} [\log p(\mathbf{y}_t, \mathbf{z}_t | \boldsymbol{\mu}_{t-1})] - \mathbb{H}(q(\mathbf{z}_t | \boldsymbol{\mu}_t)) \quad (8.33)$$

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:

$$\mathbb{E}[\log p(\mathbf{y}_t, \mathbf{z}_t | \boldsymbol{\mu}_{t-1})] \approx \mathbb{E}[\log p(\mathbf{y}_t, \mathbf{z}_t | \boldsymbol{\mu}_{t-1})] + \mathbb{E}[\nabla_{\mathbf{z}_t} p(\mathbf{y}_t, \mathbf{z}_t | \boldsymbol{\mu}_{t-1})|_{\mathbf{z}_t=\boldsymbol{\mu}_t} (\mathbf{z}_t - \boldsymbol{\mu}_t)] \quad (8.34)$$

$$+ \mathbb{E}[\nabla_{\mathbf{z}_t}^2 p(\mathbf{y}_t, \mathbf{z}_t | \boldsymbol{\mu}_{t-1})|_{\mathbf{z}_t=\boldsymbol{\mu}_t} (\mathbf{z}_t - \boldsymbol{\mu}_t)^2] \quad (8.35)$$

$$= \log p(\mathbf{y}_t, \boldsymbol{\mu}_t | \boldsymbol{\mu}_{t-1}) + \nabla_{\mathbf{z}_t} p(\mathbf{y}_t, \mathbf{z}_t | \boldsymbol{\mu}_{t-1})|_{\mathbf{z}_t=\boldsymbol{\mu}_t} \underbrace{\mathbb{E}[(\mathbf{z}_t - \boldsymbol{\mu}_t)]}_0 \quad (8.36)$$

$$+ \nabla_{\mathbf{z}_t}^2 p(\mathbf{y}_t, \mathbf{z}_t | \boldsymbol{\mu}_{t-1})|_{\mathbf{z}_t=\boldsymbol{\mu}_t} \underbrace{\mathbb{E}[(\mathbf{z}_t - \boldsymbol{\mu}_t)^2]}_{\boldsymbol{\Sigma}} \quad (8.37)$$

We can drop the second and third terms, since they are independent of $\boldsymbol{\mu}_t$. Thus we just need to solve

$$\boldsymbol{\mu}_t^* = \underset{\boldsymbol{\mu}_t}{\operatorname{argmin}} \mathcal{F}_t(\boldsymbol{\mu}_t) \quad (8.38)$$

$$\mathcal{F}_t(\boldsymbol{\mu}_t) = \log p(\mathbf{y}_t, \boldsymbol{\mu}_t | \boldsymbol{\mu}_{t-1}) \quad (8.39)$$

$$= -(\mathbf{y}_t - \mathbf{H}\boldsymbol{\mu}_t)\boldsymbol{\Sigma}_y^{-1}(\mathbf{y}_t - \mathbf{H}\boldsymbol{\mu}_t) \quad (8.40)$$

$$+ (\boldsymbol{\mu}_t - \mathbf{F}\boldsymbol{\mu}_{t-1} - \mathbf{B}\mathbf{u}_{t-1})^\top(\boldsymbol{\mu}_t - \mathbf{F}\boldsymbol{\mu}_{t-1} - \mathbf{B}\mathbf{u}_{t-1})^\top\boldsymbol{\Sigma}_z^{-1} \quad (8.41)$$

We will solve this problem by gradient descent. The form of the gradient turns out to be very simple, and involves two prediction error terms: one from the past state estimate, $\epsilon_z = \mu_t - \mathbf{F}\mu_{t-1} - \mathbf{B}u_{t-1}$, and one from the current observation, $\epsilon_y = y_t - \mathbf{H}\mu_t$:

$$\nabla \mathcal{F}_t(\mu_t) = 2\mathbf{H}^\top \Sigma_y^{-1} y_t - (\mathbf{H}^\top \Sigma_y^{-1} \mathbf{H} + \mathbf{H}^\top \Sigma_y^{-1} \mathbf{H})\mu_t \quad (8.42)$$

$$+ (\Sigma_z^{-1} + \Sigma_z - \mathbf{T})\mu_t - 2\Sigma_z^{-1} \mathbf{F}\mu_{t-1} - 2\Sigma_z^{-1} \mathbf{B}u_{t-1} \quad (8.43)$$

$$= 2\mathbf{H}^\top \Sigma_y^{-1} \mathbf{H}\mu_t - 2\mathbf{H}^\top \Sigma_y^{-1} \mathbf{H}\mu_t + 2\Sigma_z^{-1} \mu_t - 2\Sigma_z^{-1} \mathbf{F}\mu_{t-1} - 2\Sigma_z^{-1} \mathbf{B}u_{t-1} \quad (8.44)$$

$$= -\mathbf{H}^\top \Sigma_y^{-1} [y_t - \mathbf{H}\mu_t] + \Sigma_z^{-1} [\mu_t - \mathbf{F}\mu_{t-1} - \mathbf{B}u_{t-1}] \quad (8.45)$$

$$= -\mathbf{H}^\top \Sigma_y^{-1} \epsilon_y + \Sigma_z^{-1} \epsilon_z \quad (8.46)$$

Thus minimizing (precision weighted) prediction errors is equivalent to minimizing the VFE.¹ 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., [Hosseini2020pc]).

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, [Millidge2021phd] show that for the dynamics matrix the gradient is $\nabla_{\mathbf{F}} \mathcal{F}_t = -\Sigma_z \epsilon_y \mu_{t-1}^\top$, for the control matrix the gradient is $\nabla_{\mathbf{B}} \mathcal{F}_t = -\Sigma_z \epsilon_y u_{t-1}^\top$, and for the observation matrix the gradient is $\nabla_{\mathbf{H}} \mathcal{F}_t = -\Sigma_y \epsilon_y \mu_t^\top$. These expressions can be generalized to nonlinear models. Indeed, predictive coding can in fact approximate backpropagation for many kinds of model [Millidge2020predictive].

Gradient descent using these predicting coding takes the form of a **Hebbian update rule**, in which we set the new parameter to the old one plus a term that is a multiplication of the two quantities available at each end of the synaptic connection, namely the prediction error ϵ as input, and the value μ (or θ) of the neuron as output. However, there are still several aspects of this model that are biologically implausible, such as assuming symmetric weights (since both \mathbf{H} and \mathbf{H}^\top are needed, the former to compute ϵ_y and the latter to compute $\nabla_{\mu_t} \mathcal{F}_t$), 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 [Millidge2021] 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.

8.2 More extended Kalman filtering

8.2.1 Derivation of the EKF

The derivation of the EKF is similar to the derivation of the Kalman filter (Main Section 8.2.2.4), except we also need to apply the linear approximation from Main Section 8.3.1.

1. Scaling the error terms by the inverse variance can be seen as a form of normalization. To see this, consider the standardization operator: $\text{standardize}(x) = (x - \mathbb{E}[x]) / \sqrt{\mathbb{V}[x]}$. It has been argued that the widespread presence of neural circuitry for performing normalization, together with the upwards and downwards connections between brain regions, adds support for the claim that the brain implements predictive coding (see e.g., [Rao1999predictive; Friston2003; Spratling2017; Millidge2021pc; Marino2021]).

First we approximate the joint of \mathbf{z}_{t-1} and $\mathbf{z}_t = \mathbf{f}_t(\mathbf{z}_{t-1}) + \mathbf{q}_{t-1}$ to get

$$p(\mathbf{z}_{t-1}, \mathbf{z}_t | \mathbf{y}_{1:t-1}) \approx \mathcal{N}\left(\begin{pmatrix} \mathbf{z}_{t-1} \\ \mathbf{z}_t \end{pmatrix} | \mathbf{m}', \Sigma'\right) \quad (8.47)$$

$$\mathbf{m}' = \begin{pmatrix} \boldsymbol{\mu}_{t-1} \\ \mathbf{f}(\boldsymbol{\mu}_{t-1}) \end{pmatrix} \quad (8.48)$$

$$\Sigma' = \begin{pmatrix} \Sigma_{t-1} & \Sigma_{t-1} \mathbf{F}_t^\top \\ \mathbf{F}_t \Sigma_{t-1} & \mathbf{F}_t \Sigma_{t-1} \mathbf{F}_t^\top + \mathbf{Q}_{t-1} \end{pmatrix} \quad (8.49)$$

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 $p(\mathbf{z}_t, \mathbf{y}_t | \mathbf{y}_{1:t-1})$, where $\mathbf{y}_t = \mathbf{h}_t(\mathbf{z}_t) + \mathbf{r}_t$:

$$p(\mathbf{z}_t, \mathbf{y}_t | \mathbf{y}_{1:t-1}) \approx \mathcal{N}\left(\begin{pmatrix} \mathbf{z}_t \\ \mathbf{y}_t \end{pmatrix} | \mathbf{m}'', \Sigma''\right) \quad (8.50)$$

$$\mathbf{m}'' = \begin{pmatrix} \boldsymbol{\mu}_{t|t-1} \\ \mathbf{h}(\boldsymbol{\mu}_{t|t-1}) \end{pmatrix} \quad (8.51)$$

$$\Sigma'' = \begin{pmatrix} \Sigma_{t|t-1} & \Sigma_{t|t-1} \mathbf{H}_t^\top \\ \mathbf{H}_t \Sigma_{t|t-1} & \mathbf{H}_t \Sigma_{t|t-1} \mathbf{H}_t^\top + \mathbf{R}_{t-1} \end{pmatrix} \quad (8.52)$$

Finally, we use Main Equation (2.78) to get the posterior

$$p(\mathbf{z}_t | \mathbf{y}_t, \mathbf{y}_{1:t-1}) \approx \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_t, \Sigma_t) \quad (8.53)$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \Sigma_{t|t-1} \mathbf{H}_t^\top (\mathbf{H}_t \Sigma_{t|t-1} \mathbf{H}_t^\top + \mathbf{R}_t)^{-1} [\mathbf{y}_t - \mathbf{h}(\boldsymbol{\mu}_{t|t-1})] \quad (8.54)$$

$$\Sigma_t = \Sigma_{t|t-1} - \Sigma_{t|t-1} \mathbf{H}_t^\top (\mathbf{H}_t \Sigma_{t|t-1} \mathbf{H}_t^\top + \mathbf{R}_t)^{-1} \mathbf{H}_t \Sigma_{t|t-1} \quad (8.55)$$

This gives us the update step.

8.2.2 Example: Tracking a pendulum

Consider a simple pendulum of unit mass and length swinging from a fixed attachment, as in Figure 8.2a. Such an object is in principle entirely deterministic in its behavior. However, in the real world, there are often unknown forces at work (e.g., air turbulence, friction). We will model these by a continuous time random **white noise process** $w(t)$. This gives rise to the following differential equation [Sarkka13]:

$$\frac{d^2\alpha}{dt^2} = -g \sin(\alpha) + w(t) \quad (8.56)$$

We can write this as a nonlinear SSM by defining the state to be $z_1(t) = \alpha(t)$ and $z_2(t) = d\alpha(t)/dt$. Thus

$$\frac{dz}{dt} = \begin{pmatrix} z_2 \\ -g \sin(z_1) \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} w(t) \quad (8.57)$$

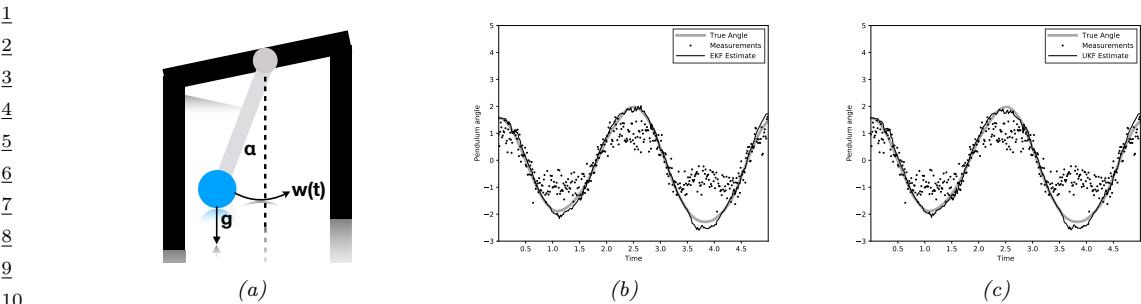


Figure 8.2: (a) Illustration of a pendulum swinging. g is the force of gravity, $w(t)$ is a random external force, and α is the angle wrt the vertical. Adapted from Figure 3.10 in [Sarkka13]. (b) Extended Kalman filter. Generated by [ekf_pendulum.ipynb](#). (b) Unscented Kalman filter. Generated by [ukf_pendulum.ipynb](#).

If we discretize this step size Δ , we get the following formulation [Sarkka13]:

$$\underbrace{\begin{pmatrix} z_{1,t} \\ z_{2,t} \end{pmatrix}}_{\mathbf{z}_t} = \underbrace{\begin{pmatrix} z_{1,t-1} + z_{2,t-1}\Delta \\ z_{2,t-1} - g \sin(z_{1,t-1})\Delta \end{pmatrix}}_{\mathbf{f}(\mathbf{z}_{t-1})} + \mathbf{q}_t \quad (8.58)$$

where $\mathbf{q}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ with

$$\mathbf{Q} = q^c \begin{pmatrix} \frac{\Delta^3}{3} & \frac{\Delta^2}{2} \\ \frac{\Delta^2}{2} & \Delta \end{pmatrix} \quad (8.59)$$

where q^c is the **spectral density** (continuous time variance) of the continuous-time noise process.

If we observe the angular position, we get the linear observation model $\mathbf{h}(\mathbf{z}_t) = \alpha_t = \mathbf{z}_{t,1}$. If we only observe the horizontal position, we get the nonlinear observation model $\mathbf{h}(\mathbf{z}_t) = \sin(\alpha_t) = \sin(\mathbf{z}_{t,1})$.

To apply the EKF to this problem, we need to compute the following Jacobian matrices:

$$\mathbf{F}(\mathbf{z}) = \begin{pmatrix} 1 & \Delta \\ -g \cos(z_1)\Delta & 1 \end{pmatrix}, \quad \mathbf{H}(\mathbf{z}) = (\cos(z_1) \quad 0) \quad (8.60)$$

The results are shown in Figure 8.2b.

8.3 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 [Ollivier2018]. We call this the **Exponential family EKF** or **EKF**. This allows us to apply the EKF for online parameter estimation of classification models, as we illustrate in Section 8.3.3.

8.3.1 Modeling assumptions

We assume the dynamics model is the usual nonlinear model plus Gaussian noise, with optional inputs \mathbf{u}_t :

$$\mathbf{z}_t = \mathbf{f}(\mathbf{z}_{t-1}, \mathbf{u}_t) + \mathcal{N}(\mathbf{0}, \mathbf{Q}_t) \quad (8.61)$$

We assume the observation model is

$$p(\mathbf{y}_t | \mathbf{z}_t) = \text{Expfam}(\mathbf{y}_t | \hat{\mathbf{y}}_t) \quad (8.62)$$

where the mean (moment) parameter of the exponential family is computed deterministically using a nonlinear observation model:

$$\hat{\mathbf{y}}_t = h(\mathbf{z}_t, \mathbf{u}_t) \quad (8.63)$$

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.

8.3.2 Algorithm

The EEKF algorithm is as follows. First, the prediction step:

$$\boldsymbol{\mu}_{t|t-1} = f(\boldsymbol{\mu}_{t-1}, \mathbf{u}_t) \quad (8.64)$$

$$\mathbf{F}_t = \frac{\partial f}{\partial \mathbf{z}}|_{(\boldsymbol{\mu}_{t-1}, \mathbf{u}_t)} \quad (8.65)$$

$$\boldsymbol{\Sigma}_{t|t-1} = \mathbf{F}_t \boldsymbol{\Sigma}_{t-1} \mathbf{F}_t^T + \mathbf{Q}_t \quad (8.66)$$

$$\hat{\mathbf{y}}_t = h(\boldsymbol{\mu}_{t|t-1}, \mathbf{u}_t) \quad (8.67)$$

Second, after seeing observation \mathbf{y}_t , we compute the following:

$$\mathbf{e}_t = \mathcal{T}(\mathbf{y}_t) - \hat{\mathbf{y}}_t \quad (8.68)$$

$$\mathbf{R}_t = \text{Cov}[\mathcal{T}(\mathbf{y})|\hat{\mathbf{y}}_t] \quad (8.69)$$

where $\mathcal{T}(\mathbf{y})$ is the vector of sufficient statistics, and \mathbf{e}_t is the error or innovation term. (For a Gaussian observation model with fixed noise, we have $\mathcal{T}(\mathbf{y}) = \mathbf{y}$, so $\mathbf{e}_t = \mathbf{y}_t - \hat{\mathbf{y}}_t$, as usual.)

Finally we perform the update:

$$\mathbf{H}_t = \frac{\partial h}{\partial \mathbf{z}}|_{(\boldsymbol{\mu}_{t|t-1}, \mathbf{u}_t)} \quad (8.70)$$

$$\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^T (\mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^T + \mathbf{R}_t)^{-1} \quad (8.71)$$

$$\boldsymbol{\Sigma}_t = (\mathbf{I} - \mathbf{K}_t \mathbf{H}_t) \boldsymbol{\Sigma}_{t|t-1} \quad (8.72)$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t \mathbf{e}_t \quad (8.73)$$

In [Ollivier2018], they show that this is equivalent to an online version of natural gradient descent (Main Section 6.4).

8.3.3 EEKF for training logistic regression

For example, consider the case where y is a class label with C possible values. (We drop the time index for brevity.) Following Main Section 2.4.2.2, Let

$$\mathcal{T}(\mathbf{y}) = [\mathbb{I}(y=1), \dots, \mathbb{I}(y=C-1)] \quad (8.74)$$

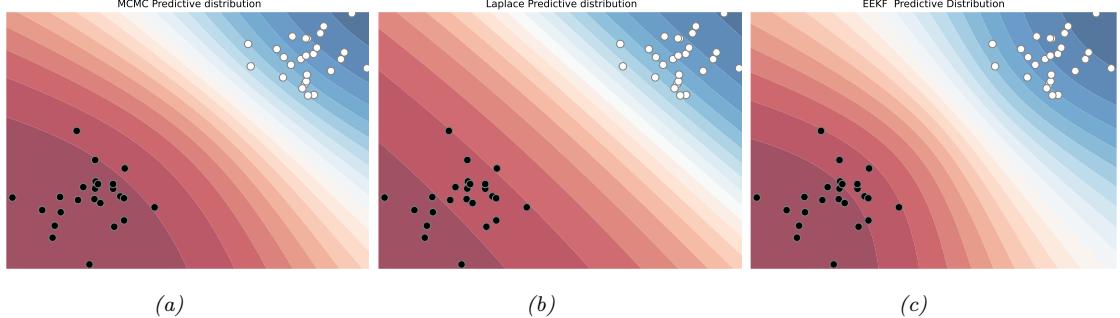


Figure 8.3: Bayesian inference applied to a 2d binary logistic regression problem, $p(y = 1|\mathbf{x}) = \sigma(w_0 + w_1x_1 + w_2x_2)$. We show the training data and the posterior predictive produced by different methods. (a) Offline MCMC approximation. (b) Offline Laplace approximation. (c) Online EEFK approximation at the final step of inference. Generated by [eekf_logistic_regression.ipynb](#).

be the $(C - 1)$ -dimensional vector of sufficient statistics, and let $\hat{\mathbf{y}} = [p_1, \dots, p_{C-1}]$ be the corresponding predicted probabilities of each class label. The probability of the C 'th class is given by $p_C = 1 - \sum_{c=1}^{C-1} \hat{y}_c$; we avoid including this to ensure that \mathbf{R} is not singular. The $(C - 1) \times (C - 1)$ covariance matrix \mathbf{R} is given by

$$R_{ij} = \text{diag}(p_i) - p_i p_j \quad (8.75)$$

Now consider the simpler case where we have two class labels, so $C = 2$. In this case, $\mathcal{T}(\mathbf{y}) = \mathbb{I}(\mathbf{y} = 1)$, and $\hat{\mathbf{y}} = p(\mathbf{y} = 1) = p$. The covariance matrix of the observation noise becomes the scalar $r = p(1 - p)$. Of course, we can make the output probabilities depend on the input covariates, as follows:

$$p(y_t|\mathbf{z}_t, \mathbf{u}_t) = \text{Ber}(y_t|\sigma(\mathbf{z}_t^\top \mathbf{u}_t)) \quad (8.76)$$

We assume the parameters \mathbf{z}_t are static, so $\mathbf{Q}_t = \mathbf{0}$. The 2d data is shown in Figure 8.3a. We sequentially compute the posterior using the EEFK, and compare to the offline estimate computed using a Laplace approximation (where the MAP estimate is computed using BFGS) and an MCMC approximation, which we take as “ground truth”. In Figure 8.3c, we see that the resulting posterior predictive distributions are similar. Finally, in Figure 8.4, we visualize how the posterior marginals converge over time. (See also Main Section 8.6.3, where we solve this same problem using ADF.)

8.3.4 EEFK performs online natural gradient descent

In this section, we show an exact equivalence, due to [Ollivier2018], between the EKF for exponential family likelihoods (Section 8.3) and an online version of natural gradient descent (Main Section 6.4).

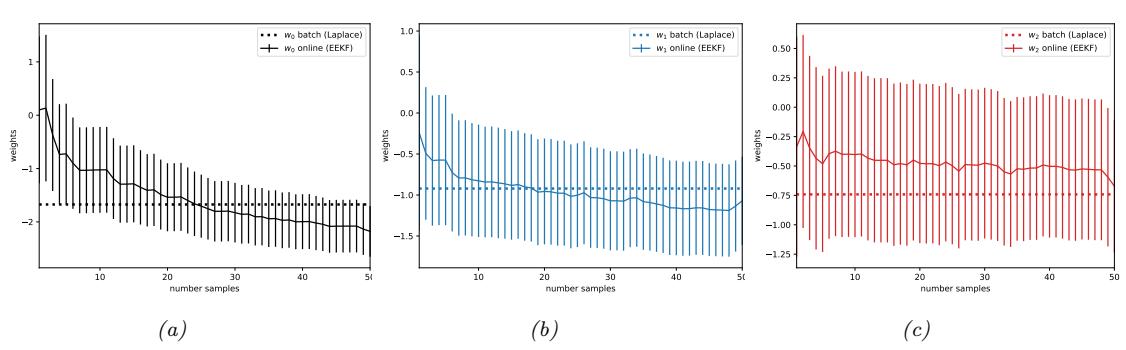


Figure 8.4: Marginal posteriors over time for the EEEKF method. The horizontal line is the offline MAP estimate. Generated by `eekf_logistic_regression.ipynb`.

8.3.4.1 Statement of the equivalence

We define online natural gradient descent as follows. Given a set of labeled pairs, $(\mathbf{u}_t, \mathbf{y}_t)$, the goal is to minimize the loss function

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_t \mathcal{L}_t(\mathbf{y}_t) \quad (8.77)$$

where

$$\mathcal{L}_t(\mathbf{y}_t) = -\log p(\mathbf{y}|\hat{\mathbf{y}}_t) \quad (8.78)$$

and

$$\hat{\mathbf{y}}_t = h(\boldsymbol{\theta}, \mathbf{u}_t) \quad (8.79)$$

is the prediction from some model, such as a neural network.

At each step of the algorithm, we perform the following updates, where \mathbf{J}_t is the Fisher information matrix (Main Section 3.3.4), and $\boldsymbol{\theta}_t$ is the current parameter estimate:

$$\mathbf{J}_t = (1 - \gamma_t) \mathbf{J}_{t-1} + \gamma_t \mathbb{E}_{p(\mathbf{y}|\hat{\mathbf{y}})} [\nabla_{\boldsymbol{\theta}} \mathcal{L}_t(\mathbf{y}) \nabla_{\boldsymbol{\theta}} \mathcal{L}_t(\mathbf{y})^T] \quad (8.80)$$

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta_t \mathbf{J}_t^{-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}_t(\mathbf{y}) \quad (8.81)$$

where η_t is the learning rate and γ_t is the Fisher matrix decay rate.

Theorem 1 (EKF performs natural gradient descent). *The above NGD algorithm is identical to performing EEEKF, with $\boldsymbol{\theta}_t = \mathbf{z}_t$, under the following conditions: static dynamics with $\mathbf{z}_{t+1} = f(\mathbf{z}_t, \mathbf{u}_t) = \mathbf{z}_t$, exponential family observations with mean parameters $\hat{\mathbf{y}}_t = h(\mathbf{z}_t, \mathbf{u}_t)$, learning rate $\eta_t = \gamma_t = 1/(t+1)$, and Fisher matrix set to $\mathbf{J}_t = \Sigma_t^{-1}/(t+1)$.*

See Section 8.3.4.3 for the proof of this claim.

1
2 **8.3.4.2 Fading memory**

3 For many problems, a learning rate schedule of the form $\eta_t = 1/(t+1)$ results in overly small updates
4 later in the sequence, resulting in very slow progress. It is therefore useful to be able to setting the
5 learning rates to larger values, such as a contant $\eta_t = \eta_0$. (We continue to assume that $\gamma_t = \eta_t$, for
6 the equivalence to hold.)
7

8 We can emulate this generic learning rate with an EKF by using the **fading memory** trick
9 [Haykin01], in which we update

10
$$\Sigma_{t-1} = \Sigma_{t-1}/(1 - \lambda_t) \quad (8.82)$$

11

12 before the prediction step, where

13

$$1 - \lambda_t = \frac{\eta_{t-1}}{\eta_t} - \eta_{t-1} \quad (8.83)$$

14 If we set $\eta_t = \eta_0$, then we find $\lambda_t = \eta_0$; if we set $\eta_t = 1/(t + \text{const})$, then we find $\lambda_t = 0$.
15

16 Fading memory is equivalent to adding artificial process noise \mathbf{Q}_t with a value proportional to
17 Σ_{t-1} . This has the effect of putting less weight on older observations. This is equivalent to NGD
18 using a Fisher matrix of the form $\mathbf{K}_t = \eta_t \Sigma_t^{-1}$.
19

20 The learning rate also controls the weight given to the prior. If we use the fading memory trick,
21 the effect of the prior in the initial time step decreases exponentially with time, which can result
22 in overfitting (especially since we are downweighting past observations). We may therefore want to
23 artificially increase the initial uncertainty, Σ_0 . This can be emulated in NGD by regularizing the
24 Fisher matrix, see Proposition 4 of [Ollivier2018] for details.
25

26
27 **8.3.4.3 Proof of the claim**

28 In this section, we prove the equivalence between EKF and NGD. We start by proving this lemma.

29

30 **Lemma 1.** *The error term of the EEKF is given by*

31

$$\mathbf{e}_t \triangleq \mathcal{T}(\mathbf{y}_t) - \hat{\mathbf{y}}_t = \mathbf{R}_t \nabla_{\hat{\mathbf{y}}_t} \log p(\mathbf{y}_t | \hat{\mathbf{y}}_t) \quad (8.84)$$

32 *Proof.* For the case of Gaussian observations, this is easy to see, since $\mathcal{T}(\mathbf{y}_t) = \mathbf{y}_t$ and

33

$$\log p(\mathbf{y}_t | \hat{\mathbf{y}}_t) = -\frac{1}{2} (\mathbf{y}_t - \hat{\mathbf{y}}_t)^T \mathbf{R}_t^{-1} (\mathbf{y}_t - \hat{\mathbf{y}}_t) \quad (8.85)$$

34 so

35

$$\nabla_{\hat{\mathbf{y}}_t} \log p(\mathbf{y}_t | \hat{\mathbf{y}}_t) = \mathbf{R}_t^{-1} (\mathcal{T}(\mathbf{y}_t) - \hat{\mathbf{y}}_t) \quad (8.86)$$

36 Now consider the general exponential family with natural parameters $\boldsymbol{\eta}$ and moment parameters
37 $\hat{\mathbf{y}}$. From the chain rule and Main Equation (2.247), We have

38

$$\frac{\partial \log p(\mathbf{y} | \boldsymbol{\eta})}{\partial \boldsymbol{\eta}} = \frac{\partial \hat{\mathbf{y}}}{\partial \boldsymbol{\eta}} \frac{\partial \log p(\mathbf{y} | \hat{\mathbf{y}})}{\partial \hat{\mathbf{y}}} = \mathcal{T}(\mathbf{y}) - \mathbb{E}[\mathcal{T}(\mathbf{y})] \quad (8.87)$$

From Main Equation (3.78) and Main Equation (3.72) we have

$$\frac{\partial \hat{\mathbf{y}}}{\partial \boldsymbol{\eta}} = \mathbf{F}_{\boldsymbol{\eta}} = \text{Cov} [\mathcal{T}(\mathbf{y})] = \mathbf{R} \quad (8.88)$$

Hence

$$\frac{\partial \log p(\mathbf{y}|\boldsymbol{\eta})}{\partial \boldsymbol{\eta}} = \mathbf{R} \frac{\partial \log p(\mathbf{y}|\hat{\mathbf{y}})}{\partial \hat{\mathbf{y}}} = \mathcal{T}(\mathbf{y}) - \mathbb{E}[\mathcal{T}(\mathbf{y})] \quad (8.89)$$

from which we get Equation (8.84). \square

Now we prove another lemma.

Lemma 2. *The Kalman gain matrix of the EKF satsifies*

$$\mathbf{K}_t \mathbf{R}_t = \boldsymbol{\Sigma}_t \mathbf{H}_t^T \quad (8.90)$$

Proof. Using the definition of \mathbf{K}_t we have

$$\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^T (\mathbf{R}_t + \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^T)^{-1} \quad (8.91)$$

$$\mathbf{K}_t \mathbf{R}_t = \mathbf{K}_t (\mathbf{R}_t + \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^T) - \mathbf{K}_t \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^T \quad (8.92)$$

$$= \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^T - \mathbf{K}_t \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^T \quad (8.93)$$

$$= (\mathbf{I} - \mathbf{K}_t \mathbf{H}_t) \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^T = \boldsymbol{\Sigma}_t \mathbf{H}_t^T \quad (8.94)$$

where we used Equation (8.72) for $\boldsymbol{\Sigma}_t$ in the last line. \square

Now we prove our first theorem,

Theorem 2 (The EKF performs preconditioned gradient descent). *The update step in the EKF corresponds to the following gradient step*

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} - \boldsymbol{\Sigma}_t \nabla_{\boldsymbol{\mu}_{t|t-1}} \mathcal{L}_t(\mathbf{y}_t) \quad (8.95)$$

where

$$\mathcal{L}_t(\mathbf{y}) = -\log p(\mathbf{y}|\hat{\mathbf{y}}_t) = -\log p(\mathbf{y}|h(\boldsymbol{\mu}_{t|t-1}, \mathbf{u}_t)) \quad (8.96)$$

Proof. By definition of the EKF, we have $\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t \mathbf{e}_t$. By Lemma 1 and Lemma 2 we have

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t \mathbf{e}_t \quad (8.97)$$

$$= \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t \mathbf{R}_t \nabla_{\hat{\mathbf{y}}_t} \mathcal{L}_t(\mathbf{y}_t) \quad (8.98)$$

$$= \boldsymbol{\mu}_{t|t-1} + \boldsymbol{\Sigma}_t \mathbf{H}_t^T \nabla_{\hat{\mathbf{y}}_t} \mathcal{L}_t(\mathbf{y}_t) \quad (8.99)$$

But $\mathbf{H}_t = \frac{\partial \hat{\mathbf{y}}_t}{\partial \boldsymbol{\mu}_{t|t-1}}$, so

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \boldsymbol{\Sigma}_t \nabla_{\boldsymbol{\mu}_{t|t-1}} \mathcal{L}_t(\mathbf{y}_t) \quad (8.100)$$

\square

We now prove another lemma.

Lemma 3 (Information filter). *The EKF update can be written in information form as follows:*

$$\Sigma_t^{-1} = \Sigma_{t|t-1}^{-1} + \mathbf{H}_t^\top \mathbf{R}_t^{-1} \mathbf{H}_t \quad (8.101)$$

Furthermore, for static dynamical systems, where $f(\mathbf{z}, \mathbf{u}) = \mathbf{z}$ and $\mathbf{Q}_t = \mathbf{0}$, the EKF becomes

$$\Sigma_t^{-1} = \Sigma_{t-1}^{-1} + \mathbf{H}_t^\top \mathbf{R}_t \mathbf{H}_t \quad (8.102)$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} - \Sigma_t \nabla_{\boldsymbol{\mu}_{t-1}} \mathcal{L}_t(\mathbf{y}_t) \quad (8.103)$$

Proof. We have

$$\Sigma_t = (\mathbf{I} - \mathbf{K}_t \mathbf{H}_t) \Sigma_{t|t-1} \quad (8.104)$$

$$= \Sigma_{t|t-1} - \Sigma_{t|t-1} \mathbf{H}_t^\top (\mathbf{H}_t \Sigma_{t|t-1} \mathbf{H}_t^\top + \mathbf{R}_t)^{-1} \mathbf{H}_t \Sigma_{t|t-1} \quad (8.105)$$

$$= (\Sigma_{t|t-1}^{-1} + \mathbf{H}_t^\top \mathbf{R}_t^{-1} \mathbf{H}_t)^{-1} \quad (8.106)$$

where we used the matrix inversion lemma in the last line. The second claim follows easily since for a static model we have $\boldsymbol{\mu}_{t|t-1} = \boldsymbol{\mu}_{t-1}$ and $\Sigma_{t|t-1} = \Sigma_{t-1}$. \square

Now we prove another lemma.

Lemma 4. *For exponential family models, the $\mathbf{H}_t^\top \mathbf{R}_t^{-1} \mathbf{H}_t$ term in the information filter is equal to the Fisher information matrix:*

$$\mathbf{H}_t^\top \mathbf{R}_t^{-1} \mathbf{H}_t = \mathbb{E}_{p(\mathbf{y}|\hat{\mathbf{y}}_t)} [\mathbf{g}_t \mathbf{g}_t^\top] \quad (8.107)$$

where $\mathbf{g}_t = \nabla_{\boldsymbol{\mu}_{t|t-1}} \mathcal{L}_t(\mathbf{y})$.

Proof. We will omit time indices for brevity. We have

$$\frac{\partial \mathcal{L}(\mathbf{y})}{\partial \boldsymbol{\mu}} = \frac{\partial \mathcal{L}(\mathbf{y})}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \boldsymbol{\mu}} = \frac{\partial \mathcal{L}(\mathbf{y})}{\partial \hat{\mathbf{y}}} \mathbf{H} \quad (8.108)$$

Hence

$$\mathbb{E}_{p(\mathbf{y})} [\nabla_{\boldsymbol{\mu}} \mathcal{L}_t(\mathbf{y}) \nabla_{\boldsymbol{\mu}} \mathcal{L}_t(\mathbf{y})^\top] = \mathbf{H}^\top \mathbb{E}_{p(\mathbf{y})} [\nabla_{\hat{\mathbf{y}}} \mathcal{L}_t(\mathbf{y}) \nabla_{\hat{\mathbf{y}}} \mathcal{L}_t(\mathbf{y})^\top] \mathbf{H} \quad (8.109)$$

But the middle term is the FIM wrt the mean parameters, which, from Main Equation (3.80), is \mathbf{R}^{-1} . \square

Finally we are able to prove Theorem 1.

Proof. From Lemma 3 and Lemma 4 we have

$$\Sigma_t^{-1} = \Sigma_{t-1}^{-1} + \mathbb{E}_{p(\mathbf{y}|\hat{\mathbf{y}}_t)} [\nabla_{\boldsymbol{\mu}_{t-1}} \mathcal{L}_t(\mathbf{y}) \nabla_{\boldsymbol{\mu}_{t-1}} \mathcal{L}_t(\mathbf{y})^\top] \quad (8.110)$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} - \Sigma_t \nabla_{\boldsymbol{\mu}_{t-1}} \mathcal{L}_t(\mathbf{y}) \quad (8.111)$$

1
2 If we define $\mathbf{J}_t = \boldsymbol{\Sigma}_t^{-1}/(t + 1)$, this becomes

3
4
$$\mathbf{J}_t = \frac{t}{t+1} \mathbf{J}_{t-1} + \frac{1}{t+1} \mathbb{E}_{p(\mathbf{y}|\hat{\mathbf{y}}_t)} \left[\nabla_{\boldsymbol{\mu}_{t-1}} \mathcal{L}_t(\mathbf{y}) \nabla_{\boldsymbol{\mu}_{t-1}} \mathcal{L}_t(\mathbf{y})^\top \right] \quad (8.112)$$

5
6
$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} - \frac{1}{t+1} \mathbf{J}_t^{-1} \nabla_{\boldsymbol{\mu}_{t-1}} \mathcal{L}_t(\mathbf{y}) \quad (8.113)$$

7 \square

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

9 Inference for graphical models

9.1 Belief propagation on trees

9.1.1 BP for polytrees

In this section, we generalize the forwards-backwards algorithm (i.e., two-filter smoothing) for chain-structured PGMs to work on a **polytree**, which is a directed graph whose undirected “backbone” is a tree, i.e., a graph with no loops. (That is, a polytree is a directed tree with multiple root nodes, in which a node may have multiple parents, whereas in a singly rooted tree, each node has a single parent.) This algorithm is called **belief propagation** and is due to [Pearl88].

We consider the case of a general discrete node X with parents U_i and children Y_j . We partition the evidence in the graph, e , into the evidence upstream of node X , e_X^+ , and all the rest, e_X^- . Thus e_X^+ contains all the evidence separated from X if its incoming arcs were deleted, and e_X^- contains the evidence below X and the evidence in X itself, if any. The posterior on node X can be computed as follows:

$$\text{bel}_X(x) \triangleq p(X = x|e) = c' \lambda_X(x) \pi_X(x) \quad (9.1)$$

$$\lambda_X(x) \triangleq p(e_X^-|X = x) \quad (9.2)$$

$$\pi_X(x) \triangleq p(X = x|e_X^+) \quad (9.3)$$

where c' is a normalizing constant.

Consider the graph shown in Figure 9.1. We will use the notation $e_{U_1 \rightarrow X}^+$ to denote the evidence 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 below the edge from X to Y_1 (i.e., in the triangle below Y_1). We use e_X to denote the local evidence attached to node X (if any).

We can compute λ_X as follows, using the fact that X ’s children are independent given X . In particular, the evidence in the subtrees rooted at each child, and the evidence in X itself (if any), are conditionally independent given X .

$$p(e_X^-|X = x) = p(e_X|X = x)p(e_{X \rightarrow Y_1}^-|X)p(e_{X \rightarrow Y_2}^-|X) \quad (9.4)$$

If we define the λ “message” that a node X sends to its parents U_i as

$$\lambda_{X \rightarrow U_i}(u_i) \triangleq p(e_{U_i \rightarrow X}^-|U_i = u_i) \quad (9.5)$$

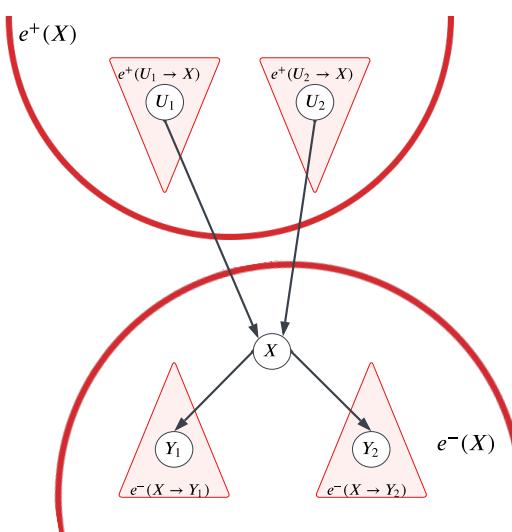


Figure 9.1: Message passing on a polytree.

we can write in general that

$$\lambda_X(x) = \lambda_{X \rightarrow X}(x) \times \prod_j \lambda_{Y_j \rightarrow X}(x) \quad (9.6)$$

where $\lambda_{X \rightarrow X}(x) = p(e_X | X = x)$. For leaves, we just write $\lambda_{X \rightarrow U_i}(u_i) = 1$, since there is no evidence below X .

We compute π_X by introducing X 's parents, to break the dependence on the upstream evidence, and then summing them out. We partition the evidence above X into the evidence in each subtree above each parent U_i .

$$p(X = x | e_X^+) = \sum_{u_1, u_2} p(X = x, U_1 = u_1, U_2 = u_2 | e_X^+) \quad (9.7)$$

$$= \sum_{u_1, u_2} p(X = x | u_1, u_2) p(u_1, u_2 | e_{U_1 \rightarrow X}^+, e_{U_2 \rightarrow X}^+) \quad (9.8)$$

$$= \sum_{u_1, u_2} p(X = x | u_1, u_2) p(u_1 | e_{U_1 \rightarrow X}^+) p(u_2 | e_{U_2 \rightarrow X}^+) \quad (9.9)$$

If we define the π “message” that a node X sends to its children Y_j as

$$\Pi_{X \rightarrow Y_j}(x) \triangleq p(X = x | e_{X \rightarrow Y_j}^+) \quad (9.10)$$

we can write in general that

$$\pi_X(x) = \sum_u p(X = x | u) \prod_i \Pi_{U_i \rightarrow X}(u_i) \quad (9.11)$$

For root nodes, we write $\pi_X(x) = p(X = x)$, which is just the prior (independent of the evidence).

9.1.1.1 Computing the messages

We now describe how to recursively compute the messages. We initially focus on the example in Figure 9.1. First we compute the λ message.

$$\lambda_{X \rightarrow U_1}(u_1) = p(e_X^-, e_{U_2 \rightarrow X}^+ | u_1) \quad (9.12)$$

all the ev. except in the U_1 triangle

$$= \sum_x \sum_{u_2} p(e_X^-, e_{U_2 \rightarrow X}^+ | u_1, u_2, x) p(u_2, x | u_1) \quad (9.14)$$

$$= \sum_x \sum_{u_2} p(e_X^- | x) p(e_{U_2 \rightarrow X}^+ | u_2) p(u_2, x | u_1) \quad (9.15)$$

since X separates the U_2 triangle from e_X^- , and U_2 separates the U_2 triangle from U_1

$$= c \sum_x \sum_{u_2} p(e_X^- | x) \frac{p(u_2 | e_{U_2 \rightarrow X}^+)}{p(u_2)} p(x | u_2, u_1) p(u_2 | u_1) \quad (9.17)$$

using Bayes' rule, where $c = p(e_{U_2 \rightarrow X}^+)$ is a constant

$$= c \sum_x \sum_{u_2} p(e_X^- | x) p(u_2 | e_{U_2 \rightarrow X}^+) p(x | u_2, u_1) \quad (9.19)$$

since U_1 and U_2 are marginally independent

$$= c \sum_x \sum_{u_2} \lambda_X(x) \Pi_{U_2 \rightarrow X}(u_2) p(x | u_2, u_1) \quad (9.21)$$

In general, we have

$$\lambda_{X \rightarrow U_i}(u_i) = c \sum_x \lambda_X(x) \left[\sum_{u_k : k \neq i} p(X = x | \mathbf{u}) \prod_{k \neq i} \Pi_{U_k \rightarrow X}(u_k) \right] \quad (9.22)$$

If the graph is a rooted tree (as opposed to a polytree), each node has a unique parent, and this simplifies to

$$\lambda_{X \rightarrow U_i}(u_i) = c \sum_x \lambda_X(x) p(X = x | u_i) \quad (9.23)$$

Finally, we derive the π messages. We note that $e_{X \rightarrow Y_j}^+ = e - e_{X \rightarrow Y_j}^-$, so $\Pi_{X \rightarrow Y_j}(x)$ is equal to $\text{bel}_X(x)$ when the evidence $e_{X \rightarrow Y_j}^-$ is suppressed:

$$\Pi_{X \rightarrow Y_j}(x) = c' \pi_X(x) \lambda_{X \rightarrow X}(x) \prod_{k \neq j} \lambda_{Y_k \rightarrow X}(x) \quad (9.24)$$

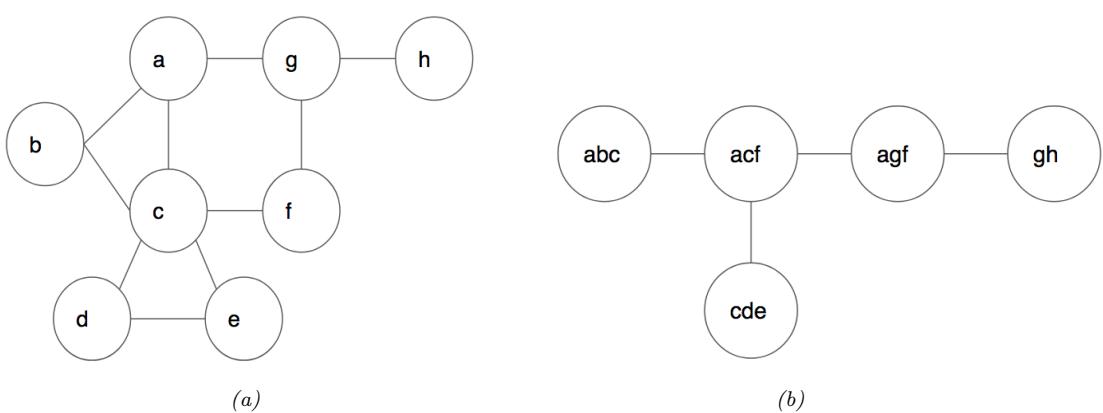


Figure 9.2: (a) An undirected graph. (b) Its corresponding junction tree.

9.1.1.2 Message passing protocol

We must now specify the order in which to send the messages. If the graph is a polytree, we can pick an arbitrary node as root. In the first pass, we send messages to it. If we go with an arrow, the messages are π messages; if we go against an arrow, the messages are λ messages. On the second pass, we send messages down from the root.

If the graph is a regular tree (not a polytree), there already is a single root. Hence the first pass will only consist of sending λ messages, and the second pass will only consist of sending π messages. This is analogous to a reversed version of the forwards-backwards algorithm, where we first send backwards likelihood messages to the root (node z_1) and then send them forwards posterior messages to the end of the chain (node z_T).

9.2 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 Main Section 9.5.5. The basic idea is to convert the graph into a tree, and then to run belief propagation on the tree. We summarize the main ideas below. For more details, see e.g., [Lauritzen96; Huang1996; Cowell99; Jensen07; KollerBook; Vandenbergh2015].

9.2.1 Tree decompositions

A **junction tree**, also called a **join tree** or **clique tree**, is a tree-structured graph, derived from the original graph, which satisfies certain key properties that we describe below; these properties ensure that local message passing results in global consistency. Note that junction trees have many applications in mathematics beyond probabilistic inference (see e.g., [Vandenbergh2015]). Note also that we can create a directed version of a junction tree, known as a **Bayes tree**, which is useful for incremental inference [Dellaert2017].

The process of converting a graph into a junction tree is called **tree decomposition** [Halin1976; Robertson1984; Heinz2013; Satzinger2015; Chekuri2014; Vandenberghe2015], which we summarize below.

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 9.2. 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 9.2, 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 9.2, 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 9.2, 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

$$\text{width}(T) = \max_{t \in T} |X_t| \quad (9.25)$$

For example, the width of the jtree in Figure 9.2(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:

$$\text{treewidth}(G) \triangleq \left(\min_{T \in \mathcal{T}(G)} \text{width}(T) \right) - 1 \quad (9.26)$$

We see that the treewidth of a tree is 1, and the treewidth of the graph in Figure 9.2(a) is 2.

9.2.1.1 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 Main Section 9.6), solving sparse systems of linear equations (see e.g., [Paskin03jtree]), etc.

9.2.1.2 Computing a tree decomposition

We now describe an algorithm known as **triangulation** or **elimination** for constructing a junction tree from an undirected graph. We first choose an ordering of the nodes, π . (See Main Section 9.5.3

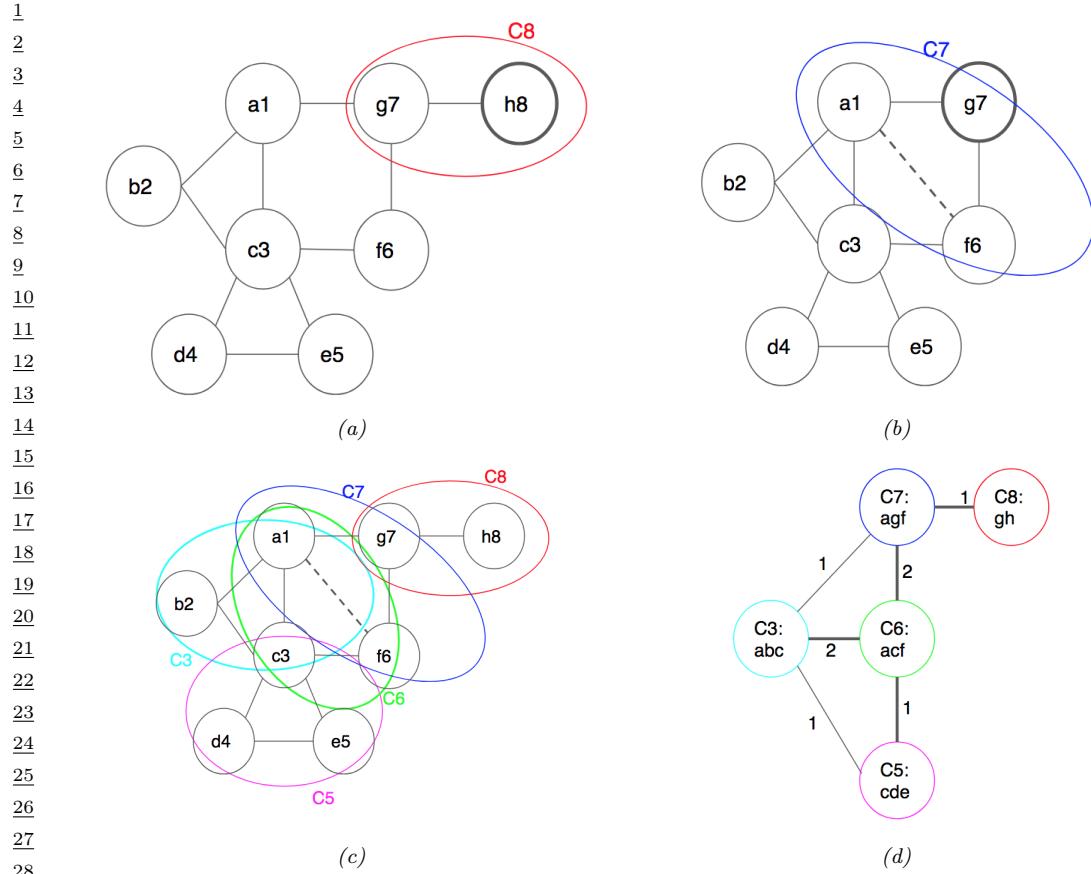


Figure 9.3: (a-b) Illustration of two steps of graph triangulation using the elimination order (a, b, c, d, e, f, g, h) applied to the graph in Figure 9.2a. The node being eliminated is shown with a darker border. Cliques are numbered by the vertex that created them. The dotted a-f line is a fill-in edge created when node g is eliminated. (c) Corresponding set of maximal cliques of the chordal graph. (d) Resulting junction graph.

for a discussion of how to choose a good elimination ordering.) 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 9.3, 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 9.3(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 9.3(b). We continue in this way until all nodes are eliminated, as shown in Figure 9.3(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 9.3(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 simple discard it, otherwise we add it to our list of cliques. For example, when triangulating the graph in Figure 9.3, 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 this edge to be $|C_i \cap C_j|$, i.e., the number of variables they have in common. One can show [Jensen94; Ajtai00] that any maximal weight spanning tree (MST) of the junction graph is a junction tree. This is illustrated in Figure 9.3d, which corresponds to the jtree in Figure 9.2b.

9.2.1.3 Computing a jtree from a directed graphical model

In this section, we show how to create a junction tree from a DPGM. For example, consider the “student” network from Main Figure 4.38(a). We can “moralize” this (by connecting unmarried parents with a common child, and then dropping all edge orientations), to get the undirected graph in Main Figure 4.38(b). We can then derive a tree decomposition by applying the variable elimination algorithm from Main Section 9.5. 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 9.4. We see that the nodes of the jtree T are cliques of the chordal graph:

$$\mathcal{C}(T) = \{C, D\}, \{G, I, D\}, \{G, S, I\}, \{G, J, S, L\}, \{H, G, J\} \quad (9.27)$$

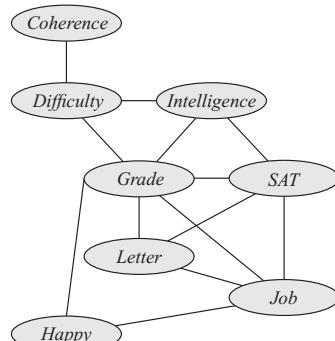
9.2.1.4 Tree decompositions of some common graph structures

In Figure 9.5, we illustrate the tree decomposition of several common graph structures which arise when using neural networks and graphical models. The resulting decomposition can be used to trade off time and memory, by storing checkpoints to partition the graph into subgraphs, and then recomputing intermediate quantities on demand; for details, see e.g., [Griewank2008; Binder97island; Zweig00; Chen2016sublinear]. For example, for a linear chain, we can reduce the memory from $O(T)$ to $O(\log T)$, if we are willing to increase the runtime from $O(T)$ to $O(T \log T)$.

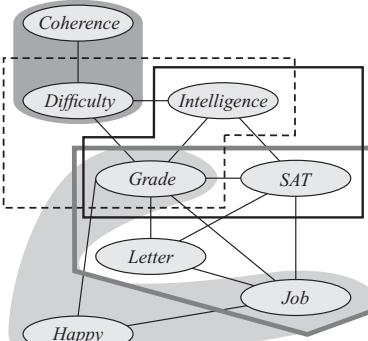
Another common graph structure is a 2d grid. If the grid has size $w \times h$, then the treewidth is $\min(w, h)$. To see this, note that we can convert the grid into a chain by grouping together all the nodes in each column or each row, depending on which is smaller. (See [Lipton79] for the formal proof.)

Note that a graph may look like it is triangulated, even though it is not. For example, Figure 9.6(a) is made of little triangles, but it is not triangulated, since it contains the chordless 5-cycle 1-2-3-4-5-1. A triangulated version of this graph is shown in Figure 9.6(b), in which we add two fill-in edges.

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



(a)



(b)

13
14
15
16
17
18
19



(c)

20 Figure 9.4: (a) A triangulated version of the (moralized) student graph from Main Figure 4.38(b). The extra
21 fill-in edges (such as $G-S$) are derived from the elimination ordering used in Main Figure 9.18. (b) The
22 maximal cliques. (c) The junction tree. From Figure 9.11 of [KollerBook]. Used with kind permission of
23 Daphne Koller.

24
25
26
27

28 9.2.2 Message passing on a junction tree

29

30 In this section, we discuss how to extend the belief propagation algorithm of Main Section 9.3 to
31 work with junction trees. This will let us compute the exact marginals in time linear in the size of
32 the tree. We focus on the **Lauritzen-Spiegelhalter** algorithm [Lauritzen88], although there are
33 many other variants (see e.g., [Huang1996; Jensen07]).

34
35

36 9.2.2.1 Potential functions

37

38 We present the algorithm abstractly in terms of potential functions ϕ_i associated with each node
39 (clique) in the junction tree. A potential function is just a non-negative function of its arguments. If
40 the arguments are discrete, we can represent potentials as multi-dimensional arrays (tensors). We
41 discuss the Gaussian case in Main Section 2.3.3, and the general case in Section 9.2.3.

42 We assume each potential has an identity element, and that there is a way to multiply, divide and
43 marginalize potentials. For the discrete case, the identity element is a vector of all 1s. To explain
44 marginalization, suppose clique i has domain C_i , let $S_{ij} = C_i \cap C_j$ be the separator between node
45 i and j . Let us partition the domain of ϕ_i into S_{ij} and $C'_i = C_i \setminus S_{ij}$, where C'_i are the variables
46 that are unique to i and not shared with S_{ij} . We denote marginalization of potential $\phi_i(C_i)$ onto the
47

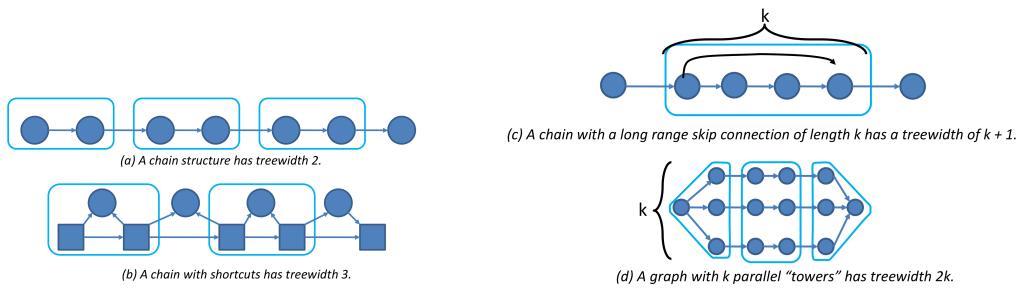


Figure 9.5: Examples of optimal tree decompositions for some common graph structures. Adapted from <https://bit.ly/2m5vauG>.

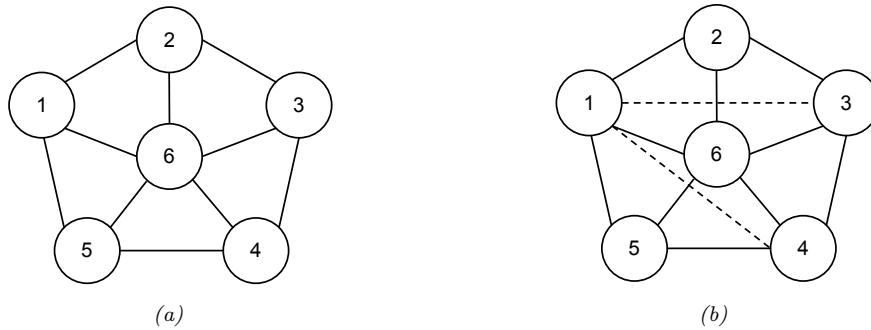


Figure 9.6: A triangulated graph is not just composed of little triangles. Left: this graph is not triangulated, despite appearances, since it contains a chordless 5-cycle 1-2-3-4-5-1. Right: one possible triangulation, by adding the 1-3 and 1-4 fill-in edges. Adapted from [Armstrong05]

domain S_{ij} as follows

$$\phi_{ij}(S_{ij}) = \phi_i(C_i) \downarrow S_{ij} = \sum_{C'_i \in C_i \setminus S_{ij}} \phi_i(C'_i, S_{ij}) \quad (9.28)$$

We define multiplication elementwise, by broadcast ϕ_{ij} over C'_j .

$$(\phi_j * \phi_{ij})(C_j) = \phi_j(C'_j, S_{ij}) \phi_{ij}(S_{ij}) \quad (9.29)$$

where $C_j = (C'_j, S_{ij})$. We define division similarly:

$$\left(\frac{\phi_j}{\phi_{ij}} \right)(C_j) = \frac{\phi_j(C'_j, S_{ij})}{\phi_{ij}(S_{ij})} \quad (9.30)$$

where $0/0 = 0$. (The **Shafer-Shenoy** version of the algorithm, from [Shafer90], avoids division by keeping track of the individual terms and multiplying all but one of them on demand.)

We can interpret division as computing a conditional distribution, since $\phi_j^* = \phi_j / \phi_{ij} = p(C'_j, S_{ij}) / p(S_{ij}) = p(C'_j | S_{ij})$. Similarly we can interpret multiplication as adding updated information back in. To see

```

1
2 // Collect to root
3 for each node n in post-order
4   p = parent(n)
5    $\phi_{np} = \phi_n \downarrow S_{np}$ 
6    $\phi_p = \phi_p * \phi_{np}$ 
7
8 // Distribute from root
9 for each node n in pre-order
10   for each child c of n
11      $\phi_c = \frac{\phi_c}{\phi_{nc}}$ 
12      $\phi_{nc} = \phi_n \downarrow S_{nc}$ 
13      $\phi_c = \phi_c * \phi_{nc}$ 
14
15
16
17

```

Figure 9.7: Message passing on a (directed) junction tree.

18 this, let $\phi_{ij}^* = p(S_{ij}|e)$ be the new separator potential, where e is some evidence. Let $\phi_j^{**} = \phi_j^* * \phi_{ij}^*$
 19 be the resulting of dividing out the old separator and multiplying in the new separator. Then
 20 $\phi_j^{**} \propto p(C'_j, S_{ij}|e)$. So we have successfully passed information from i to j . We will leverage this
 21 result below.

22

23 9.2.2.2 Initialization

24 To initialize the junction tree potentials, we first assign each factor F_k to a unique node $j = A_k$ such
 25 that the domain of node j contains all of F_k 's variables. Let $A_i^{-1} = \{k : A_k = i\}$ be all the factors
 26 assigned to node i . We set the node potentials to
 27

$$28 \quad \phi_i = \prod_{k \in A_i^{-1}} F_k \tag{9.31}$$

30 where $\phi_i = 1$ if no factors are assigned to i . We set the separator potentials to $\phi_{ij} = 1$.

32

33 9.2.2.3 Calibration

34 We now describe a simple serial ordering for sending messages on the junction tree. We first pick
 35 an arbitrary node as root. Then the algorithm has two phases, similar to forwards and backwards
 36 passes over a chain (see Main Section 9.2.3).
 37

In the **collect evidence** phase, we visit nodes in post-order (children before parents), and each
 38 node n sends a message to its parents p , until we reach the root. The parent p first divides out any
 39 information it received (via the separator) from its child n by computing
 40

$$41 \quad \phi_p = \frac{\phi_p}{\phi_{np}} \tag{9.32}$$

43 However, since the separator potentials are initialized to 1s, this operation is not strictly necessary.
 44 Next we compute the message from child to parent by computing an updated separator potential:
 45

$$46 \quad \phi_{np} = \phi_n \downarrow S_{np} \tag{9.33}$$

47

2 Finally the parent “**absorbs the flow**” from its child by computing

3

$$\phi_p = \phi_p * \phi_{np} \quad (9.34)$$

4

5 In the **distribute evidence** phase we visit nodes in pre-order (parents before children), and each
6 node n sends a message to each child c , starting with the root. In particular, each child divides out
7 message it previously sent to its parent n :

8

9

$$\phi_c = \frac{\phi_c}{\phi_{nc}} \quad (9.35)$$

10

11 Then we compute the new message from parent to child:

12

13

$$\phi_{nc} = \phi_n \downarrow S_{nc} \quad (9.36)$$

14

15 Finally the child absorbs this new information:

16

17

$$\phi_c = \phi_c * \phi_{nc} \quad (9.37)$$

18

19 The overall process is sometimes called “**calibrating**” the jtree [Lauritzen88]. See Figure 9.7 for
20 the pseudocode.

21

22 9.2.3 The generalized distributive law

23

24 We have seen how we can define potentials for discrete and Gaussian distributions, and we can then
25 use message passing on a junction tree to efficiently compute posterior marginals, as well as the
26 likelihood of the data. For example, consider a graphical model with pairwise potentials unrolled for
27 4 time steps. The partition function is defined by

28

$$Z = \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \psi_{12}(x_1, x_2) \psi_{23}(x_2, x_3) \psi_{34}(x_3, x_4) \quad (9.38)$$

29

30 We can distribute sums over products to compute this more cheaply as follows:

31

32

$$Z = \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{23}(x_2, x_3) \sum_{x_4} \psi_{34}(x_3, x_4) \quad (9.39)$$

33

34 By defining suitable implementations of the sum and multiplication operations, we can use this same
35 trick to solve a variety of problems. This general formulation is called the **generalized distributive**
36 law [Aji00].

37

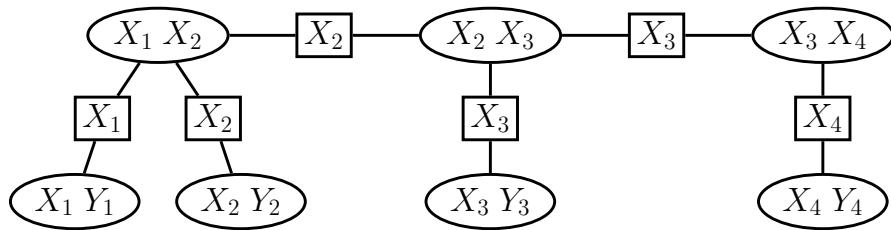
38 The key property we require is that the local clique functions ψ_c are associated with a **commutative**
39 **semi-ring**. This is a set \mathcal{K} , together with two binary operations called “ $+$ ” and “ \times ”, which satisfy
40 the following three axioms:

41

- 42 1. The operation “ $+$ ” is associative and commutative, and there is an additive identity element called
43 “0” such that $k + 0 = k$ for all $k \in \mathcal{K}$.
- 44
- 45 2. The operation “ \times ” is associative and commutative, and there is a multiplicative identity element
46 called “1” such that $k \times 1 = k$ for all $k \in \mathcal{K}$.
- 47

	Domain	+	\times	Name
2	$[0, \infty)$	$(+, 0)$	$(\times, 1)$	sum-product
3	$[0, \infty)$	$(\max, 0)$	$(\times, 1)$	max-product
4	$(-\infty, \infty]$	(\min, ∞)	$(+, 0)$	min-sum
5	$\{T, F\}$	(\vee, F)	(\wedge, T)	Boolean satisfiability

Table 9.1: Some commutative semirings.

Figure 9.8: The junction tree derived from an HMM of length $T = 4$.

3. The **distributive law** holds, i.e.,

$$(a \times b) + (a \times c) = a \times (b + c) \quad (9.40)$$

for all triples (a, b, c) from \mathcal{K} .

There are many such semi-rings; see Table 9.1 for some examples. We can therefore use the JTA to solve many kinds of problems, such as: computing posterior marginals (as we have seen); computing posterior samples [Dawid92]; computing the N most probable assignments [Nilsson98]; constraint satisfaction problems [Bistarelli97; Dechter03; Dechter2019]; logical reasoning problems [Amir05]; solving linear systems of the form $\mathbf{Ax} = \mathbf{b}$ where \mathbf{A} is a sparse matrix [Blair92; Paskin03jtree; Bickson09]; etc. See [Lauritzen97] for more details.

9.2.4 JTA applied to a chain

It is interesting to see what happens if we apply the junction tree algorithm to a chain structured graph such as an HMM. A detailed discussion can be found in [Smyth97], but the basic idea is as follows. First note that for a pairwise graph, the cliques are the edges, and the separators are the nodes, as shown in Figure 9.8. We initialize the potentials as follows: we set $\psi_s = 1$ for all the separators, we set $\psi_c(x_{t-1}, x_t) = p(x_t | x_{t-1})$ for clique $c = (X_{t-1}, X_t)$, and we set $\psi_c(x_t, y_t) = p(y_t | x_t)$ for clique $c = (X_t, Y_t)$.

Next we send messages from left to right along the “backbone”, and from observed child leaves up to the backbone. Consider the clique $j = (X_{t-1}, X_t)$ and its two children, $i = (X_{t-2}, X_{t-1})$, and $i' = (X_t, Y_t)$. To compute the new clique potential for j , we first marginalize the clique potentials for

i onto S_{ij} and for i' onto $S_{i'j}$ to get

$$\psi_{ij}^*(X_{t-1}) = \sum_{X_{t-2}} \psi_i(X_{t-2}, X_{t-1}) = p(X_{t-1} | \mathbf{y}_{1:t-1}) = \alpha_{t-1})(X_t) \quad (9.41)$$

$$\psi_{i'j}^*(X_t) = \sum_{Y_t} \psi_i(X_t, Y_t) \propto p(Y_t | X_t) = \lambda_t(X_t) \quad (9.42)$$

We then absorb messages from these separator potentials to compute the new clique potential:

$$\psi_i^*(X_{t-1}, X_t) \propto \psi_i(X_{t-1}, X_t) \frac{\psi_{ij}^*(X_{t-1})}{\psi_{ij}(X_{t-1})} \frac{\psi_{i'j}(X_t)}{\psi_{i'j}(X_t)} \quad (9.43)$$

$$= A(X_{t-1}, X_t) \frac{\alpha_{t-1}(X_{t-1})}{1} \frac{\lambda_t(X_t)}{1} \propto p(X_{t-1}, X_t | \mathbf{y}_{1:t}) \quad (9.44)$$

which we recognize as the filtered two-slice marginal.

Now consider the backwards pass. Let $k = (X_t, X_{t+1})$ be the parent of j . We send a message from k to j via their shared separator $S_{k,j}(X_t)$ to get the final potential:

$$\psi_j^{**}(X_{t-1}, X_t) \propto \psi_j^*(X_{t-1}, X_t) \frac{\psi_{k,j}^{**}(X_t)}{\psi_i^*(X_t)} \quad (9.45)$$

$$= [A(X_{t-1}, X_t) \alpha_{t-1}(X_{t-1}) \lambda_t(X_t)] \frac{\gamma_t(X_t)}{\alpha^*(X^*)} \quad (9.46)$$

$$\propto p(X_{t-1}, X_t | \mathbf{y}_{1:T}) \quad (9.47)$$

where $\alpha_t(X_t) = p(X_t|\mathbf{y}_{1:t})$ and $\gamma_t(X_t) = p(X_t|\mathbf{y}_{1:T})$ are the separator potentials for S_{jk} on the forwards and backwards passes. This matches the two slice smoothed marginal in Main Equation (9.35).

9.2.5 JTA for general temporal graphical models

In this section, we discuss how to perform exact inference in temporal graphical models, which includes dynamic Bayes nets (Main Section 29.5.5) and their undirected analogs.

The simplest approach to inference in such models is to **flatten** the model into a chain, by defining a **mega-variable** z_t whose state space is the cross product of all the individual hidden variables in slice t , and then to compute the corresponding transition matrix. For example, suppose we have two independent binary chains, with transition matrices given by

$$\mathbf{A}_1 = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \quad \mathbf{A}_2 = \begin{pmatrix} e & f \\ g & h \end{pmatrix} \quad (9.48)$$

Then the transition matrix of the flattened model has the following Kronecker product form:

$$\mathbf{A} = \mathbf{A}_1 \otimes \mathbf{A}_2 = \begin{pmatrix} ae & af & be & bf \\ ag & ah & bg & bh \\ ce & cf & de & df \\ cq & ch & dq & dh \end{pmatrix} \quad (9.49)$$

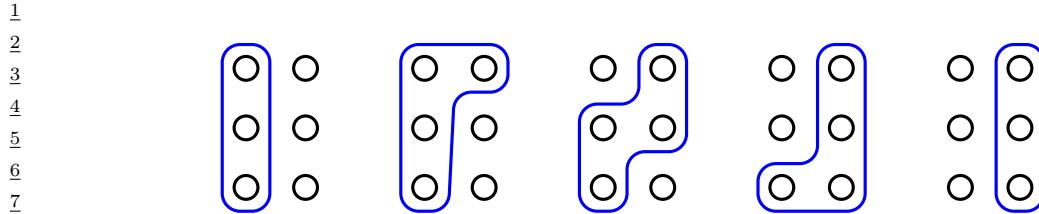


Figure 9.9: The cliques in the junction tree decomposition as we advance the frontier from one time-slice to the next in a 3-chain factorial HMM model.

For example, the probability of going from state $(1,2)$ to $(2,1)$ is $p(1 \rightarrow 2) \times p(2 \rightarrow 1) = b \times g$. We note that this is not a sparse matrix, even though the chains are completely independent.

One can use this expanded matrix inside the forwards-backwards algorithm to compute $p(z_{1:t}, z_{2:t} | \mathbf{y}_{1:T}, \theta)$, from which the marginals of each chain, $p(z_{i,t} | \mathbf{y}_{1:T}, \theta)$, can easily be derived. If each hidden node has K states, and there are M hidden nodes per time step, the transition matrix has size $(K^M) \times (K^M)$, so this method takes $O(TK^{2M})$ time, which is often unacceptably slow.

Of course, the above method ignores the structure within each time slice. For example, the above flattened matrix does not exploit the fact that the chains are completely independent. By using the JTA, we can derive a more efficient algorithm. For example, consider the 3-chain factorial HMM (FHMM) in Main Figure 29.19a. All the hidden variables x_{mt} within a time slice become correlated due to the observed common child y_t (explaining away), so the exact belief state $p(x_{1:M,t} | \mathbf{y}_{1:t}, \theta)$ will necessarily have size $O(K^M)$. However, rather than multiplying this large vector by a $(K^M) \times (K^M)$ matrix, we can update the belief state one variable at a time, as illustrated in Figure 9.9. This takes $O(TM K^{M+1})$ time (see [Ghahramani97] for the details of the algorithm). This method has been called the **frontier algorithm** [Zweig96b], since it sweeps a “frontier” across the network (forwards and backwards); however, this is just a special case of the JTA. For a detailed discussion of how to apply the JTA to temporal graphical models, see [Bilmes10].

Although the JTA for FHMMs is better than the naive approach to inference, it still takes time exponential in the number of hidden nodes per chain (ignoring any transient nodes that do not connect across time). For the FHMM, this is unavoidable, since all the hidden variables immediately become correlated within a single time slice due to the observed common child y_t . What about for graphs with sparser structure? For example, consider the coupled HMM in Main Section 29.5.4. Here each hidden node only depends on two nearest neighbors and some local evidence. Thus initially the belief state can be factored. However, after $T = M$ time steps, the belief state becomes fully correlated, because there is now a direct path of influence between variables in non-neighboring chains. This is known as the **entanglement** problem, and it means that, in general, exact inference in temporal graphical models is exponential in the number of (persistent) hidden variables. Looking carefully at Main Figure 29.19a, this is perhaps not so surprising, since the model looks like a short and wide grid-structured graph, for which exact inference is known to be intractable in general.

Fortunately, we can still leverage sparse graph structure when performing *approximate* inference. The intuition is that although all the variables may be correlated, the correlation between distant variables is likely to be weak. In Section 10.3.2, we derive a structured mean field approximation for FHMMs, which exploits the parallel chain structure. This only takes $O(TM K^2 I)$ time, where I is

the number of iterations of the inference algorithm (typically $I \sim 10$ suffices for good performance). See also [Boyen98] for an approach based on assumed density filtering.

Note that the situation with linear dynamical systems is somewhat different. In that context, combining multiple hidden random variables merely increases the size of the state space additively rather than multiplicatively. Thus inference takes $O(T(CL)^3)$ time, if there are T time steps, and C hidden chains each with dimensionality L . Furthermore, two independent chains combine to produce a sparse block-diagonal transition weight matrix, rather than a dense Kronecker product matrix, so the structural information is not “lost”.

9.3 MAP estimation for discrete PGMs

In this section, we consider the problem of finding the most probable configuration of variables in a probabilistic graphical model, i.e., our goal is to find a MAP assignment $\mathbf{x}^* = \arg \max_{\mathbf{x} \in \mathcal{X}^V} p(\mathbf{x})$, where $\mathcal{X} = \{1, \dots, K\}$ is the discrete state space of each node, V is the number of nodes, and the distribution is defined according to a Markov Random field (Main Section 4.3) with pairwise cliques, one per edge:

$$p(\mathbf{x}) = \frac{1}{Z} \exp \left\{ \sum_{s \in \mathcal{V}} \theta_s(x_s) + \sum_{(s,t) \in \mathcal{E}} \theta_{st}(x_s, x_t) \right\} \quad (9.50)$$

Here $\mathcal{V} = \{x_1, \dots, x_V\}$ are the nodes, \mathcal{E} are the edges, θ_s and θ_{st} are the node and edge potentials, and Z is the partition function:

$$Z = \sum_{\mathbf{x}} \exp \left\{ \sum_{s \in \mathcal{V}} \theta_s(x_s) + \sum_{(s,t) \in \mathcal{E}} \theta_{st}(x_s, x_t) \right\} \quad (9.51)$$

Since we just want the MAP configuration, we can ignore Z , and just compute

$$\mathbf{x}^* = \operatorname{argmax}_{\mathbf{x}} \sum_{s \in \mathcal{V}} \theta_s(x_s) + \sum_{(s,t) \in \mathcal{E}} \theta_{st}(x_s, x_t) \quad (9.52)$$

We can compute this exactly using dynamic programming as we explain in Section 9.2; However, this takes time exponential in the treewidth of the graph, which is often too slow. In this section, we focus on approximate methods that can scale to intractable models. We only give a brief description here; more details can be found in [Monster; KollerBook].

9.3.1 Notation

To simplify the presentation, we write the distribution in the following form:

$$p(\mathbf{x}) = \frac{1}{Z(\boldsymbol{\theta})} \exp(-\mathcal{E}(\mathbf{x})) \quad (9.53)$$

$$\mathcal{E}(\mathbf{x}) \triangleq -\boldsymbol{\theta}^\top \mathcal{T}(\mathbf{x}) \quad (9.54)$$

where $\boldsymbol{\theta} = (\{\theta_{s;j}\}, \{\theta_{s,t;j,k}\})$ are all the node and edge parameters (the canonical parameters), and $\mathcal{T}(\mathbf{x}) = (\{\mathbb{I}(x_s = j)\}, \{\mathbb{I}(x_s = j, x_t = k)\})$ are all the node and edge indicator functions (the sufficient statistics). 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 known as the mean parameters of the model, and are given by

$$\boldsymbol{\mu} = \mathbb{E}[\mathcal{T}(\mathbf{x})] = (\{p(x_s = j)\}_s, \{p(x_s = j, x_t = k)\}_{s \neq t}) = (\{\mu_{s;j}\}_s, \{\mu_{st;jk}\}_{s \neq t}) \quad (9.55)$$

This is a vector of length $d = KV + K^2E$, where $K = |\mathcal{X}|$ is the number of states, $V = |\mathcal{V}|$ is the number of nodes, and $E = |\mathcal{E}|$ is the number of edges. Since $\boldsymbol{\mu}$ completely characterizes the distribution $p(\mathbf{x})$, so we sometimes treat $\boldsymbol{\mu}$ as a distribution itself.

Equation (9.55) is called the **standard overcomplete representation**. It is called “overcomplete” because it ignores the sum-to-one constraints. In some cases, it is convenient to remove this redundancy. For example, consider an Ising model where $X_s \in \{0, 1\}$. The model can be written as

$$p(\mathbf{x}) = \frac{1}{Z(\boldsymbol{\theta})} \exp \left\{ \sum_{s \in \mathcal{V}} \theta_s x_s + \sum_{(s,t) \in \mathcal{E}} \theta_{st} x_s x_t \right\} \quad (9.56)$$

Hence we can use the following minimal parameterization

$$\mathcal{T}(\mathbf{x}) = (x_s, s \in V; x_s x_t, (s, t) \in \mathcal{E}) \in \mathbb{R}^d \quad (9.57)$$

where $d = V+E$. The corresponding mean parameters are $\mu_s = p(x_s = 1)$ and $\mu_{st} = p(x_s = 1, x_t = 1)$.

9.3.2 The marginal polytope

The space of allowable $\boldsymbol{\mu}$ vectors is called the **marginal polytope**, and is denoted $\mathbb{M}(G)$, where G is the structure of the graph. This is defined to be the set of all mean parameters for the given model that can be generated from a valid probability distribution:

$$\mathbb{M}(G) \triangleq \{\boldsymbol{\mu} \in \mathbb{R}^d : \exists p \text{ s.t. } \boldsymbol{\mu} = \sum_{\mathbf{x}} \mathcal{T}(\mathbf{x}) p(\mathbf{x}) \text{ for some } p(\mathbf{x}) \geq 0, \sum_{\mathbf{x}} p(\mathbf{x}) = 1\} \quad (9.58)$$

For example, consider an Ising model. If we have just two nodes connected as $X_1 - X_2$, one can show that we have the following minimal set of constraints: $0 \leq \mu_{12}, 0 \leq \mu_{12} \leq \mu_1, 0 \leq \mu_{12} \leq \mu_2$, and $1 + \mu_{12} - \mu_1 - \mu_2 \geq 0$. We can write these in matrix-vector form as

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \\ -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_{12} \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \end{pmatrix} \quad (9.59)$$

These four constraints define a series of half-planes, whose intersection defines a polytope, as shown in Figure 9.10(a).

Since $\mathbb{M}(G)$ is obtained by taking a convex combination of the $\mathcal{T}(\mathbf{x})$ vectors, it can also be written as the convex hull of these vectors:

$$\mathbb{M}(G) = \text{conv}\{\mathcal{T}_1(\mathbf{x}), \dots, \mathcal{T}_d(\mathbf{x})\} \quad (9.60)$$

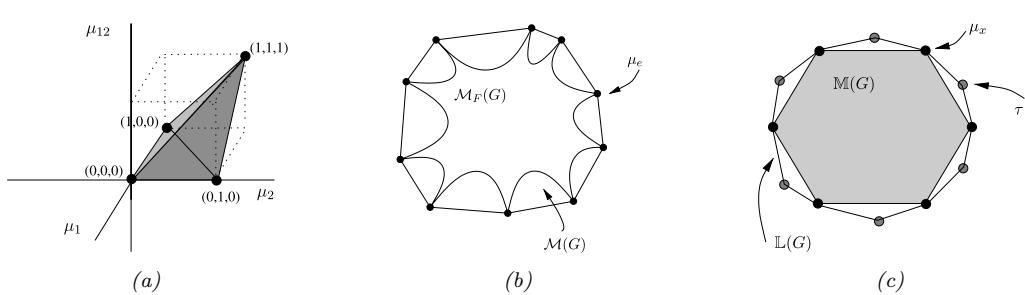


Figure 9.10: (a) Illustration of the marginal polytope for an Ising model with two variables. (b) Cartoon illustration of the set $\mathbb{M}_F(G)$, which is a nonconvex inner bound on the marginal polytope $\mathbb{M}(G)$. $\mathbb{M}_F(G)$ is used by mean field. (c) Cartoon illustration of the relationship between $\mathbb{M}(G)$ and $\mathbb{L}(G)$, which is used by loopy BP. The set $\mathbb{L}(G)$ is always an outer bound on $\mathbb{M}(G)$, and the inclusion $\mathbb{M}(G) \subset \mathbb{L}(G)$ is strict whenever G has loops. Both sets are polytopes, which can be defined as an intersection of half-planes (defined by facets), or as the convex hull of the vertices. $\mathbb{L}(G)$ actually has fewer facets than $\mathbb{M}(G)$, despite the picture. In fact, $\mathbb{L}(G)$ has $O(|\mathcal{X}||V| + |\mathcal{X}|^2|E|)$ facets, where $|\mathcal{X}|$ is the number of states per variable, $|V|$ is the number of variables, and $|E|$ is the number of edges. By contrast, $\mathbb{M}(G)$ has $O(|\mathcal{X}|^{|V|})$ facets. On the other hand, $\mathbb{L}(G)$ has more vertices than $\mathbb{M}(G)$, despite the picture, since $\mathbb{L}(G)$ contains all the binary vector extreme points $\boldsymbol{\mu} \in \mathbb{M}(G)$, plus additional fractional extreme points. From Figures 3.6, 5.4 and 4.2 of [Monster]. Used with kind permission of Martin Wainwright.

For example, for a 2 node MRF $X_1 - X_2$ with binary states, we have

$$\mathbb{M}(G) = \text{conv}\{(0, 0, 0), (1, 0, 0), (0, 1, 0), (1, 1, 1)\} \quad (9.61)$$

These are the four black dots in Figure 9.10(a). We see that the convex hull defines the same volume as the intersection of half-spaces.

9.3.3 Linear programming relaxation

We can write the MAP estimation problem as follows:

$$\max_{\mathbf{x} \in \mathcal{X}^V} \boldsymbol{\theta}^\top \mathcal{T}(\mathbf{x}) = \max_{\boldsymbol{\mu} \in \mathbb{M}(G)} \boldsymbol{\theta}^\top \boldsymbol{\mu} \quad (9.62)$$

To see why this equation is true, note that we can just set $\boldsymbol{\mu}$ to be a degenerate distribution with $\mu(x_s) = \mathbb{I}(x_s = x_s^*)$, where x_s^* is the optimal assignment of node s . Thus we can “emulate” the task of optimizing over discrete assignments by optimizing over probability distributions $\boldsymbol{\mu}$. Furthermore, the non-degenerate (“soft”) distributions will not correspond to corners of the polytope, and hence will not maximize a linear function.

It seems like we have an easy problem to solve, since the objective in Equation (9.62) is linear in $\boldsymbol{\mu}$, and the constraint set $\mathbb{M}(G)$ is convex. The trouble is, $\mathbb{M}(G)$ in general has a number of facets that is exponential in the number of nodes.

A standard strategy in combinatorial optimization is to relax the constraints. In this case, instead of requiring probability vector $\boldsymbol{\mu}$ to live in the marginal polytope $\mathbb{M}(G)$, we allow it to live inside a simpler, convex enclosing set $\mathbb{L}(G)$, which we define in Section 9.3.3.1. Thus we try to maximize the

1 following upper bound on the original objective:
2

$$\begin{aligned} \underline{3} \quad \tau^* &= \operatorname{argmax}_{\tau \in \mathbb{L}(G)} \theta^\top \tau \\ \underline{4} \end{aligned} \tag{9.63}$$

5 This is called a **linear programming relaxation** of the problem. If the solution τ^* is integral, it
6 corresponds to the exact MAP estimate; this will be the case when the graph is a tree. In general,
7 τ^* will be fractional; we can derive an approximate MAP estimate by rounding (see [Werner07] for
8 details).
9

10

11 9.3.3.1 A convex outer approximation to the marginal polytope

12 Consider a set of probability vectors τ that satisfy the following **local consistency** constraints:
13

$$\begin{aligned} \underline{14} \quad \sum_{x_s} \tau_s(x_s) &= 1 \\ \underline{15} \end{aligned} \tag{9.64}$$

$$\begin{aligned} \underline{16} \quad \sum_{x_t} \tau_{st}(x_s, x_t) &= \tau_s(x_s) \\ \underline{17} \end{aligned} \tag{9.65}$$

18 The first constraint is called the normalization constraint, and the second is called the marginalization
19 constraint. We then define the set
20

$$\begin{aligned} \underline{22} \quad \mathbb{L}(G) &\triangleq \{\tau \geq 0 : (\text{Equation (9.64)}) \text{ holds } \forall s \in \mathcal{V}, (\text{Equation (9.65)}) \text{ holds } \forall (s, t) \in \mathcal{E}\} \\ \underline{23} \end{aligned} \tag{9.66}$$

24 The set $\mathbb{L}(G)$ is also a polytope, but it only has $O(|V| + |E|)$ constraints. It is a convex **outer**
25 **approximation** on $\mathbb{M}(G)$, as shown in Figure 9.10(c). (By contrast, the mean field approximation,
26 which we discuss in Main Section 10.3, is a non-convex inner approximation, as we discuss in
27 Main Section 10.3.)

28 We call the terms $\tau_s, \tau_{st} \in \mathbb{L}(G)$ **pseudo marginals**, since they may not correspond to marginals
29 of any valid probability distribution. As an example of this, consider Figure 9.11(a). The picture
30 shows a set of pseudo node and edge marginals, which satisfy the local consistency requirements.
31 However, they are not globally consistent. To see why, note that τ_{12} implies $p(X_1 = X_2) = 0.8$, τ_{23}
32 implies $p(X_2 = X_3) = 0.8$, but τ_{13} implies $p(X_1 = X_3) = 0.2$, which is not possible (see [Monster]
33 for a formal proof). Indeed, Figure 9.11(b) shows that $\mathbb{L}(G)$ contains points that are not in $\mathbb{M}(G)$.
34 We claim that $\mathbb{M}(G) \subseteq \mathbb{L}(G)$, with equality iff G is a tree. To see this, first consider an element
35 $\mu \in \mathbb{M}(G)$. Any such vector must satisfy the normalization and marginalization constraints, hence
36 $\mathbb{M}(G) \subseteq \mathbb{L}(G)$.

37 Now consider the converse. Suppose T is a tree, and let $\mu \in \mathbb{L}(T)$. By definition, this satisfies the
38 normalization and marginalization constraints. However, any tree can be represented in the form
39

$$\begin{aligned} \underline{40} \quad p_\mu(x) &= \prod_{s \in V} \mu_s(x_s) \prod_{(s,t) \in E} \frac{\mu_{st}(x_s, x_t)}{\mu_s(x_s) \mu_t(x_t)} \\ \underline{41} \end{aligned} \tag{9.67}$$

42 Hence satisfying normalization and local consistency is enough to define a valid distribution for any
43 tree. Hence $\mu \in \mathbb{M}(T)$ as well.

44 In contrast, if the graph has loops, we have that $\mathbb{M}(G) \neq \mathbb{L}(G)$. See Figure 9.11(b) for an example
45 of this fact. The importance of this observation will become clear in Section 10.4.3.
46

47

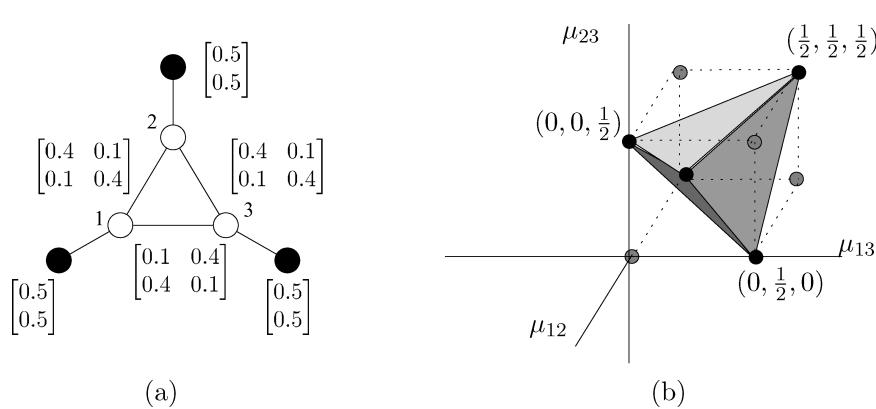


Figure 9.11: (a) Illustration of pairwise UGM on binary nodes, together with a set of pseudo marginals that are not globally consistent. (b) A slice of the marginal polytope illustrating the set of feasible edge marginals, assuming the node marginals are clamped at $\mu_1 = \mu_2 = \mu_3 = 0.5$. From Figure 4.1 of [Monster]. Used with kind permission of Martin Wainwright.

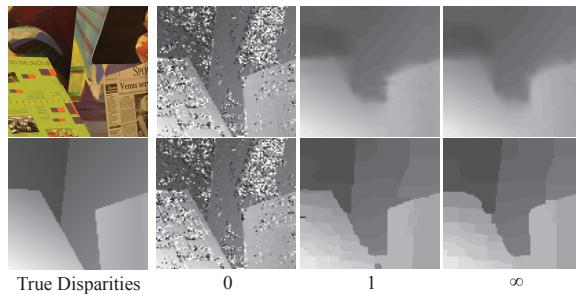


Figure 9.12: Illustration of belief propagation for stereo depth estimation applied to the Venus image from the Middlebury stereo benchmark dataset [Scharstein02]. Left column: image and true disparities. Remaining columns: initial estimate, estimate after 1 iteration, and estimate at convergence. Top row: Gaussian edge potentials using a continuous state space. Bottom row: robust edge potentials using a quantized state space. From Figure 4 of [Sudderth08bp]. Used with kind permission of Erik Sudderth.

9.3.3.2 Algorithms

Our task is to solve Equation (9.63), which requires maximizing a linear function over a simple convex polytope. For this, we could use a generic linear programming package. However, this is often very slow.

Fortunately, one can show that a simple algorithm, that sends messages between nodes in the graph, can be used to compute τ^* . In particular, the **tree reweighted belief propagation** algorithm can be used; see Section 10.4.5.3 for details.

1 **9.3.3.3 Application to stereo depth estimation**

3 Belief propagation is often applied to low-level computer vision problems (see e.g., [**Szeliski10**;
4 **Blake11**; **Prince12**]). For example, Figure 9.12 illustrates its application to the problem of **stereo**
5 **depth estimation** given a pair of monocular images (only one is shown). The value x_i is the
6 distance of pixel i from the camera (quantized to a certain number of values). The goal is to infer
7 these values from noisy measurements. We quantize the state space, rather than using a Gaussian
8 model, in order to avoid oversmoothing at discontinuities, which occur at object boundaries, as
9 illustrated in Figure 9.12. (We can also use a hybrid discrete-continuous state space, as discussed in
10 [**Yamaguchi2012**], but we can no longer apply BP.)

11 Not surprisingly, people have recently applied deep learning to this problem. For example,
12 [**Xu2019stereo**] describes a differentiable version of message passing (Main Section 9.4), which is
13 fast and can be trained end-to-end. However, it requires labeled data for training, i.e., pixel-wise
14 ground truth depth values. For this particular problem, such data can be collected from depth
15 cameras, but for other problems, BP on “unsupervised” MRFs may be needed.

16

17 **9.3.4 Graphcuts**

18 In this section, we show how to find MAP state estimates, or equivalently, minimum energy con-
19 figurations, by using the **maxflow** / **mincut** algorithm for graphs. This class of methods is
20 known as **graphcuts** and is very widely used, especially in computer vision applications (see e.g.,
21 [**Boykov2004**]).

22 We will start by considering the case of MRFs with binary nodes and a restricted class of potentials;
23 in this case, graphcuts will find the exact global optimum. We then consider the case of multiple
24 states per node; we can approximately solve this case by solving a series of binary subproblems, as
25 we will see.

26

27 **9.3.4.1 Graphcuts for the Ising model**

28 Let us start by considering a binary MRF where the edge energies have the following form:

$$\mathcal{E}_{uv}(x_u, x_v) = \begin{cases} 0 & \text{if } x_u = x_v \\ \lambda_{uv} & \text{if } x_u \neq x_v \end{cases} \quad (9.68)$$

29 where $\lambda_{st} \geq 0$ is the edge cost. This encourages neighboring nodes to have the same value (since we
30 are trying to minimize energy). Since we are free to add any constant we like to the overall energy
31 without affecting the MAP state estimate, let us rescale the local energy terms such that either
32 $\mathcal{E}_u(1) = 0$ or $\mathcal{E}_u(0) = 0$.

33 Now let us construct a graph which has the same set of nodes as the MRF, plus two distinguished
34 nodes: the source s and the sink t . If $\mathcal{E}_u(1) = 0$, we add the edge $x_u \rightarrow t$ with cost $\mathcal{E}_u(0)$. Similarly,
35 If $\mathcal{E}_u(0) = 0$, we add the edge $s \rightarrow x_u$ with cost $\mathcal{E}_u(1)$. Finally, for every pair of variables that are
36 connected in the MRF, we add edges $x_u \rightarrow x_v$ and $x_v \rightarrow x_u$, both with cost $\lambda_{u,v} \geq 0$. Figure 9.13
37 illustrates this construction for an MRF with 4 nodes and the following parameters:

$$\mathcal{E}_1(0) = 7, \mathcal{E}_2(1) = 2, \mathcal{E}_3(1) = 1, \mathcal{E}_4(1) = 6 \quad \lambda_{1,2} = 6, \lambda_{2,3} = 6, \lambda_{3,4} = 2, \lambda_{1,4} = 1 \quad (9.69)$$

40 Having constructed the graph, we compute a minimal $s - t$ cut. This is a partition of the nodes into
41 two sets, \mathcal{X}_s and \mathcal{X}_t , such that $s \in \mathcal{X}_s$ and $t \in \mathcal{X}_t$. We then find the partition which minimizes the
42

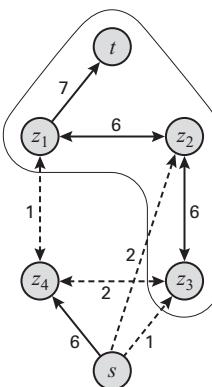


Figure 9.13: Illustration of graphcuts applied to an MRF with 4 nodes. Dashed lines are ones which contribute to the cost of the cut (for bidirected edges, we only count one of the costs). Here the min cut has cost 6. From Figure 13.5 from [KollerBook]. Used with kind permission of Daphne Koller.

sum of the cost of the edges between nodes on different sides of the partition:

$$\text{cost}(\mathcal{X}_s, \mathcal{X}_t) = \sum_{x_u \in \mathcal{X}_s, x_v \in \mathcal{X}_t} \text{cost}(x_u, x_v) \quad (9.70)$$

In Figure 9.13, we see that the min-cut has cost 6. Minimizing the cost in this graph is equivalent to minimizing the energy in the MRF. Hence nodes that are assigned to s have an optimal state of 0, and the nodes that are assigned to t have an optimal state of 1. In Figure 9.13, we see that the optimal MAP estimate is $(1, 1, 1, 0)$.

Thus we have converted the MAP estimation problem to a standard graph theory problem for which efficient solvers exist (see e.g., [CLR90]).

9.3.4.2 Graphcuts for binary MRFs with submodular potentials

We now discuss how to extend the graphcuts construction to binary MRFs with more general kinds of potential functions. In particular, suppose each pairwise energy satisfies the following condition:

$$\mathcal{E}_{uv}(1, 1) + \mathcal{E}_{uv}(0, 0) \leq \mathcal{E}_{uv}(1, 0) + \mathcal{E}_{uv}(0, 1) \quad (9.71)$$

In other words, the sum of the diagonal energies is less than the sum of the off-diagonal energies. In this case, we say the energies are submodular (Main Section 6.9). An example of a submodular energy is an Ising model where $\lambda_{uv} > 0$. This is also known as an **attractive MRF** or **associative MRF**, since the model “wants” neighboring states to be the same.

It is possible to modify the graph construction process for this setting, and then apply graphcuts, such that the resulting estimate is the global optimum [Greig89].

9.3.4.3 Graphcuts for nonbinary metric MRFs

We now discuss how to use graphcuts for approximate MAP estimation in MRFs where each node can have multiple states [Boykov01].

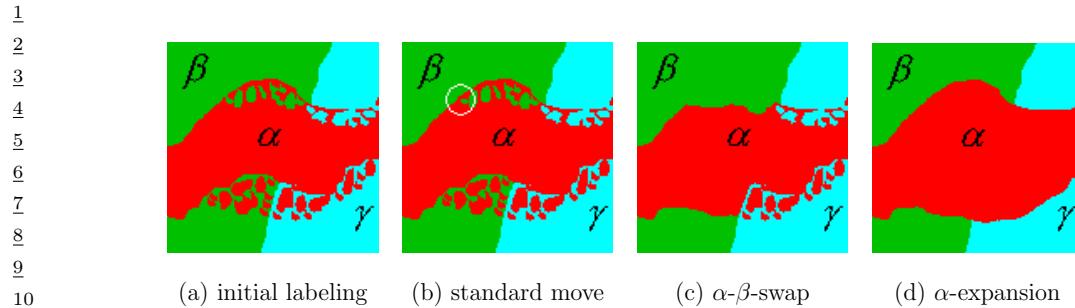


Figure 9.14: (a) An image with 3 labels. (b) A standard local move (e.g., by iterative conditional modes) just flips the label of one pixel. (c) An $\alpha - \beta$ swap allows all nodes that are currently labeled as α to be relabeled as β if this decreases the energy. (d) An α expansion allows all nodes that are not currently labeled as α to be relabeled as α if this decreases the energy. From Figure 2 of [Boykov01]. Used with kind permission of Ramin Zabih.

One approach is to use **alpha expansion**. At each step, it picks one of the available labels or states and calls it α ; then it solves a binary subproblem where each variable can choose to remain in its current state, or to become state α (see Figure 9.14(d) for an illustration).

Another approach is to use **alpha-beta swap**. At each step, two labels are chosen, call them α and β . All the nodes currently labeled α can change to β (and vice versa) if this reduces the energy (see Figure 9.14(c) for an illustration).

In order to solve these binary subproblems optimally, we need to ensure the potentials for these subproblems are submodular. This will be the case if the pairwise energies form a metric. We call such a model a **metric MRF**. For example, suppose the states have a natural ordering, as commonly arises if they are a discretization of an underlying continuous space. In this case, we can define a metric of the form $\mathcal{E}(x_s, x_t) = \min(\delta, \|x_s - x_t\|)$ or a semi-metric of the form $\mathcal{E}(x_s, x_t) = \min(\delta, (x_s - x_t)^2)$, for some constant $\delta > 0$. This energy encourages neighbors to have similar labels, but never “punishes” them by more than δ . (This δ term prevents over-smoothing, which we illustrate in Figure 9.12.)

9.3.4.4 Application to stereo depth estimation

Graphcuts is often applied to low-level computer vision problems, such as stereo depth estimation, which we discussed in Section 9.3.3.3. Figure 9.15 compares graphcuts (both swap and expansion version) to two other algorithms (simulated annealed, and a patch matching method based on normalization cross correlation) on the famous Tsukuba test image. The graphcuts approach works the best on this example, as well as others [Szeliski08; Tappen2003]. It also tends to outperform belief propagation (results not shown) in terms of speed and accuracy on stereo problems [Szeliski08; Tappen2003], as well as other problems such as CRF labeling of LIDAR point cloud data [Landrieu2017].

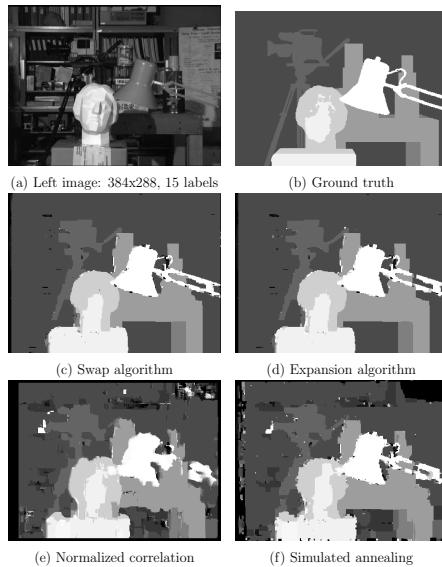


Figure 9.15: An example of stereo depth estimation using MAP estimation in a pairwise discrete MRF. (a) Left image, of size 384×288 pixels, from the University of Tsukuba. (The corresponding right image is similar, but not shown.) (b) Ground truth depth map, quantized to 15 levels. (c-f): MAP estimates using different methods: (c) $\alpha - \beta$ swap, (d) α expansion, (e) normalized cross correlation, (f) simulated annealing.

From Figure 10 of [Boykov01]. Used with kind permission of Ramin Zabih.

10 Variational inference

10.1 More Gaussian VI

In this section, we give more examples of Gaussian variational inference.

10.1.1 Example: Full-rank vs diagonal GVI on 1d linear regression

In this section, we give a comparison of HMC (Main Section 12.5) and Gaussian VI using both a mean field and full rank approximation. We use a simple example from [rethinking2].¹ Here the goal is to predict (log) GDP G of various countries (in the year 2000) as a function of two input variables: the ruggedness R of the country’s terrain, and whether the country is in Africa or not (A). Specifically, we use the following 1d regression model:

$$y_i \sim \mathcal{N}(\mu_i, \sigma^2) \quad (10.1)$$

$$\mu_i = \alpha + \beta^\top \mathbf{x}_i \quad (10.2)$$

$$\alpha \sim \mathcal{N}(0, 10) \quad (10.3)$$

$$\beta_j \sim \mathcal{N}(0, 1) \quad (10.4)$$

$$\sigma \sim \text{Unif}(0, 10) \quad (10.5)$$

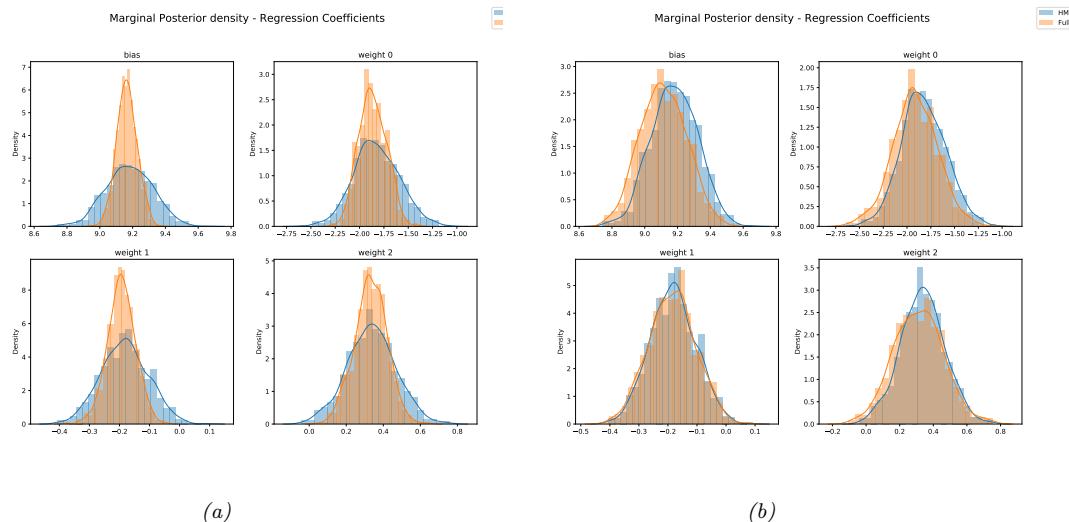
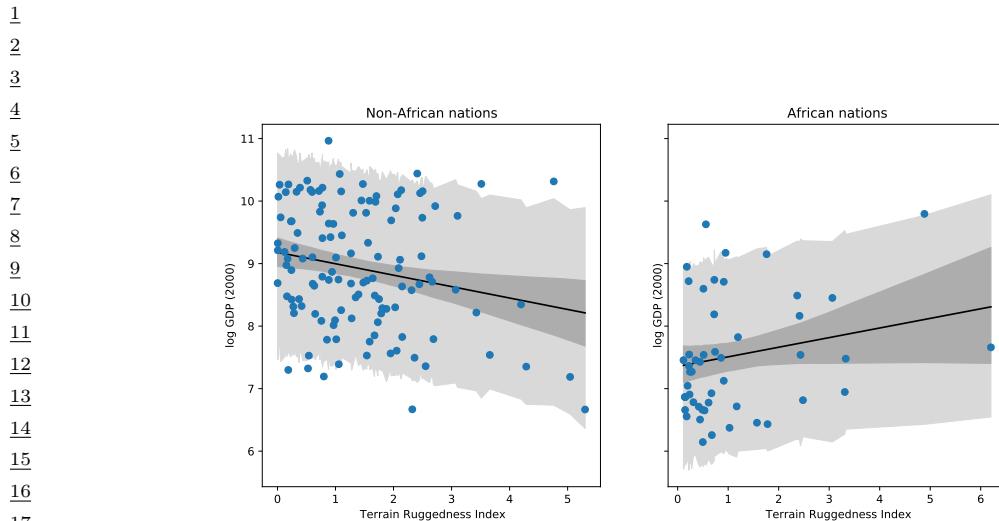
where $\mathbf{x}_i = (R_i, A_i, A_i \times R_i)$ are the features, and $y_i = G_i$ is the response. Note that this is almost a conjugate model, except for the non-conjugate prior on σ .

We first use HMC, which is often considered the “gold standard” of posterior inference. The resulting model fit is shown in Figure 10.1. This shows that GDP increases as a function of ruggedness for African countries, but decreases for non-African countries. (The reasons for this are unclear, but [Nunn2012] suggest that it is because more rugged Africa countries were less exploited by the slave trade, and hence are now wealthier.)

Now we consider a variational approximation to the posterior, of the form $p(\boldsymbol{\theta}|\mathcal{D}) \approx q(\boldsymbol{\theta}) = q(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. (Since the standard deviation σ must lie in the interval $[0, 10]$ due to the uniform prior, first transform it to the unconstrained value $\tau = \text{logit}(\sigma/10)$ before applying the Gaussian approximation, as explained in Main Section 10.2.2.)

Suppose we initially choose a diagonal Gaussian approximation. In Figure 10.2a, we compare the marginals of this posterior approximation (for the bias term and the 3 regression coefficients) with the “exact” posterior from HMC. We see that the variational marginals have roughly the same mean,

1. We choose this example since it is used as the introductory example in the [Pyro tutorial](#).



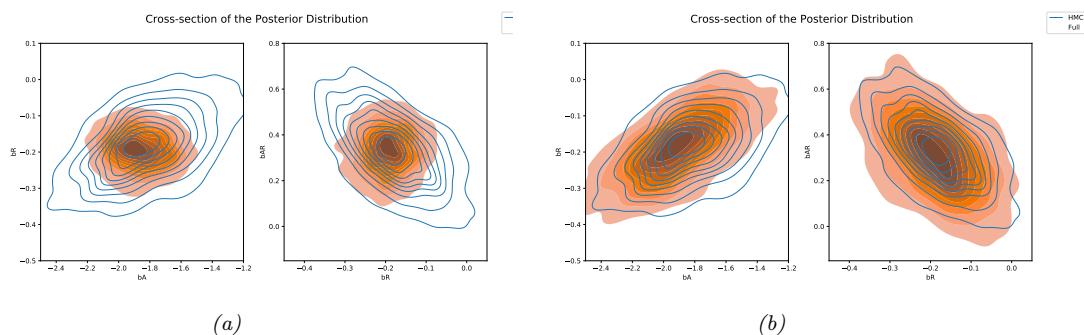


Figure 10.3: Joint posterior of pairs of variables for the linear model applied to the Africa data. (a) Blue is HMC, orange is Gaussian approximation with diagonal covariance. (b) Blue is HMC, orange is Gaussian approximation with full covariance. Generated by `linreg_bayes_svi_hmc.ipynb`.

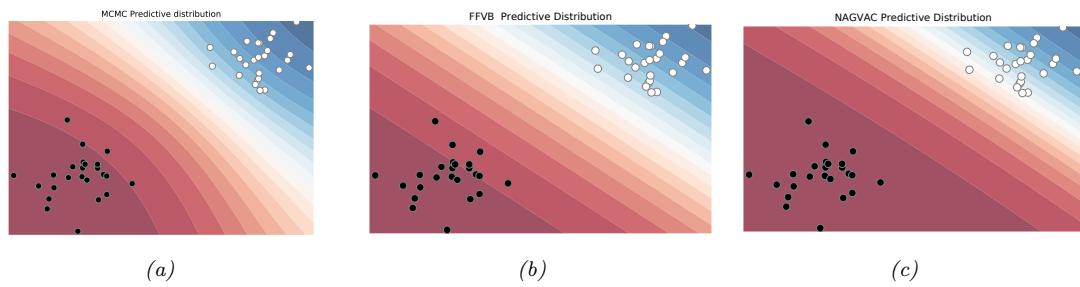


Figure 10.4: Bayesian inference applied to a 2d binary logistic regression problem, $p(y = 1|\mathbf{x}) = \sigma(w_0 + w_1x_1 + w_2x_2)$. We show the training data and the posterior predictive produced by different methods. (a) MCMC approximation. (b) VB approximation using full covariance matrix (Cholesky decomposition). (c) VB using rank 1 approximation. Generated by `vb_gauss_biclusters demo.ipynb`.

but their variances are too small, meaning they are overconfident. Furthermore, the variational approximation neglects any posterior correlations, as shown in Figure 10.3a.

We can improve the quality of the approximation by using a full covariance Gaussian. The resulting posterior marginals are shown in Figure 10.2b, and some bivariate posteriors are shown in Figure 10.3b. We see that the posterior approximation is now much more accurate.

Interestingly, both variational approximations give a similar predictive distribution to the HMC one in Figure 10.1. However, in some statistical problems we care about interpreting the parameters themselves (e.g., to assess the strength of the dependence on ruggedness), so a more accurate approximation is necessary to avoid reaching invalid conclusions.

10.1.2 Example: Full-rank vs rank-1 GVI for logistic regression

In this section, we compare full-rank GVI, rank-1 GVI and HMC on a simple 2d binary logistic regression problem. The results are shown in Figure 10.4. We see that the predictive distribution from the VI posterior is similar to that produced by MCMC.

1 2 **10.1.3 Structured (sparse) Gaussian VI**

3 In many problems, the target posterior can be represented in terms of a factor graph (see Main Sec-
4 tion 4.6.1). That is, we assume the negative log unnormalized joint probability (energy) can be
5 decomposed as follows:

6

$$\underline{8} \quad -\log p(\mathbf{z}, \mathcal{D}) = \phi(\mathbf{z}) = \sum_{c=1}^C \phi_c(\mathbf{z}_c) \quad (10.6)$$

9

10 where ϕ_c is the c 'th clique potential. Note that the same variables can occur in multiple potential
11 functions, but the dependency structure can be represented by a sparse graph G . Below we show
12 that the optimal Gaussian variational posterior $q(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1})$ will have a precision matrix $\boldsymbol{\Lambda}$
13 with the same sparsity structure as G . Furthermore, the natural gradient of the ELBO will also enjoy
14 the same sparsity structure. This allows us to use VI with the same accuracy as using a full-rank
15 covariance matrix but with efficiency closer to mean field (the cost per gradient step is potentially
16 only linear in the number of latent variables, depending on the treewidth of G).
17

18

19 20 **10.1.3.1 Sparsity of the ELBO**

21 To see why the optimal q is sparse, recall that the negative ELBO consists of two terms: the expected
22 energy, $-\mathbb{E}_{q(\mathbf{z})} [\log p(\mathbf{z}, \mathcal{D})]$, minus the entropy, $\mathbb{H}(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1}))$. That is,

23

$$\underline{25} \quad V(\boldsymbol{\psi}) = \mathbb{E}_{q(\mathbf{z}|\boldsymbol{\mu}, \boldsymbol{\Lambda})} [\phi(\mathbf{z})] + \frac{1}{2} \log(|\boldsymbol{\Lambda}|) \quad (10.7)$$

26

27 where $\boldsymbol{\psi} = (\boldsymbol{\mu}, \boldsymbol{\Lambda})$. To compute the first term, we only need the marginals $q(\mathbf{z}_c)$ for each clique, as
28 we see from Equation (10.6), so any dependencies with variables outside of \mathbf{z}_c are irrelevant. Thus
29 the optimal q will have the same sparsity structure as G , since this maximizes the entropy (no
30 unnecessary constraints). In other words, the optimal Gaussian q will be the following Gaussian
31 MRF (see Main Section 4.3.5):
32

33

$$\underline{35} \quad q(\mathbf{z}|\boldsymbol{\mu}, \boldsymbol{\Lambda}) \propto \prod_{c=1}^C \mathcal{N}_c(\mathbf{z}_c|\boldsymbol{\Lambda}_c \boldsymbol{\mu}_c, \boldsymbol{\Lambda}_c) \quad (10.8)$$

36

37 where $\mathcal{N}_c(\mathbf{x}|\mathbf{h}, \mathbf{K})$ is a Gaussian distribution in canonical (information) form with precision \mathbf{K} and
38 precision weighted mean \mathbf{h} . (For a more formal proof of this result, see e.g, [Barfoot2020sparse;
39 Courts2021]).
40

41

42 43 **10.1.3.2 Sparsity of the natural gradient of the ELBO**

44 In [Barfoot2020sparse], they show that the natural gradient of the ELBO also inherits the same
45 sparsity structure as G . In particular, at iteration i , they derive the following updates for the
46

1
2 variational parameters:

3
4 $\Lambda^{i+1} = \mathbb{E}_{q^i} \left[\frac{\partial^2}{\partial z^\top \partial z} \phi(z) \right] = \sum_{c=1}^C \mathbf{P}_c^\top \mathbb{E}_{q_c^i} \left[\frac{\partial^2}{\partial z_c^\top \partial z_c} \phi_c(z_c) \right] \mathbf{P}_c$ (10.9)

5
6 $\Lambda^{i+1} \delta^{i+1} = -\mathbb{E}_{q^i} \left[\frac{\partial}{\partial z^\top} \phi(z) \right] = -\sum_{c=1}^C \mathbb{E}_{q_c^i} \left[\frac{\partial}{\partial z_c^\top} \phi_c(z_c) \right]$ (10.10)

7
8 $\mu^{i+1} = \mu^i + \delta^{i+1}$ (10.11)

9
10 where \mathbf{P}_c is the projection matrix that extracts z_c from z , i.e. $z_c = \mathbf{P}_c z$. We can calculate the
11 gradient \mathbf{g}_c and Hessian \mathbf{H}_c of each factor c using automatic differentiation. We can then compute
12 $\delta^{i+1} = -(\Lambda^{i+1})^{-1} \mathbb{E}_{q^i} [\mathbf{g}]$ by solving a sparse linear system.

14 10.1.3.3 Computing posterior expectations

16 Finally, we discuss how to compute the expectations needed to evaluate the (gradient of the) ELBO.
17 (We drop the i superscript for brevity.) One approach, discussed in [Barfoot2020sparse], is to
18 use quadrature. This requires access to the marginals $q_c(z_c) = \mathcal{N}(z_c | \mu_c, \Sigma_c)$ for each factor. Note
19 that $\Sigma_c \neq (\Lambda_c)^{-1}$, so we cannot just invert each local block of the precision matrix. Instead we
20 must compute the covariance for the full joint, $\Sigma = \Lambda^{-1}$, and then extract the relevant blocks,
21 Σ_c . Fortunately, there are various methods, such as **Takahashi's algorithm** [Takahashi1973;
22 Barfoot2020fundamental] for efficiently computing the blocks Σ_c without first needing to compute
23 all of Σ . Alternatively, we can just use message passing on a Gaussian junction tree, as explained in
24 Main Section 2.3.3.

26 10.1.3.4 Gaussian VI for nonlinear least squares problems

28 We now consider the special case where the energy function can be written as a nonlinear least
29 squares objective:

30
31 $\phi(z) = \frac{1}{2} e(z)^\top \mathbf{W}^{-1} e(z) = \frac{1}{2} \sum_{c=1}^C e_c(z_c) \mathbf{W}_c^{-1} e_c(z_c)$ (10.12)

33
34 where $e(z) = [e_c(z_c)]_{c=1}^C$ is a vector of error terms, $z_c \in \mathbb{R}^{d_c}$, $e_c(z_c) \in \mathbb{R}^{n_c}$, $\mathbf{W} = \text{diag}(\mathbf{W}_1, \dots, \mathbf{W}_c)$,
35 and $\mathbf{W}_c \in \mathbb{R}^{n_c \times n_c}$. In this case, [Barfoot2020sparse] propose the following alternative objective
36 that is more conservative (entropic):

37
38 $V'(\psi) = \frac{1}{2} \mathbb{E}_q [e(z)^\top \mathbf{W}^{-1} e(z)] + \frac{1}{2} \log(|\Lambda|)$ (10.13)

39 This can be optimized using a Gauss-Newton method, that avoids the need to compute the Hessian
40 of each factor. Let us define the expected error vector at iteration i as

42
43 $\bar{e}^i = \mathbb{E}_{q^i} [e(z)] = [\mathbb{E}_{q_c^i} [e_c(z_c)]]_c = [\bar{e}_c^i]_c$ (10.14)

44 Similarly the expected Jacobian at iteration i is

45
46 $\bar{\mathbf{E}}^i = \mathbb{E}_{q^i} \left[\frac{\partial}{\partial z} e(z) \right] = \left[[\mathbb{E}_{q_c^i} \left[\frac{\partial}{\partial z_c} e_c(z_c) \right]]_c \right] = [\bar{\mathbf{J}}_c^i]_c$ (10.15)

1 where $\mathbf{J}_c^i \in \mathbb{R}^{n_c \times d_c}$ is the Jacobian matrix of $e_c(\mathbf{z}_c)$ wrt inputs z_{cj} for $j = 1 : d_c$. Then the updates
 2 are as follows:
 3

$$\frac{4}{5} \quad \boldsymbol{\Lambda}^{i+1} = (\bar{\mathbf{E}}^i)^\top \mathbf{W}^{-1} \bar{\mathbf{E}}^i = [(\bar{\mathbf{J}}_c^i)^\top \mathbf{W}_c^{-1} \bar{\mathbf{J}}_c^i]_c \quad (10.16)$$

$$\frac{6}{7} \quad \boldsymbol{\Lambda}^{i+1} \boldsymbol{\delta}^{i+1} = (\bar{\mathbf{E}}^i)^\top \mathbf{W}^{-1} \bar{\boldsymbol{\epsilon}}^i = [(\bar{\mathbf{J}}_c^i)^\top \mathbf{W}_c^{-1} \bar{\boldsymbol{\epsilon}}_c^i]_c \quad (10.17)$$

$$\frac{8}{9} \quad \boldsymbol{\mu}^{i+1} = \boldsymbol{\mu}^i + \boldsymbol{\delta}^{i+1} \quad (10.18)$$

10.2 Online variational inference

12 In this section, we discuss how to perform **online variational inference**. In particular, we discuss
 13 the **streaming variational Bayes (SVB)** approach of [Broderick2013] in which we, at step t ,
 14 we compute the new posterior using the previous posterior as the prior:

$$\frac{16}{17} \quad \boldsymbol{\psi}_t = \operatorname{argmin}_{\boldsymbol{\psi}} \underbrace{\mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\psi})} [\ell_t(\boldsymbol{\theta})] + D_{\text{KL}}(q(\boldsymbol{\theta}|\boldsymbol{\psi}) \| q(\boldsymbol{\theta}|\boldsymbol{\psi}_{t-1}))}_{-\mathcal{L}_t(\boldsymbol{\psi})} \quad (10.19)$$

$$\frac{18}{19} \quad = \operatorname{argmin}_{\boldsymbol{\psi}} \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\psi})} [\ell_t(\boldsymbol{\theta}) + \log q(\boldsymbol{\theta}|\boldsymbol{\psi}) - \log q(\boldsymbol{\theta}|\boldsymbol{\psi}_{t-1})] \quad (10.20)$$

21 where $\ell_t(\boldsymbol{\theta}) = -\log p(\mathcal{D}_t|\boldsymbol{\theta})$ is the negative log likelihood (or, more generally, some loss function)
 22 of the data batch at step t . This approach is also called **variational continual learning** or **VCL**
 23 [Nguyen2018]. (We discuss continual learning in Main Section 19.7.)

10.2.1 FOO-VB

27 In this section, we discuss a particular implementation of sequential VI called **FOO-VB**, which
 28 stands for “Fixed-point Operator for Online Variational Bayes” [Zeno2021]. This assumes Gaussian
 29 priors and posteriors. In particular, let

$$\frac{31}{32} \quad q(\boldsymbol{\theta}|\boldsymbol{\psi}_t) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma}), \quad q(\boldsymbol{\theta}|\boldsymbol{\psi}_{t-1}) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{m}, \mathbf{V}) \quad (10.21)$$

33 In this case, we can write the ELBO as follows:

$$\frac{35}{36} \quad \mathcal{L}_t(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{2} \left[\log \frac{\det(\mathbf{V})}{\det(\boldsymbol{\Sigma})} - D + \operatorname{tr}(\mathbf{V}^{-1} \boldsymbol{\Sigma}) + (\mathbf{m} - \boldsymbol{\mu})^\top \mathbf{V}^{-1} (\mathbf{m} - \boldsymbol{\mu}) \right] + \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})} [\ell_t(\boldsymbol{\theta})] \quad (10.22)$$

37 where D is the dimensionality of $\boldsymbol{\theta}$.

38 Let $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^\top$. We can compute the new variational parameters by solving the joint first order
 39 stationary conditions, $\nabla_{\boldsymbol{\mu}} \mathcal{L}_t(\boldsymbol{\mu}, \mathbf{L}) = \mathbf{0}$ and $\nabla_{\mathbf{L}} \mathcal{L}_t(\boldsymbol{\mu}, \mathbf{L}) = \mathbf{0}$. For the derivatives of the KL term, we
 40 use the identities

$$\frac{42}{43} \quad \frac{\partial \operatorname{tr}(\mathbf{V}^{-1} \boldsymbol{\Sigma})}{\partial L_{ij}} = 2 \sum_n V_{in}^{-1} L_{nj} \quad (10.23)$$

$$\frac{45}{46} \quad \frac{\partial \log |\det(\mathbf{L})|}{\partial L_{ij}} = L_{ij}^{-T} \quad (10.24)$$

For the derivatives of the expected loss, we use the the reparameterization trick, $\boldsymbol{\theta} = \boldsymbol{\mu} + \mathbf{L}\boldsymbol{\epsilon}$, and following identities:

$$\mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\mu}, \mathbf{L})} [\ell_t(\boldsymbol{\theta})] = \mathbb{E}_{\boldsymbol{\epsilon}} [\ell_t(\boldsymbol{\theta}(\boldsymbol{\epsilon}))] \quad (10.25)$$

$$\frac{\partial \mathbb{E}_{\boldsymbol{\epsilon}} [\ell_t(\boldsymbol{\theta})]}{\partial L_{ij}} = \mathbb{E}_{\boldsymbol{\epsilon}} \left[\frac{\partial \ell_t(\boldsymbol{\theta})}{\partial \theta_i} \epsilon_j \right] \quad (10.26)$$

Note that the expectation depends on the unknown variational parameters for q_t , so we get a fixed point equation which we need to iterate. As a faster alternative, [Zeno2018; Zeno2021] propose to evaluate the expectations using the variational parameters from the previous step, which then gives the new parameters in a single step, similar to EM.

We now derive the update equations. From $\nabla_{\boldsymbol{\mu}} \mathcal{L}_t(\boldsymbol{\mu}, \mathbf{L}) = \mathbf{0}$ we get

$$\mathbf{0} = -\mathbf{V}^{-1}(\mathbf{m} - \boldsymbol{\mu}) + \mathbb{E}_{\boldsymbol{\epsilon}} [\nabla \ell_t(\boldsymbol{\theta})] \quad (10.27)$$

$$\boldsymbol{\mu} = \mathbf{m} - \mathbf{V} \mathbb{E}_{\boldsymbol{\epsilon}} [\nabla \ell_t(\boldsymbol{\theta})] \quad (10.28)$$

From $\nabla_{\mathbf{L}} \mathcal{L}_t(\boldsymbol{\mu}, \mathbf{L}) = \mathbf{0}$. we get

$$0 = -(L^{-\top})_{ij} + \sum_n V_{i,n}^{-1} L_{n,j} + \mathbb{E}_{\boldsymbol{\epsilon}} \left[\frac{\partial \ell_t(\boldsymbol{\theta})}{\partial \theta_i} \epsilon_j \right] \quad (10.29)$$

In matrix form, we have

$$\mathbf{0} = -\mathbf{L}^{-\top} + \mathbf{V}^{-1}\mathbf{L} + \mathbb{E}_{\boldsymbol{\epsilon}} [\nabla \ell_t(\boldsymbol{\theta}) \boldsymbol{\epsilon}^{\top}] \quad (10.30)$$

Explicitly solving for \mathbf{L} in the case of a general (or low rank) matrix $\boldsymbol{\Sigma}$ is somewhat complicated; for the details, see [Zeno2021]. Fortunately, in the case of a diagonal appproximation, things simplify significantly, as we discuss in Section 10.2.2.

10.2.2 Bayesian gradient descent

In this section, we discuss a simplification of FOO-VB to the diagonal case. In [Zeno2018], they call the resulting algorithm “**Bayesian gradient descent**”, and they show it works well on some continual learning problems (see Main Section 19.7).

Let $\mathbf{V} = \text{diag}(v_i^2)$, $\boldsymbol{\Sigma} = \text{diag}(\sigma_i^2)$, so $\mathbf{L} = \text{diag}(\sigma_i)$. Also, let $g_i = \frac{\partial \ell_t(\boldsymbol{\theta})}{\partial \theta_i}$, which depends on $\boldsymbol{\epsilon}_i$. Then Equation (10.28) becomes

$$\mu_i = m_i - \eta v_i^2 \mathbb{E}_{\boldsymbol{\epsilon}_i} [g_i(\boldsymbol{\epsilon}_i)] \quad (10.31)$$

where we have included an explicit learning rate η to compensate for the fact that the fixed point equation update is approximate. For the variance terms, Equation (10.30) becomes

$$0 = -\frac{1}{\text{diag}(\sigma_i)} + \frac{\text{diag}(\sigma_i)}{\text{diag}(v_i^2)} + \mathbb{E}_{\boldsymbol{\epsilon}_i} [g_i \boldsymbol{\epsilon}_i] \quad (10.32)$$

This is a quadratic equation for each σ_i :

$$\frac{1}{v_i^2} \sigma_i^2 + \mathbb{E}_{\boldsymbol{\epsilon}_i} [g_i \boldsymbol{\epsilon}_i] \sigma_i - 1 = 0 \quad (10.33)$$

1 the solution of which is given by the following (since $\sigma_i > 0$):
2

$$\frac{1}{4} \sigma_i = \frac{-\mathbb{E}_{\epsilon_i} [g_i \epsilon_i] + \sqrt{(\mathbb{E}_{\epsilon_i} [g_i \epsilon_i])^2 + 4/v_i^2}}{2/v_i^2} = -\frac{1}{2} v_i^2 \mathbb{E}_{\epsilon_i} [g_i \epsilon_i] + \frac{1}{2} v_i^2 \sqrt{(\mathbb{E}_{\epsilon_i} [g_i \epsilon_i])^2 + 4/v_i^2} \quad (10.34)$$

$$\frac{6}{7} = -\frac{1}{2} v_i^2 \mathbb{E}_{\epsilon_i} [g_i \epsilon_i] + \sqrt{\frac{v_i^4}{4} ((\mathbb{E}_{\epsilon_i} [g_i \epsilon_i])^2 + 4/v_i^2)} = -\frac{1}{2} v_i^2 \mathbb{E}_{\epsilon_i} [g_i \epsilon_i] + v_i \sqrt{1 + (\frac{1}{2} v_i \mathbb{E}_{\epsilon_i} [g_i \epsilon_i])^2} \quad (10.35)$$

10 We can approximate the above expectations using K Monte Carlo samples. Thus the overall algorithm
11 is very similar to standard SGD, except we compute the gradient K times, and we update $\mu \in \mathbb{R}^D$
12 and $\sigma \in \mathbb{R}^D$ rather than $\theta \in \mathbb{R}^D$. See Algorithm 10.1 for the pseudocode, and see [Kurle2020] for
13 a related algorithm.

14

15 **Algorithm 10.1:** One step of Bayesian gradient descent

```

16 1 Function  $(\mu_t, \sigma_t, \mathcal{L}_t) = \text{BGD-update}(\mu_{t-1}, \sigma_{t-1}, \mathcal{D}_t; \eta, K)$ :
17 2 for  $k = 1 : K$  do
18 3   Sample  $\epsilon^k \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
19 4    $\theta^k = \mu_{t-1} + \sigma_{t-1} \odot \epsilon^k$ 
20 5    $\mathbf{g}^k = \nabla_{\theta} - \log p(\mathcal{D}_t | \theta)|_{\theta^k}$ 
22 6 for  $i = 1 : D$  do
23 7    $E_{1i} = \frac{1}{K} \sum_{k=1}^K g_i^k$ 
24 8    $E_{2i} = \frac{1}{K} \sum_{k=1}^K g_i^k \epsilon_i^k$ 
25 9    $\mu_{t,i} = \mu_{t-1,i} - \sigma_{t-1,i}^2 E_{1i}$ 
26 10   $\sigma_{t,i} = \sigma_{t-1,i} \sqrt{1 + (\frac{1}{2} \sigma_{t-1,i} E_{2i})^2} - \frac{1}{2} \sigma_{t-1,i}^2 E_{2i}$ 
28 11 for  $k = 1 : K$  do
29 12    $\theta^k = \mu_t + \sigma_t \odot \epsilon^k$ 
30 13    $\ell_t^k = -\log p(\mathcal{D}_t | \theta^k)$ 
32 14  $\mathcal{L}_t = - \left[ \frac{1}{K} \sum_{k=1}^K \ell_t^k + D_{\text{KL}} (\mathcal{N}(\mu_t, \sigma_t) \parallel \mathcal{N}(\mu_{t-1}, \sigma_{t-1})) \right]$ 


---



```

3435

36 10.3 Beyond mean field

37 In this sections, we discuss various improvements to VI that go beyond the mean field approximation.
39

40 10.3.1 Exploiting partial conjugacy

42 If the full conditionals of the joint model are conjugate distributions, we can use the VMP approach
43 of Main Section 10.3.7 to approximate the posterior one term at a time, similar to Gibbs sampling
44 (Main Section 12.3). However, in many models, some parts of the joint distribution are conjugate,
45 and some are non-conjugate. In [Khan2017aistats] they proposed the **conjugate-computation**
46 **variational inference** or **CVI** method to tackle models of this form. They exploit the partial
47

conjugacy to perform some updates in closed form, and perform the remaining updates using stochastic approximations.

To explain the method in more detail, let us assume the joint distribution has the form

$$p(\mathbf{y}, \mathbf{z}) \propto \tilde{p}_{nc}(\mathbf{y}, \mathbf{z}) \tilde{p}_c(\mathbf{y}, \mathbf{z}) \quad (10.36)$$

where \mathbf{z} are all the latents (global or local), \mathbf{y} are all the observables (data)², p_c is the conjugate part, p_{nc} is the non-conjugate part, and the tilde symbols indicate that these distributions may not be normalized wrt \mathbf{z} . More precisely, we assume the conjugate part is an exponential family model of the following form:

$$\tilde{p}_c(\mathbf{y}, \mathbf{z}) = h(\mathbf{z}) \exp[\mathcal{T}(\mathbf{z})^\top \boldsymbol{\eta} - A_c(\boldsymbol{\eta})] \quad (10.37)$$

where $\boldsymbol{\eta}$ is a *known* vector of natural parameters. (Any unknown model parameters should be included in the latent state \mathbf{z} , as we illustrate below.) We also assume that the variational posterior is an exponential family model with the same sufficient statistics, but different parameters:

$$q(\mathbf{z}|\boldsymbol{\lambda}) = h(\mathbf{z}) \exp[\mathcal{T}(\mathbf{z})^\top \boldsymbol{\eta} - A(\boldsymbol{\lambda})] \quad (10.38)$$

The mean parameters are given by $\boldsymbol{\mu} = \mathbb{E}_q [\mathcal{T}(\mathbf{z})]$. We assume the sufficient statistics are minimal, so that there is a unique 1:1 mapping between $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$: using

$$\boldsymbol{\mu} = \nabla_{\boldsymbol{\lambda}} A(\boldsymbol{\lambda}) \quad (10.39)$$

$$\boldsymbol{\lambda} = \nabla_{\boldsymbol{\mu}} A^*(\boldsymbol{\mu}) \quad (10.40)$$

where A^* is the conjugate of A (see Main Section 2.4.4). The ELBO is given by

$$\mathcal{L}(\boldsymbol{\lambda}) = \mathbb{E}_q [\log p(\mathbf{y}, \mathbf{z}) - \log q(\mathbf{z}|\boldsymbol{\lambda})] \quad (10.41)$$

$$\mathcal{L}(\boldsymbol{\mu}) = \mathcal{L}(\boldsymbol{\lambda}(\boldsymbol{\mu})) \quad (10.42)$$

The simplest way to fit this variational posterior is to perform SGD on the ELBO wrt the natural parameters:

$$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t + \eta_t \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}_t) \quad (10.43)$$

(Note the + sign in front of the gradient, since we are maximizing the ELBO.) The above gradient update is equivalent to solving the following optimization problem:

$$\boldsymbol{\lambda}_{t+1} = \underset{\boldsymbol{\lambda} \in \Omega}{\operatorname{argmin}} \underbrace{(\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}_t))^\top \boldsymbol{\lambda} - \frac{1}{2\eta_t} \|\boldsymbol{\lambda} - \boldsymbol{\lambda}_t\|_2^2}_{J(\boldsymbol{\lambda})} \quad (10.44)$$

where Ω is the space of valid natural parameters, and $\|\cdot\|_2$ is the Euclidean norm. To see this, note that the first order optimality conditions satisfy

$$\nabla_{\boldsymbol{\lambda}} J(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}_t) - \frac{1}{2\eta_t} (2\boldsymbol{\lambda} - 2\boldsymbol{\lambda}_t) = \mathbf{0} \quad (10.45)$$

² We denote observables by \mathbf{y} since the examples we consider later on are conditional models, where \mathbf{x} denote the inputs.

1 from which we get Equation (10.43).
 2

3 We can replace the Euclidean distance with a more general proximity function, such as the
 4 Bregman divergence between the distributions (see Main Section 5.1.10). This gives rise to the
 5 **mirror descent** algorithm (Section 6.1.4). We can also perform updates in the mean parameter
 6 space. In [Raskutti2015], they show that this is equivalent to performing natural gradient updates
 7 in the natural parameter space. Thus this method is sometimes called **natural gradient VI** or
 8 **NGVI**. Combining these two steps gives the following update equation:

$$9 \quad \underline{\mu}_{t+1} = \underset{\mu \in \mathcal{M}}{\operatorname{argmin}} (\nabla_{\mu} \bar{L}(\mu_t))^T \mu - \frac{1}{\eta_t} B_{A^*}(\mu || \mu_t) \quad (10.46)$$

12 where \mathcal{M} is the space of valid mean parameters and $\eta_t > 0$ is a stepsize.

13 In [Khan2017aistats], they show that the above update is equivalent to performing exact Bayesian
 14 inference in the following conjugate model:

$$16 \quad q(\mathbf{z} | \boldsymbol{\lambda}_{t+1}) \propto e^{\mathcal{T}(\mathbf{z})^T \tilde{\boldsymbol{\lambda}}_t} \tilde{p}_c(\mathbf{y}, \mathbf{z}) \quad (10.47)$$

17 We can think of the first term as an exponential family approximation to the non-conjugate part of
 18 the model, using local variational natural parameters $\tilde{\boldsymbol{\lambda}}_t$. (These are similar to the site parameters
 19 used in expectation propagation Main Section 10.7.) These can be computed using the following
 20 recursive update:
 21

$$22 \quad \tilde{\boldsymbol{\lambda}}_t = (1 - \eta_t) \tilde{\boldsymbol{\lambda}}_{t-1} + \eta_t \nabla_{\mu} \mathbb{E}_{q(\mathbf{z} | \boldsymbol{\mu}_t)} [\log \tilde{p}_{nc}(\mathbf{y}, \mathbf{z})] \quad (10.48)$$

24 where $\tilde{\boldsymbol{\lambda}}_0 = \mathbf{0}$ and $\tilde{\boldsymbol{\lambda}}_1 = \boldsymbol{\eta}$. (Details on how to compute this derivative are given in Main Section 6.4.5.)
 25 Once we can have “conjugated” the non-conjugate part, the natural parameter of the new variational
 26 posterior is obtained by
 27

$$28 \quad \boldsymbol{\lambda}_{t+1} = \tilde{\boldsymbol{\lambda}}_t + \boldsymbol{\eta} \quad (10.49)$$

29 This corresponds to a multiplicative update of the form
 30

$$31 \quad q_{t+1}(\mathbf{z}) \propto q_t(\mathbf{z})^{1-\eta_t} \left[\exp(\tilde{\boldsymbol{\lambda}}_t^T \mathcal{T}(\mathbf{z})) \right]^{\eta_t} \quad (10.50)$$

33 We give some examples of this below.
 34

35 10.3.1.1 Example: Gaussian process with non-Gaussian likelihoods

37 In Main Chapter 18, we discuss Gaussian processes, which are a popular model for non-parametric
 38 regression. Given a set of N inputs $\mathbf{x}_n \in \mathcal{X}$ and outputs $y_n \in \mathbb{R}$, we define the following joint
 39 Gaussian distribution:
 40

$$41 \quad p(\mathbf{y}_{1:N}, \mathbf{z}_{1:N} | \mathbf{X}) = \left[\prod_{n=1}^N \mathcal{N}(y_n | z_n, \sigma^2) \right] \mathcal{N}(\mathbf{z} | \mathbf{0}, \mathbf{K}) \quad (10.51)$$

44 where \mathbf{K} is the kernel matrix computed using $K_{ij} = \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$, and $z_n = f(\mathbf{x}_n)$ is the unknown
 45 function value for input n . Since this model is jointly Gaussian, we can easily compute the exact
 46 posterior $p(\mathbf{z} | \mathbf{y})$ in $O(N^3)$ time. (Faster approximations are also possible, see Main Section 18.5.)
 47

One challenge with GPs arises when the likelihood function $p(y_n|z_n)$ is non-Gaussian, as occurs with classification problems. To tackle this, we will use CVI. Since the conjugate part of the model is a Gaussian, we require that the variational approximation also be Gaussian, so we use $q(z|\lambda) = \mathcal{N}(z|\lambda^{(1)}, \lambda^{(2)})$.

Since the likelihood term factorizes across data points $n = 1 : N$, we will only need to compute marginals of this variational posterior. From Main Section 2.4.2.3 we know that the sufficient statistics and natural parameters of a univariate Gaussian are given by

(10.52)

$$\boldsymbol{\lambda}_n = \left[\frac{m_n}{v_n}, -\frac{1}{2v_n} \right] \quad (10.53)$$

The corresponding moment parameters are

$$\mu_n = [m_n, m_n^2 + v_n] \quad (10.54)$$

$$m_n = v_n \lambda_{\text{cl}}^{(1)} \quad (10.55)$$

$$v_n = \frac{1}{2\lambda_n^{(2)}} \quad (10.56)$$

We need to compute the gradient terms $\nabla_{\mu_n} \mathbb{E}_{\mathcal{N}(z_n | \mu_n)} [\log p(y_n | z_n)]$. We can do this by sampling z_n from the local Gaussian posterior, and then pushing gradients inside, using the results from Main Section 6.4.5.1. Let the resulting stochastic gradients at step t be $\hat{g}_{n,t}^{(1)}$ and $\hat{g}_{n,t}^{(2)}$. We can then update the likelihood approximation as follows:

$$\tilde{\lambda}_{n,t}^{(i)} = (1 - \eta_t) \tilde{\lambda}_{n,t-1}^{(i)} + \eta_t \hat{g}_{n,t}^{(i)} \quad (10.57)$$

We can also perform a “doubly stochastic” approximation (as in Main ??) by just updating a random subset of these terms. Once we have updated the likelihood, we can update the posterior using

$$q(\mathbf{z}|\boldsymbol{\lambda}_{t+1}) \propto \left[\prod_{n=1}^N e^{z_n \tilde{\lambda}_{n,t}^{(1)} + z_n^2 \tilde{\lambda}_{n,t}^{(2)}} \right] \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{K}) \quad (10.58)$$

$$\propto \left[\prod_{n=1}^N \mathcal{N}_c(z_n | \tilde{\lambda}_{n,t}^{(1)}, \tilde{\lambda}_{n,t}^{(2)}) \right] \mathcal{N}(z | \mathbf{0}, \mathbf{K}) \quad (10.59)$$

$$= \left[\prod_{n=1}^N \mathcal{N}(z_n | \tilde{m}_{n,t}, \tilde{v}_{n,t}) \right] \mathcal{N}(\mathbf{z} | \mathbf{0}, \mathbf{K}) \quad (10.60)$$

$$= \left[\prod_{n=1}^N \mathcal{N}(\tilde{m}_{n,t}|z_n, \tilde{v}_{n,t}) \right] \mathcal{N}(z|\mathbf{0}, \mathbf{K}) \quad (10.61)$$

where $\tilde{m}_{n,t}$ and $\tilde{v}_{n,t}$ are derived from $\tilde{\lambda}_{n,t}$. We can think of this as **Gaussianizing the likelihood** at each step, where we replace the observations y_n by **pseudo-observations** $\tilde{m}_{n,t}$ and use a variational variance $\tilde{v}_{n,t}$. This lets us use exact GP regression updates in the inner loop. See [Shi2019gp; Chang2020gp] for details.

1 **10.3.1.2 Example: Bayesian logistic regression**

3 In this section, we discuss how to compute a Gaussian approximation to $p(\mathbf{w}|\mathcal{D})$ for a binary logistic
4 regression model with a Gaussian prior on the weights. We will use CVI in which we “Gaussianize”
5 the likelihoods, and then perform closed form Bayesian linear regression in the inner loop. This is
6 similar to the approach used in Main Section 15.3.8, where we derive a quadratic lower bound to the
7 log likelihood. However, such “local VI” methods are not guaranteed to converge to a local maximum
8 of the ELBO [Khan12thesis], unlike the CVI method.

9 The joint distribution has the form

11

$$\underline{12} \quad p(\mathbf{y}_{1:N}, \mathbf{w}|\mathbf{X}) = \left[\prod_{n=1}^N p(y_n|z_n) \right] \mathcal{N}(\mathbf{w}|\mathbf{0}, \delta\mathbf{I}) \quad (10.62)$$

13

14 where $z_n = \mathbf{w}^\top \mathbf{x}_n$ is the local latent, and $\delta > 0$ is the prior variance (analogous to an ℓ_2 regularizer).
15 We compute the local Gaussian likelihood terms $\tilde{\lambda}_n$ as in in Section 10.3.1.1. We then have the
16 following variational joint:

18

$$\underline{19} \quad q(\mathbf{w}|\boldsymbol{\lambda}_{t+1}) \propto \left[\prod_{n=1}^N \mathcal{N}(\tilde{m}_{n,t}|\mathbf{w}^\top \mathbf{x}_n, \tilde{v}_{n,t}) \right] \mathcal{N}(\mathbf{w}|\mathbf{0}, \delta\mathbf{I}) \quad (10.63)$$

20

21 This corresponds to a Bayesian linear regression problem with pseudo-observations $\tilde{m}_{n,t}$ and variational
22 variance $\tilde{v}_{n,t}$.

24 **10.3.1.3 Example: Kalman smoothing with GLM likelihoods**

26 We can extend the above examples to perform posterior inference in a linear-Gaussian state-space
27 model (Main Section 29.6) with generalized linear model (GLM) likelihoods: we alternate between
28 Gaussianizing the likelihoods and running the Kalman smoother (Main Section 8.2.3).

30 **10.3.2 Structured mean for factorial HMMs**

32 Consider the factorial HMM model [Ghahramani97] introduced in Main Section 29.5.3. Suppose
33 there are M chains, each of length T , and suppose each hidden node has K states, as shown in
34 Figure 10.5(a). We will derive a structured mean field algorithm that takes $O(TM K^2 I)$ time, where
35 I is the number of mean field iterations (typically $I \sim 10$ suffices for good performance).

36 We can write the exact posterior in the following form:

37

$$\underline{38} \quad p(\mathbf{z}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp(-\mathcal{E}(\mathbf{z}, \mathbf{x})) \quad (10.64)$$

39

40

$$\underline{41} \quad \mathcal{E}(\mathbf{z}, \mathbf{x}) = \frac{1}{2} \sum_{t=1}^T \left(\mathbf{x}_t - \sum_m \mathbf{W}_m \mathbf{z}_{tm} \right)^\top \boldsymbol{\Sigma}^{-1} \left(\mathbf{x}_t - \sum_m \mathbf{W}_m \mathbf{z}_{tm} \right)$$

42

$$\underline{43} \quad - \sum_{m=1}^M \mathbf{z}_{1m}^\top \tilde{\boldsymbol{\pi}}_m - \sum_{t=2}^T \sum_{m=1}^M \mathbf{z}_{t-1,m}^\top \tilde{\mathbf{A}}_m \mathbf{z}_{t,m}$$

44

45

46 where $\tilde{\mathbf{A}}_m \triangleq \log \mathbf{A}_m$ and $\tilde{\boldsymbol{\pi}}_m \triangleq \log \boldsymbol{\pi}_m$, where the log is applied elementwise.

47

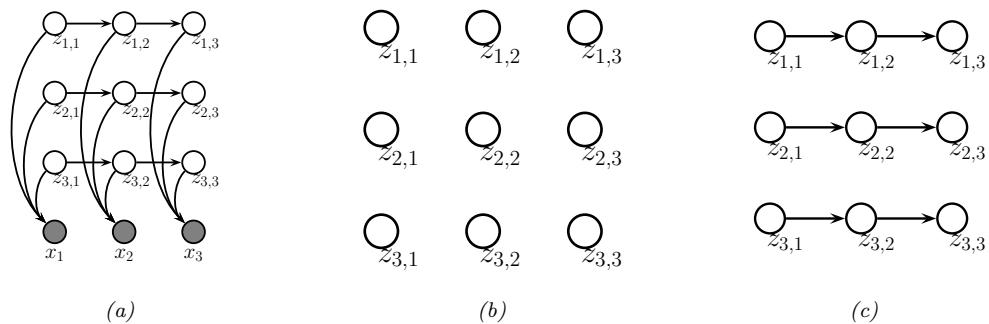


Figure 10.5: (a) A factorial HMM with 3 chains. (b) A fully factorized approximation. (c) A product-of-chains approximation. Adapted from Figure 2 of [Ghahramani97].

We can approximate the posterior as a product of marginals, as in Figure 10.5(b), but a better approximation is to use a product of chains, as in Figure 10.5(c). Each chain can be tractably updated individually, using the forwards-backwards algorithm (Main Section 9.2.3). More precisely, we assume

$$q(\mathbf{z}; \boldsymbol{\psi}) = \frac{1}{Z_q(\mathbf{x})} \prod_{m=1}^M q(z_{1m}; \boldsymbol{\psi}_{1m}) \prod_{t=2}^T q(z_{tm}|z_{t-1,m}; \boldsymbol{\psi}_{tm}) \quad (10.66)$$

$$q(z_{1m}; \boldsymbol{\psi}_{1m}) = \prod_{k=1}^K (\psi_{1mk} \pi_{mk})^{z_{1mk}} \quad (10.67)$$

$$q(z_{tm}|z_{t-1,m}; \boldsymbol{\psi}_{tm}) = \prod_{k=1}^K \left(\psi_{tmk} \prod_{j=1}^K (A_{mj} \psi_{tmj})^{z_{t-1,m,j}} \right)^{z_{tmk}} \quad (10.68)$$

Here the variational parameter ψ_{tmk} plays the role of an approximate local evidence, averaging out the effects of the other chains. This is in contrast to the exact local evidence, which couples all the chains together.

By separating out the approximate local evidence terms, we can rewrite the above as $q(\mathbf{z}) = \frac{1}{Z_q(\mathbf{x})} \exp(-\mathcal{E}_q(\mathbf{z}, \mathbf{x}))$, where

$$\mathcal{E}_q(\mathbf{z}, \mathbf{x}) = - \sum_{t=1}^T \sum_{m=1}^M \mathbf{z}_{tm}^\top \tilde{\boldsymbol{\psi}}_{tm} - \sum_{m=1}^M \mathbf{z}_{1m}^\top \tilde{\boldsymbol{\pi}}_m - \sum_{t=2}^T \sum_{m=1}^M \mathbf{z}_{t-1,m}^\top \tilde{\mathbf{A}}_m \mathbf{z}_{t,m} \quad (10.69)$$

where $\tilde{\boldsymbol{\psi}}_{tm} = \log \psi_{tm}$. We see that this has the same temporal factors as the exact log joint in Equation (10.65), but the local evidence terms are different: the dependence on the visible data \mathbf{x} has been replaced by dependence on ‘‘virtual data’’ $\boldsymbol{\psi}$.

The objective function is given by

$$D_{\text{KL}}(q \parallel \tilde{q}) = \mathbb{E}_q [\log q - \log \tilde{q}] \quad (10.70)$$

$$= -\mathbb{E}_q [\mathcal{E}_q(\mathbf{z}, \mathbf{x})] - \log Z_q(\mathbf{x}) + \mathbb{E}_q [\mathcal{E}(\mathbf{z}, \mathbf{x})] + \log Z(\mathbf{x}) \quad (10.71)$$

1 where $q = q(\mathbf{z}|\mathbf{x})$ and $\tilde{p} = p(\mathbf{z}|\mathbf{x})$. In [Ghahramani97] they show that we can optimize this using
 2 coordinate descent, where each update step is given by
 3

$$\frac{4}{5} \quad \psi_{tm} = \exp \left(\mathbf{W}_m^\top \boldsymbol{\Sigma}^{-1} \tilde{\mathbf{x}}_{tm} - \frac{1}{2} \boldsymbol{\delta}_m \right) \quad (10.72)$$

$$\frac{6}{7} \quad \boldsymbol{\delta}_m \triangleq \text{diag}(\mathbf{W}_m^\top \boldsymbol{\Sigma}^{-1} \mathbf{W}_m) \quad (10.73)$$

$$\frac{8}{9} \quad \tilde{\mathbf{x}}_{tm} \triangleq \mathbf{x}_t - \sum_{\ell \neq m}^M \mathbf{W}_\ell \mathbb{E}[\mathbf{z}_{t,\ell}] \quad (10.74)$$

11 The intuitive interpretation of $\tilde{\mathbf{x}}_{tm}$ is that it is the observation \mathbf{x}_t minus the predicted effect from
 12 all the other chains apart from m . This is then used to compute the approximate local evidence,
 13 ψ_{tm} . Having computed the variational local evidence terms for each chain, we can perform forwards-
 14 backwards in parallel, using these approximate local evidence terms to compute $q(\mathbf{z}_{t,m})$ for each m
 15 and t .

16 The update cost is $O(TM^2)$ for a full “sweep” over all the variational parameters, since we have
 17 to run forwards-backwards M times, for each chain independently. This is the same cost as a fully
 18 factorized approximation, but is much more accurate.

19

20 10.4 VI for graphical model inference

22 In this section, we discuss exact and approximate inference for discrete PGMs from a variational
 23 perspective, following [Monster].

24 Similar to Section 9.3, we will assume a pairwise MRF of the form

$$\frac{26}{27} \quad p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = \frac{1}{Z} \exp \left\{ \sum_{s \in \mathcal{V}} \theta_s(z_s) + \sum_{(s,t) \in \mathcal{E}} \theta_{st}(z_s, z_t) \right\} \quad (10.75)$$

29 We can write this as an exponential family model, $p(\mathbf{z}|\mathbf{x}) = \tilde{p}(\mathbf{z})/Z$, where $Z = \log p(\mathbf{x})$, $\tilde{p}(\mathbf{z}) =$
 30 $\mathcal{T}(\mathbf{z})^\top \boldsymbol{\theta}$, $\boldsymbol{\theta} = (\{\theta_{s;j}\}, \{\theta_{s,t;j,k}\})$ are all the node and edge parameters (the canonical parameters), and
 31 $\mathcal{T}(\mathbf{z}) = (\{\mathbb{I}(z_s = j)\}, \{\mathbb{I}(z_s = j, z_t = k)\})$ are all the node and edge indicator functions (the sufficient
 32 statistics). Note: we use $s, t \in \mathcal{V}$ to index nodes and $j, k \in \mathcal{X}$ to index states.

33

34 10.4.1 Exact inference as VI

36 We know that the ELBO is a lower bound on the log marginal likelihood:

$$\frac{37}{38} \quad \mathcal{L}(q) = \mathbb{E}_{q(\mathbf{z})} [\log \tilde{p}(\mathbf{z})] + \mathbb{H}(q) \leq \log Z \quad (10.76)$$

39 Let $\boldsymbol{\mu} = \mathbb{E}_q[\mathcal{T}(\mathbf{z})]$ be the mean parameters of the variational distribution. Then we can rewrite this
 40 as

$$\frac{41}{42} \quad \mathcal{L}(\boldsymbol{\mu}) = \boldsymbol{\theta}^\top \boldsymbol{\mu} + \mathbb{H}(\boldsymbol{\mu}) \leq \log Z \quad (10.77)$$

43 The set of all valid (unrestricted) mean parameters $\boldsymbol{\mu}$ is the **marginal polytope** corresponding to
 44 the graph, $\mathbb{M}(G)$, as explained in Section 9.3.2. Optimizing over this set recovers $q = p$, and hence

$$\frac{45}{46} \quad \max_{\boldsymbol{\mu} \in \mathbb{M}(G)} \boldsymbol{\theta}^\top \boldsymbol{\mu} + \mathbb{H}(\boldsymbol{\mu}) = \log Z \quad (10.78)$$

47

Method	Definition	Objective	Opt. Domain	Section
Exact	$\max_{\boldsymbol{\mu} \in \mathbb{M}(G)} \boldsymbol{\theta}^T \boldsymbol{\mu} + \mathbb{H}(\boldsymbol{\mu}) = \log Z$	Concave	Marginal polytope, convex	Section 10.4.1
Mean field	$\max_{\boldsymbol{\mu} \in \mathbb{M}_F(G)} \boldsymbol{\theta}^T \boldsymbol{\mu} + \mathbb{H}_{MF}(\boldsymbol{\mu}) \leq \log Z$	Concave	Nonconvex inner approx.	Section 10.4.2
Loopy BP	$\max_{\boldsymbol{\tau} \in \mathbb{L}(G)} \boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{Bethe}(\boldsymbol{\tau}) \approx \log Z$	Non-concave	Convex outer approx.	Section 10.4.3
TRBP	$\max_{\boldsymbol{\tau} \in \mathbb{L}(G)} \boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{TRBP}(\boldsymbol{\tau}) \geq \log Z$	Concave	Convex outer approx.	Section 10.4.5

Table 10.1: Summary of some variational inference methods for graphical models. TRBP is tree-reweighted belief propagation.

Equation (10.78) seems easy to optimize: the objective is concave, since it is the sum of a linear function and a concave function (see Main Figure 5.4 to see why entropy is concave); furthermore, we are maximizing this over a convex set, $\mathbb{M}(G)$. Hence there is a unique global optimum. However, the entropy is typically intractable to compute, since it requires summing over all states. We discuss approximations below. See Table 10.1 for a high level summary of the methods we discuss.

10.4.2 Mean field VI

The mean field approximation to the entropy is simply

$$\mathbb{H}_{MF}(\boldsymbol{\mu}) = \sum_s \mathbb{H}(\boldsymbol{\mu}_s) \quad (10.79)$$

which follows from the factorization assumption. Thus the mean field objective is

$$\mathcal{L}_{MF}(\boldsymbol{\mu}) = \boldsymbol{\theta}^T \boldsymbol{\mu} + \mathbb{H}_{MF}(\boldsymbol{\mu}) \leq \log Z \quad (10.80)$$

This is a concave lower bound on $\log Z$. We will maximize this over a simpler, but non-convex, inner approximation to $\mathbb{M}(G)$, as we now show.

First, let F be an edge subgraph of the original graph G , and let $\mathcal{I}(F) \subseteq \mathcal{I}$ be the subset of sufficient statistics associated with the cliques of F . Let Ω be the set of canonical parameters for the full model, and define the canonical parameter space for the submodel as follows:

$$\Omega(F) \triangleq \{\boldsymbol{\theta} \in \Omega : \boldsymbol{\theta}_\alpha = 0 \forall \alpha \in \mathcal{I} \setminus \mathcal{I}(F)\} \quad (10.81)$$

In other words, we require that the natural parameters associated with the sufficient statistics α outside of our chosen class to be zero. For example, in the case of a fully factorized approximation, F_0 , we remove all edges from the graph, giving

$$\Omega(F_0) \triangleq \{\boldsymbol{\theta} \in \Omega : \boldsymbol{\theta}_{st} = 0 \forall (s, t) \in E\} \quad (10.82)$$

In the case of structured mean field (Main Section 10.4.1), we set $\theta_{st} = 0$ for edges which are not in our tractable subgraph.

Next, we define the mean parameter space of the restricted model as follows:

$$\mathbb{M}_F(G) \triangleq \{\boldsymbol{\mu} \in \mathbb{R}^d : \boldsymbol{\mu} = \mathbb{E}_{\boldsymbol{\theta}} [\mathcal{T}(\boldsymbol{z})] \text{ for some } \boldsymbol{\theta} \in \Omega(F)\} \quad (10.83)$$

This is called an **inner approximation** to the marginal polytope, since $\mathbb{M}_F(G) \subseteq \mathbb{M}(G)$. See Figure 9.10(b) for a sketch. Note that $\mathbb{M}_F(G)$ is a non-convex polytope, which results in multiple local optima.

1 Thus the mean field problem becomes
 2

$$3 \quad 4 \quad \max_{\boldsymbol{\mu} \in \mathbb{M}_F(G)} \boldsymbol{\theta}^\top \boldsymbol{\mu} + \mathbb{H}_{\text{MF}}(\boldsymbol{\mu}) \quad (10.84)$$

5

6 This requires maximizing a concave objective over a non-convex set. It is typically optimized using
 7 coordinate ascent, since it is easy to optimize a scalar concave function over the marginal distribution
 8 for each node.
 9

10 11 10.4.3 Loopy belief propagation as VI

12 Recall from Section 10.4.1 that exact inference can be posed as solving the following optimization
 13 problem: $\max_{\boldsymbol{\mu} \in \mathbb{M}(G)} \boldsymbol{\theta}^\top \boldsymbol{\mu} + \mathbb{H}(\boldsymbol{\mu})$, where $\mathbb{M}(G)$ is the marginal polytope corresponding to the graph
 14 (see Section 9.3.2 for details). Since this set has exponentially many facets, it is intractable to
 15 optimize over.

16 In Section 10.4.2, we discussed the mean field approximation, which uses a nonconvex inner
 17 approximation, $\mathbb{M}_F(G)$, obtained by dropping some edges from the graphical model, thus enforcing
 18 a factorization of the posterior. We also approximated the entropy by using the entropy of each
 19 marginal.

20 In this section, we will consider a convex outer approximation, $\mathbb{L}(G)$, based on pseudo marginals,
 21 as in Section 9.3.3.1. We also need to approximate the entropy (which was not needed when
 22 performing MAP estimation, discussed in Section 9.3.3). We discuss this entropy approximation in
 23 Section 10.4.3.1, and then show how we can use this to approximate $\log Z$. Finally we show that
 24 loopy belief propagation attempts to optimize this approximation.
 25

26 27 10.4.3.1 Bethe free energy

28 From Equation (9.67), we know that a joint distribution over a tree-structured graphical model can
 29 be represented exactly by the following:
 30

$$31 \quad 32 \quad p_{\boldsymbol{\mu}}(\mathbf{x}) = \prod_{s \in V} \mu_s(x_s) \prod_{(s,t) \in E} \frac{\mu_{st}(x_s, x_t)}{\mu_s(x_s) \mu_t(x_t)} \quad (10.85)$$

33

34 This satisfies the normalization and pairwise marginalization constraints of the outer approximation
 35 by construction.
 36

37 From Equation 10.85, we can write the exact entropy of any tree structured distribution $\boldsymbol{\mu} \in \mathbb{M}(T)$
 38 as follows:

$$39 \quad 40 \quad \mathbb{H}(\boldsymbol{\mu}) = \sum_{s \in V} H_s(\mu_s) - \sum_{(s,t) \in E} I_{st}(\mu_{st}) \quad (10.86)$$

41

$$42 \quad 43 \quad H_s(\mu_s) = - \sum_{x_s \in \mathcal{X}_s} \mu_s(x_s) \log \mu_s(x_s) \quad (10.87)$$

44

$$45 \quad 46 \quad I_{st}(\mu_{st}) = \sum_{(x_s, x_t) \in \mathcal{X}_s \times \mathcal{X}_t} \mu_{st}(x_s, x_t) \log \frac{\mu_{st}(x_s, x_t)}{\mu_s(x_s) \mu_t(x_t)} \quad (10.88)$$

47

Note that we can rewrite the mutual information term in the form $I_{st}(\mu_{st}) = H_s(\mu_s) + H_t(\mu_t) - H_{st}(\mu_{st})$, and hence we get the following alternative but equivalent expression:

$$\mathbb{H}(\boldsymbol{\mu}) = - \sum_{s \in V} (d_s - 1) H_s(\mu_s) + \sum_{(s,t) \in E} H_{st}(\mu_{st}) \quad (10.89)$$

where d_s is the degree (number of neighbors) for node s .

The **Bethe**³ approximation to the entropy is simply the use of Equation 10.86 even when we don't have a tree:

$$\mathbb{H}_{\text{Bethe}}(\boldsymbol{\tau}) = \sum_{s \in V} H_s(\tau_s) - \sum_{(s,t) \in E} I_{st}(\tau_{st}) \quad (10.90)$$

We define the **Bethe free energy** as the expected energy minus approximate entropy:

$$\mathcal{F}_{\text{Bethe}}(\boldsymbol{\tau}) \triangleq -[\boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{\text{Bethe}}(\boldsymbol{\tau})] \approx -\log Z \quad (10.91)$$

Thus our final objective becomes

$$\max_{\boldsymbol{\tau} \in \mathbb{L}(G)} \boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{\text{Bethe}}(\boldsymbol{\tau}) \quad (10.92)$$

We call this the **Bethe variational problem** or BVP. The space we are optimizing over is a convex set, but the objective itself is not concave (since $\mathbb{H}_{\text{Bethe}}$ is not concave). Thus there can be multiple local optima. Also, the entropy approximation is not a bound (either upper or lower) on the true entropy. Thus the value obtained by the BVP is just an approximation to $\log Z(\boldsymbol{\theta})$. However, in the case of trees, the approximation is exact. Also, in the case of models with attractive potentials, the resulting value turns out to be an upper bound [Sudderth08]. In Section 10.4.5, we discuss how to modify the algorithm so it always minimizes an upper bound for any model.

10.4.3.2 LBP messages are Lagrange multipliers

In this subsection, we will show that any fixed point of the LBP algorithm defines a stationary point of the above constrained objective. Let us define the normalization constraint as $C_{ss}(\boldsymbol{\tau}) \triangleq -1 + \sum_{x_s} \tau_s(x_s)$, and the marginalization constraint as $C_{ts}(x_s; \boldsymbol{\tau}) \triangleq \tau_s(x_s) - \sum_{x_t} \tau_{st}(x_s, x_t)$ for each edge $t \rightarrow s$. We can now write the Lagrangian as

$$\begin{aligned} \mathcal{L}(\boldsymbol{\tau}, \boldsymbol{\lambda}; \boldsymbol{\theta}) &\triangleq \boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{\text{Bethe}}(\boldsymbol{\tau}) + \sum_s \lambda_{ss} C_{ss}(\boldsymbol{\tau}) \\ &\quad + \sum_{s,t} \left[\sum_{x_s} \lambda_{ts}(x_s) C_{ts}(x_s; \boldsymbol{\tau}) + \sum_{x_t} \lambda_{st}(x_t) C_{st}(x_t; \boldsymbol{\tau}) \right] \end{aligned} \quad (10.93)$$

(The constraint that $\boldsymbol{\tau} \geq 0$ is not explicitly enforced, but one can show that it will hold at the optimum since $\boldsymbol{\theta} > 0$.) Some simple algebra then shows that $\nabla_{\boldsymbol{\tau}} \mathcal{L} = \mathbf{0}$ yields

$$\log \tau_s(x_s) = \lambda_{ss} + \theta_s(x_s) + \sum_{t \in \text{nbr}(s)} \lambda_{ts}(x_s) \quad (10.94)$$

$$\log \frac{\tau_{st}(x_s, x_t)}{\tilde{\tau}_s(x_s) \tilde{\tau}_t(x_t)} = \theta_{st}(x_s, x_t) - \lambda_{ts}(x_s) - \lambda_{st}(x_t) \quad (10.95)$$

³ Hans Bethe was a German-American physicist, 1906–2005.

1 where we have defined $\tilde{\tau}_s(x_s) \triangleq \sum_{x_t} \tau(x_s, x_t)$. Using the fact that the marginalization constraint
 2 implies $\tilde{\tau}_s(x_s) = \tau_s(x_s)$, we get
 3

$$\begin{aligned} 4 \quad \log \tau_{st}(x_s, x_t) &= \lambda_{ss} + \lambda_{tt} + \theta_{st}(x_s, x_t) + \theta_s(x_s) + \theta_t(x_t) \\ 5 \quad &+ \sum_{u \in \text{nbr}(s) \setminus t} \lambda_{us}(x_s) + \sum_{u \in \text{nbr}(t) \setminus s} \lambda_{ut}(x_t) \end{aligned} \quad (10.96)$$

6 To make the connection to message passing, define $m_{t \rightarrow s}(x_s) = \exp(\lambda_{ts}(x_s))$. With this notation, we
 7 can rewrite the above equations (after taking exponents of both sides) as follows:
 8

$$9 \quad \tau_s(x_s) \propto \exp(\theta_s(x_s)) \prod_{t \in \text{nbr}(s)} m_{t \rightarrow s}(x_s) \quad (10.97)$$

$$\begin{aligned} 10 \quad \tau_{st}(x_s, x_t) &\propto \exp(\theta_{st}(x_s, x_t) + \theta_s(x_s) + \theta_t(x_t)) \\ 11 \quad &\times \prod_{u \in \text{nbr}(s) \setminus t} m_{u \rightarrow s}(x_s) \prod_{u \in \text{nbr}(t) \setminus s} m_{u \rightarrow t}(x_t) \end{aligned} \quad (10.98)$$

12 where the λ terms and irrelevant constants are absorbed into the constant of proportionality. We see
 13 that this is equivalent to the usual expression for the node and edge marginals in LBP.
 14

15 To derive an equation for the messages in terms of other messages (rather than in terms of λ_{ts}),
 16 we enforce the marginalization condition $\sum_{x_t} \tau_{st}(x_s, x_t) = \tau_s(x_s)$. Then one can show that
 17

$$\begin{aligned} 18 \quad m_{t \rightarrow s}(x_s) &\propto \sum_{x_t} \left[\exp\{\theta_{st}(x_s, x_t) + \theta_t(x_t)\} \prod_{u \in \text{nbr}(t) \setminus s} m_{u \rightarrow t}(x_t) \right] \end{aligned} \quad (10.99)$$

19 We see that this is equivalent to the usual expression for the messages in LBP.
 20

21 10.4.3.3 Kikuchi free energy

22 We have shown that LBP minimizes the Bethe free energy. In this section, we show that generalized
 23 BP (Main Section 9.4.6) minimizes the **Kikuchi free energy**; we define this below, but the key
 24 idea is that it is a tighter approximation to $\log Z$.

25 In more detail, define $\mathbb{L}_t(G)$ to be the set of all pseudo-marginals such that normalization and
 26 marginalization constraints hold on a hyper-graph whose largest hyper-edge is of size $t + 1$. For
 27 example, in Main Figure 9.14, we impose constraints of the form

$$\begin{aligned} 28 \quad \sum_{x_1, x_2} \tau_{1245}(x_1, x_2, x_4, x_5) &= \tau_{45}(x_4, x_5), \quad \sum_{x_6} \tau_{56}(x_5, x_6) = \tau_5(x_5), \dots \end{aligned} \quad (10.100)$$

29 Furthermore, we approximate the entropy as follows:
 30

$$\begin{aligned} 31 \quad \mathbb{H}_{\text{Kikuchi}}(\boldsymbol{\tau}) &\triangleq \sum_{g \in E} c(g) H_g(\tau_g) \end{aligned} \quad (10.101)$$

32 where $H_g(\tau_g)$ is the entropy of the joint (pseudo) distribution on the vertices in set g , and $c(g)$ is called
 33 the **overcounting number** of set g . These are related to **Mobius numbers** in set theory. Rather
 34

than giving a precise definition, we just give a simple example. For the graph in Main Figure 9.14, we have

$$\mathbb{H}_{\text{Kikuchi}}(\boldsymbol{\tau}) = -[H_{1245} + H_{2356} + H_{4578} + H_{5689}] - [H_{25} + H_{45} + H_{56} + H_{58}] + H_5 \quad (10.102)$$

Putting these two approximations together, we can define the **Kikuchi free energy**⁴ as follows:

$$\mathcal{F}_{\text{Kikuchi}}(\boldsymbol{\tau}) \triangleq -[\boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{\text{Kikuchi}}(\boldsymbol{\tau})] \approx -\log Z \quad (10.103)$$

Our variational problem becomes

$$\max_{\boldsymbol{\tau} \in \mathbb{L}(G)} \boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{\text{Kikuchi}}(\boldsymbol{\tau}) \quad (10.104)$$

Just as with the Bethe free energy, this is not a concave objective. There are several possible algorithms for finding a local optimum of this objective, including generalized belief propagation. For details, see e.g., [Monster] or [KollerBook].

10.4.4 Convex belief propagation

The mean field energy functional is concave, but it is maximized over a non-convex inner approximation to the marginal polytope. The Bethe and Kikuchi energy functionals are not concave, but they are maximized over a convex outer approximation to the marginal polytope. Consequently, for both MF and LBP, the optimization problem has multiple optima, so the methods are sensitive to the initial conditions. Given that the exact formulation Equation (10.78) is a concave objective maximized over a convex set, it is natural to try to come up with an approximation of a similar form, without local optima.

Convex belief propagation involves working with a set of tractable submodels, \mathcal{F} , such as trees or planar graphs. For each model $F \subset G$, the entropy is higher, $\mathbb{H}(\boldsymbol{\mu}(F)) \geq \mathbb{H}(\boldsymbol{\mu}(G))$, since F has fewer constraints. Consequently, any convex combination of such subgraphs will have higher entropy, too:

$$\mathbb{H}(\boldsymbol{\mu}(G)) \leq \sum_{F \in \mathcal{F}} \rho(F) \mathbb{H}(\boldsymbol{\mu}(F)) \triangleq \mathbb{H}(\boldsymbol{\mu}, \rho) \quad (10.105)$$

where $\rho(F) \geq 0$ and $\sum_F \rho(F) = 1$. Furthermore, $\mathbb{H}(\boldsymbol{\mu}, \rho)$ is a concave function of $\boldsymbol{\mu}$.

Having defined an upper bound on the entropy, we now consider a convex outerbound on the marginal polytope of mean parameters. We want to ensure we can evaluate the entropy of any vector $\boldsymbol{\tau}$ in this set, so we restrict it so that the projection of $\boldsymbol{\tau}$ onto the subgraph G lives in the projection of \mathbb{M} onto F :

$$\mathbb{L}(G; \mathcal{F}) \triangleq \{\boldsymbol{\tau} \in \mathbb{R}^d : \boldsymbol{\tau}(F) \in \mathbb{M}(F) \forall F \in \mathcal{F}\} \quad (10.106)$$

This is a convex set since each $\mathbb{M}(F)$ is a projection of a convex set. Hence we define our problem as

$$\max_{\boldsymbol{\tau} \in \mathbb{L}(G; \mathcal{F})} \boldsymbol{\tau}^T \boldsymbol{\theta} + \mathbb{H}(\boldsymbol{\tau}, \rho) \quad (10.107)$$

⁴ Ryoichi Kikuchi is a Japanese physicist.

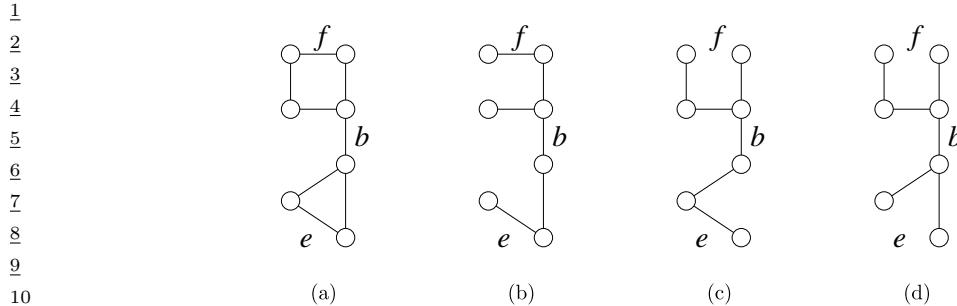


Figure 10.6: (a) A graph. (b-d) Some of its spanning trees. From Figure 7.1 of [Monster]. Used with kind permission of Martin Wainwright.

This is a concave objective being maximized over a convex set, and hence has a unique optimum. Furthermore, the result is always an upper bound on $\log Z$, because the entropy is an upper bound, and we are optimizing over a larger set than the marginal polytope.

It remains to specify the set of tractable submodels, \mathcal{F} , and the distribution ρ . We discuss some options below.

10.4.5 Tree-reweighted belief propagation

In this section, we discuss **tree reweighted BP** [Wainwright05map; Kolmogorov06], which is a form of convex BP which uses spanning trees as the set of tractable models \mathcal{F} , as we describe below.

10.4.5.1 Spanning tree polytope

It remains to specify the set of tractable submodels, \mathcal{F} , and the distribution ρ . We will consider the case where \mathcal{F} is all spanning trees of a graph. For any given tree, the entropy is given by Equation 10.86. To compute the upper bound, obtained by averaging over all trees, note that the terms $\sum_F \rho(F) H(\mu(F)_s)$ for single nodes will just be H_s , since node s appears in every tree, and $\sum_F \rho(F) = 1$. But the mutual information term I_{st} receives weight $\rho_{st} = \mathbb{E}_\rho [\mathbb{I}((s,t) \in E(T))]$, known as the **edge appearance probability**. Hence we have the following upper bound on the entropy:

$$\mathbb{H}(\boldsymbol{\mu}) \leq \sum_{s \in V} H_s(\mu_s) - \sum_{(s,t) \in E} \rho_{st} I_{st}(\mu_{st}) \triangleq \mathbb{H}_{\text{TRBP}}(\boldsymbol{\mu}) \quad (10.108)$$

This is called the **tree reweighted BP** approximation [Wainwright05map; Kolmogorov06]. This is similar to the Bethe approximation to the entropy except for the crucial ρ_{st} weights. So long as $\rho_{st} > 0$ for all edges (s,t) , this gives a valid concave upper bound on the exact entropy.

The edge appearance probabilities live in a space called the **spanning tree polytope**. This is because they are constrained to arise from a distribution over trees. Figure 10.6 gives an example of a graph and three of its spanning trees. Suppose each tree has equal weight under ρ . The edge f occurs in 1 of the 3 trees, so $\rho_f = 1/3$. The edge e occurs in 2 of the 3 trees, so $\rho_e = 2/3$. The edge b

appears in all of the trees, so $\rho_b = 1$. And so on. Ideally we can find a distribution ρ , or equivalently edge probabilities in the spanning tree polytope, that make the above bound as tight as possible. An algorithm to do this is described in [Wainwright05]. A simpler approach is to use all single edges with weight $\rho_e = 1/E$.

What about the set we are optimizing over? We require $\boldsymbol{\mu}(T) \in \mathbb{M}(T)$ for each tree T , which means enforcing normalization and local consistency. Since we have to do this for every tree, we are enforcing normalization and local consistency on every edge. Thus we are effectively optimizing in the pseudo-marginal polytope $\mathbb{L}(G)$. So our final optimization problem is as follows:

$$\max_{\boldsymbol{\tau} \in \mathbb{L}(G)} \boldsymbol{\tau}^T \boldsymbol{\theta} + \mathbb{H}_{\text{TRBP}}(\boldsymbol{\tau}) \geq \log Z \quad (10.109)$$

10.4.5.2 Message passing implementation

The simplest way to minimize Equation (10.109) is a modification of belief propagation known as **tree reweighted belief propagation**. The message from t to s is now a function of all messages sent from other neighbors v to t , as before, but now it is also a function of the message sent from s to t . Specifically, we have the following [Monster]:

$$m_{t \rightarrow s}(x_s) \propto \sum_{x_t} \exp \left(\frac{1}{\rho_{st}} \theta_{st}(x_s, x_t) + \theta_t(x_t) \right) \frac{\prod_{v \in \text{nbr}(t) \setminus s} [m_{v \rightarrow t}(x_t)]^{\rho_{vt}}}{[m_{s \rightarrow t}(x_t)]^{1-\rho_{ts}}} \quad (10.110)$$

At convergence, the node and edge pseudo marginals are given by

$$\tau_s(x_s) \propto \exp(\theta_s(x_s)) \prod_{v \in \text{nbr}(s)} [m_{v \rightarrow s}(x_s)]^{\rho_{vs}} \quad (10.111)$$

$$\tau_{st}(x_s, x_t) \propto \varphi_{st}(x_s, x_t) \frac{\prod_{v \in \text{nbr}(s) \setminus t} [m_{v \rightarrow s}(x_s)]^{\rho_{vs}}}{[m_{t \rightarrow s}(x_s)]^{1-\rho_{st}}} \frac{\prod_{v \in \text{nbr}(t) \setminus s} [m_{v \rightarrow t}(x_t)]^{\rho_{vt}}}{[m_{s \rightarrow t}(x_t)]^{1-\rho_{ts}}} \quad (10.112)$$

$$\varphi_{st}(x_s, x_t) \triangleq \exp \left(\frac{1}{\rho_{st}} \theta_{st}(x_s, x_t) + \theta_s(x_s) + \theta_t(x_t) \right) \quad (10.113)$$

If $\rho_{st} = 1$ for all edges $(s, t) \in E$, the algorithm reduces to the standard LBP algorithm. However, the condition $\rho_{st} = 1$ implies every edge is present in every spanning tree with probability 1, which is only possible if the original graph is a tree. Hence the method is only equivalent to standard LBP on trees, when the method is of course exact.

In general, this message passing scheme is not guaranteed to converge to the unique global optimum. One can devise double-loop methods that are guaranteed to converge [Hazan08], but in practice, using damped updates as in Equation Main Equation (9.77) is often sufficient to ensure convergence.

10.4.5.3 Max-product version

We can modify TRBP to solve the MAP estimation problem (as opposed to estimating posterior marginals) by replacing sums with products in Equation (10.110) (see [Monster] for details). This is guaranteed to converge to the LP relaxation discussed in Section 9.3.3 under a suitable scheduling known as **sequential tree-reweighted message passing** [Kolmogorov06].

1
2

10.4.6 Other tractable versions of convex BP

3
4

It is possible to upper bound the entropy using convex combinations of other kinds of tractable models besides trees. One example is a **planar MRF** (one where the graph has no edges that cross), with binary nodes and no external field, i.e., the model has the form $p(\mathbf{x}) \propto \exp(\sum_{(s,t) \in E} \theta_{st} x_s x_t)$. It turns out that it is possible to perform exact inference in this model. Hence one can use convex combinations of such graphs which can sometimes yield more accurate results than TRBP, albeit at higher computational cost. See [Globerson07] for details, and [Schraudolph10] for a related *exact* method for planar Ising models.

1011121314151617181920212223242526272829303132333435363738394041424344454647

11 Monte Carlo Inference

12 Markov Chain Monte Carlo (MCMC) inference

13 Sequential Monte Carlo (SMC) inference

13.1 More applications of particle filtering

In this section, we give some examples of particle filtering applied to some state estimation problems in different kinds of state-space models. We focus on using the simplest kind of SMC algorithm, namely the bootstrap filter (Main Section 13.2.3.1).

13.1.1 1d pendulum model with outliers

In this section, we consider the pendulum example from Section 8.2.2. Rather than Gaussian observation noise, we assume that some fraction $p = 0.4$ of the observations are outliers, coming from a $\text{Unif}(-2, 2)$ distribution. (These could represent a faulty sensor, for example.) In this case, the bootstrap filter is more robust than deterministic filters, as shown in Figure 13.1, since it can handle multimodal posteriors induced by uncertainty about which observations are signal and which are noise. By contrast, EKF and UKF assume a unimodal (Gaussian) posterior, which is very sensitive to outliers.

13.1.2 Visual object tracking

In Main Section 13.1.2, we tracked an object given noisy measurements of its location, as estimated by some kind of distance sensor. A harder problem is to track an object just given a sequence of frames from a video camera. This is called **visual tracking**. In this section we consider an example where the object is a remote-controlled helicopter [Nummiaro03]. We will use a simple linear motion model for the centroid of the object, and a color histogram for the likelihood model, using **Bhattacharya distance** to compare histograms.

Figure 13.2 shows some example frames. The system uses $S = 250$ particles, with an effective sample size of 134. (a) shows the belief state at frame 1. The system has had to resample 5 times to keep the effective sample size above the threshold of 150; (b) shows the belief state at frame 251; the red lines show the estimated location of the center of the object over the last 250 frames. (c) shows that the system can handle **visual clutter** (the hat of the human operator), as long as it does not have the same color as the target object; (d) shows that the system is confused between the grey of the helicopter and the grey of the building (the posterior is bimodal, but the green ellipse, representing the posterior mean and covariance, is in between the two modes); (e) shows that the probability mass has shifted to the wrong mode: i.e., the system has lost track; (f) shows the particles spread out over the gray building; recovery of the object is very unlikely from this state using this

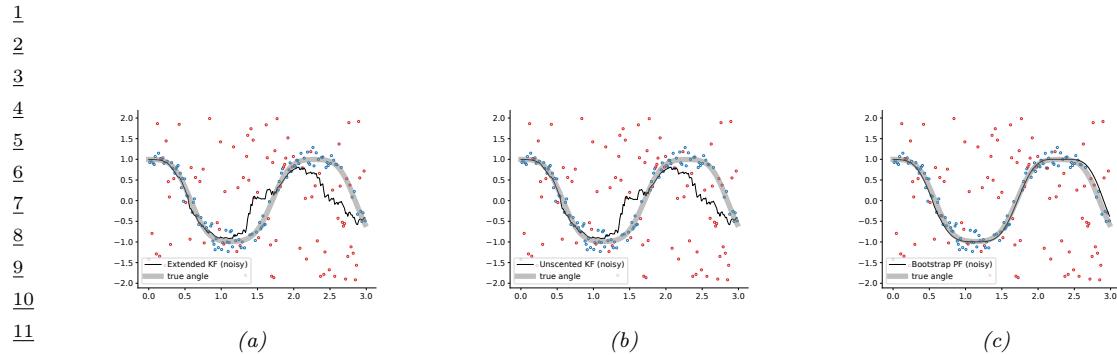


Figure 13.1: Filtering algorithms applied to the noisy pendulum model with 40% outliers (shown in red). (a) EKF. (b) UKF. (c) Bootstrap filter. Generated by `pendulum_1d.ipynb`.

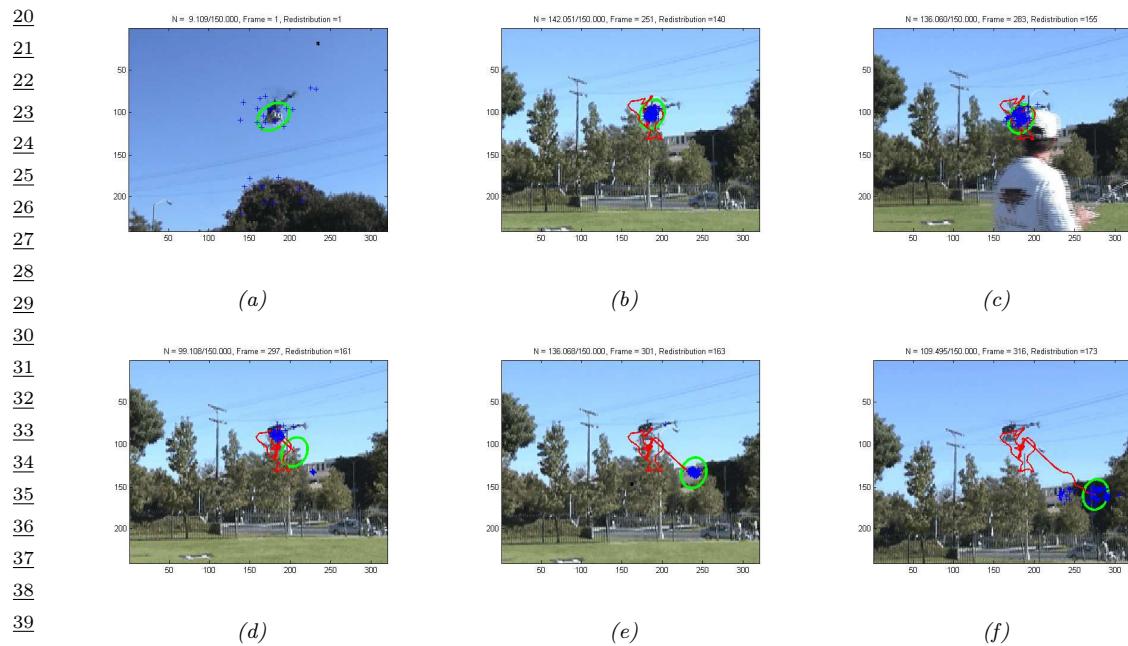


Figure 13.2: Example of particle filtering applied to visual object tracking, based on color histograms. Blue dots are posterior samples, green ellipse is Gaussian approximation to posterior. (a-c) Successful tracking. (d): Tracker gets distracted by an outlier gray patch in the background, and moves the posterior mean away from the object. (e-f): Losing track. See text for details. Used with kind permission of Sébastien Paris.

1
2 proposal.

3 The simplest way to improve performance of this method is to use more particles. A more efficient
4 approach is to perform **tracking by detection**, by running an object detector over the image
5 every few frames, and to use these as proposals (see Main Section 13.3). This provides a way to
6 combine discriminative, bottom-up object detection (which can fail in the presence of occlusion)
7 with generative, top-down tracking (which can fail if there are unpredictable motions, or new objects
8 entering the scene). See e.g., [**Hess2009; VanGool2009; Gurkan2019; Okada2019**] for further
9 details.

10

11 13.1.3 Online parameter estimation

12

13 It is tempting to use particle filtering to perform online Bayesian inference for the parameters of a
14 model $p(\mathbf{y}_t | \mathbf{x}_t, \boldsymbol{\theta})$, just as we did using the Kalman filter for linear regression (Main Section 29.7.2) and
15 the EKF for MLPs (Main Section 17.5.2). However, this technique will not work. The reason is that
16 static parameters correspond to a dynamical model with zero system noise, $p(\boldsymbol{\theta}_t | \boldsymbol{\theta}_{t-1}) = \delta(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1})$.
17 However, such a deterministic model causes problems for particle filtering, because the particles can
18 only be reweighted by the likelihood, but cannot be moved by the deterministic transition model.
19 Thus the diversity in the trajectories rapidly goes to zero, and the posterior collapses [**Kantas2015**].

20 It is possible to add **artificial process noise**, but this causes the influence of earlier observations
21 to decay exponentially with time, and also “washes out” the initial prior. In Main Section 13.6.3, we
22 present a solution to this problem based on SMC samplers, which generalize the particle filter by
23 allowing static variables to be turned into a sequence by adding auxiliary random variables.

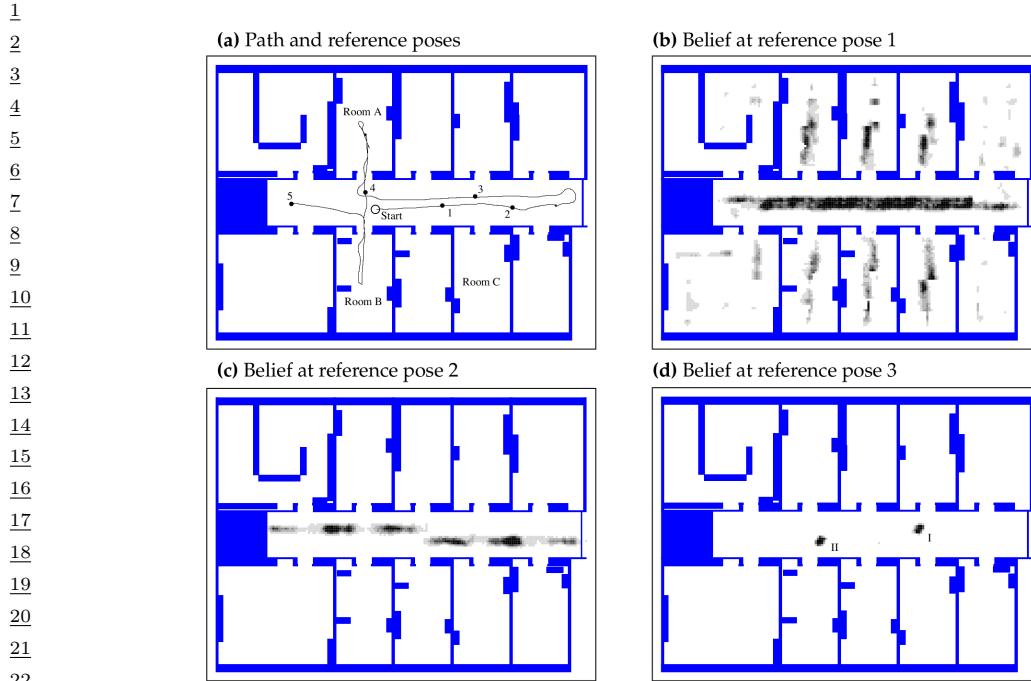
24

25 13.1.4 Monte Carlo robot localization

26

27 Consider a mobile robot wandering around an indoor environment. We will assume that it already
28 has a map of the world, represented in the form of an **occupancy grid**, which just specifies whether
29 each grid cell is empty space or occupied by something solid like a wall. The goal is for the robot to
30 estimate its location. (See also Main Section 13.4.3, where we discuss the problem of simultaneously
31 localizing and mapping the environment.) This can be solved optimally using an HMM filter (also
32 called a **histogram filter** [**Jonschkowski2016**]), since we are assuming the state space is discrete.
33 However, since the number of states, K , is often very large, the $O(K^2)$ time complexity per update is
34 prohibitive. We can use a particle filter as a sparse approximation to the belief state. This is known
35 as **Monte Carlo localization** [**Thrun06**].

36 Figure 13.3 gives an example of the method in action. The robot uses a sonar range finder, so
37 it can only sense distance to obstacles. It starts out with a uniform prior, reflecting the fact that
38 the owner of the robot may have turned it on in an arbitrary location. (Figuring out where you are,
39 starting from a uniform prior, is called **global localization**.) After the first scan, which indicates
40 two walls on either side, the belief state is shown in (b). The posterior is still fairly broad, since the
41 robot could be in any location where the walls are fairly close by, such as a corridor or any of the
42 narrow rooms. After moving to location 2, the robot is pretty sure it must be in a corridor and not a
43 room, as shown in (c). After moving to location 3, the sensor is able to detect the end of the corridor.
44 However, due to symmetry, it is not sure if it is in location I (the true location) or location II. (This
45 is an example of **perceptual aliasing**, which refers to the fact that different things may look the
46 same.) After moving to locations 4 and 5, it is finally able to figure out precisely where it is (not
47



23 Figure 13.3: Illustration of Monte Carlo localization for a mobile robot in an office environment using a sonar
24 sensor. From Figure 8.7 of [Thrun06]. Used with kind permission of Sebastian Thrun.
25

26
27
28 shown). The whole process is analogous to someone getting lost in an office building, and wandering
29 the corridors until they see a sign they recognize.

30

31 13.2 Particle MCMC methods

32
33 In this section, we discuss some other sampling techniques that leverage the fact that SMC can give an
34 unbiased estimate of the normalization constant Z for the target distribution. This can be useful for
35 sampling with models where the exact likelihood is intractable. These are called **pseudo-marginal**
36 **methods** [Andrieu2009].

37 To be more precise, note that the SMC algorithm can be seen as mapping a stream of random
38 numbers \mathbf{u} into a set of samples, $\mathbf{z}_{1:T}^{1:N_s}$. We need random numbers $\mathbf{u}_{z,1:T}^{1:N_s}$ to specify the hidden
39 states that are sampled at each step (using the inverse CDF of the proposal), and random numbers
40 $\mathbf{u}_{a,1:T-1}^{1:N_s}$ to control the ancestor indices that are chosen (using the resampling algorithm), where each
41 $u_{z,t}^i, u_{a,t}^i \sim \text{Unif}(0, 1)$. The normalization constant is also a function of these random numbers, so we
42 denote it $\hat{Z}_t(\mathbf{u})$, where

43
44
45
$$\hat{Z}_t(\mathbf{u}) = \prod_{s=1}^t \frac{1}{N_s} \sum_{n=1}^{N_s} \tilde{w}_t^n(\mathbf{u}) \quad (13.1)$$

46
47

1 One can show (see e.g., [Naesseth2019]) that
 2

$$3 \quad \mathbb{E} [\hat{Z}_t(\mathbf{u})] = Z_t \quad (13.2)$$

4 where the expectation is wrt the distribution of \mathbf{u} , denoted $\tau(\mathbf{u})$. (Note that \mathbf{u} can be represented
 5 by a random seed.) This allows us to plug SMC inside other MCMC algorithms, as we show below.

6 Such methods are often used by **probabilistic programming systems** (see e.g., [Zhou2020pps]),
 7 since PPLs often define models with many latent variable models defined implicitly (via sampling
 8 statements), as discussed in Main Section 4.6.6.
 9

10 13.2.1 Particle Marginal Metropolis Hastings

11 Suppose we want to compute the parameter posterior $p(\boldsymbol{\theta}|\mathbf{y}) = p(\mathbf{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})/p(\mathbf{y})$ for a latent variable
 12 model with prior $p(\boldsymbol{\theta})$ and likelihood $p(\mathbf{y}|\boldsymbol{\theta}) = \int p(\mathbf{y}, \mathbf{h}|\boldsymbol{\theta})d\mathbf{h}$, where \mathbf{h} are latent variables (e.g., from
 13 a SSM). We can use Metropolis Hastings (Main Section 12.2) to avoid having to compute the partition
 14 function $p(\mathbf{y})$. However, in many cases it is intractable to compute the likelihood $p(\mathbf{y}|\boldsymbol{\theta})$ itself, due
 15 to the need to integrate over \mathbf{h} . This makes it hard to compute the MH acceptance probability
 16

$$17 \quad A = \min \left(1, \frac{p(\mathbf{y}|\boldsymbol{\theta}')p(\boldsymbol{\theta}')q(\boldsymbol{\theta}^{j-1}|\boldsymbol{\theta}')}{p(\mathbf{y}|\boldsymbol{\theta}^{j-1})p(\boldsymbol{\theta}^{j-1})q(\boldsymbol{\theta}'|\boldsymbol{\theta}^{j-1})} \right) \quad (13.3)$$

18 where $\boldsymbol{\theta}^{j-1}$ is the parameter vector at iteration $j - 1$, and we are proposing $\boldsymbol{\theta}'$ from $q(\boldsymbol{\theta}'|\boldsymbol{\theta}^{j-1})$.
 19 However, we can use SMC to compute $\hat{Z}(\boldsymbol{\theta})$ as an unbiased approximation to $p(\mathbf{y}|\boldsymbol{\theta})$, which can be
 20 used to evaluate the MH acceptance ratio:
 21

$$22 \quad A = \min \left(1, \frac{\hat{Z}(\mathbf{u}', \boldsymbol{\theta}')p(\boldsymbol{\theta}')q(\boldsymbol{\theta}^{j-1}|\boldsymbol{\theta}')}{\hat{Z}(\mathbf{u}^{j-1}, \boldsymbol{\theta}^{j-1})p(\boldsymbol{\theta}^{j-1})q(\boldsymbol{\theta}'|\boldsymbol{\theta}^{j-1})} \right) \quad (13.4)$$

23 More precisely, we apply MH to an extended space, where we sample both the parameters $\boldsymbol{\theta}$ and the
 24 randomness \mathbf{u} for SMC.
 25

26 We can generalize the above to return samples of the latent states as well as the latent parameters,
 27 by sampling a single trajectory from
 28

$$29 \quad p(\mathbf{h}_{1:T}|\boldsymbol{\theta}, \mathbf{y}) \approx \hat{p}(\mathbf{h}|\boldsymbol{\theta}, \mathbf{y}, \mathbf{u}) = \sum_{i=1}^{N_s} W_T^i \delta(\mathbf{h}_{1:T} - \mathbf{h}_{1:T}^i) \quad (13.5)$$

30 by using the internal samples generated by SMC. Thus we can sample $\boldsymbol{\theta}$ and \mathbf{h} jointly. This is
 31 called the **particle marginal Metropolis Hastings (PMMH)** algorithm [Andrieu2010]. See
 32 Algorithm 13.1 for the pseudocode. See e.g. [Dahlin2015] for more practical details.
 33

34 13.2.2 Particle Independent Metropolis Hastings

35 Now suppose we just want to sample the latent states \mathbf{h} , with the parameters $\boldsymbol{\theta}$ being fixed. In
 36 this case we can simplify PMMH algorithm by not sampling $\boldsymbol{\theta}$. Since the latent states \mathbf{h} are now
 37 sampled independently of the state of the Markov chain, this is called the **particle independent**
 38

1
2 **Algorithm 13.1:** Particle Marginal Metropolis-Hastings
3 1 **for** $j = 1 : J$ **do**
4 2 Sample $\theta' \sim q(\theta' | \theta^{j-1})$, $\mathbf{u}' \sim \tau(\mathbf{u}')$, $\mathbf{h}' \sim \hat{p}(\mathbf{h}' | \theta', \mathbf{y}, \mathbf{u}')$
5 3 Compute $\hat{Z}(\mathbf{u}', \theta')$ using SMC
6 4 Compute A using Equation (13.4)
7 5 Sample $u \sim \text{Unif}(0, 1)$
8 6 **if** $u < A$ **then**
9 7 | Set $\theta^j = \theta'$, $\mathbf{u}^j = \mathbf{u}'$, $\mathbf{h}^j = \mathbf{h}'$
10 8 **else**
11 9 | Set $\theta^j = \theta^{j-1}$, $\mathbf{u}^j = \mathbf{u}^{j-1}$, $\mathbf{h}^j = \mathbf{h}^{j-1}$

13
14
15 **Algorithm 13.2:** Particle Independent Metropolis-Hastings
16 1 **for** $j = 1 : J$ **do**
17 2 Sample $\mathbf{u}' \sim \tau(\mathbf{u}')$, $\mathbf{h}' \sim \hat{p}(\mathbf{h}' | \theta, \mathbf{y}, \mathbf{u}')$
18 3 Compute $\hat{Z}(\mathbf{u}', \theta)$ using SMC
19 4 Compute $A = \min\left(1, \frac{\hat{Z}(\mathbf{u}', \theta)}{\hat{Z}(\mathbf{u}^{j-1}, \theta)}\right)$
20 5 Sample $u \sim \text{Unif}(0, 1)$
21 6 **if** $u < A$ **then**
22 7 | Set $\mathbf{u}^j = \mathbf{u}'$, $\mathbf{h}^j = \mathbf{h}'$
23 8 **else**
24 9 | Set $\mathbf{u}^j = \mathbf{u}^{j-1}$, $\mathbf{h}^j = \mathbf{h}^{j-1}$

27
28
29 **MH** algorithm. The acceptance ratio term also simplifies, since we can drop all terms involving θ .
30 See Algorithm 13.2 for the pseudocode.

31 One might wonder what the advantage of PIMH is over just using SMC. The answer is that PIMH
32 can return unbiased estimates of smoothing expectations, such as

33
34
$$\pi(\varphi) = \int \varphi(\mathbf{h}_{1:T}) \pi(\mathbf{h}_{1:T} | \theta, \mathbf{y}) d\mathbf{h}_{1:T} \quad (13.6)$$

35

36 whereas estimating this directly with SMC results in a consistent but biased estimate (in contrast to
37 the estimate of Z , which is unbiased). For details, see [Middleton2019].

38
39 **13.2.3 Particle Gibbs**

40
41 In PMMH, we define a transition kernel that, given $(\theta^{(j-1)}, \mathbf{h}^{(j-1)})$, generates a sample $(\theta^{(j)}, \mathbf{h}^{(j)})$,
42 while leaving the target distribution invariant. Another way to perform this task is to use **particle**
43 **Gibbs sampling**, which avoids needing to specify any proposal distributions. In this approach, we
44 first sample $N - 1$ trajectories $\mathbf{h}_{1:T}^{1:N-1} \sim p(\mathbf{h} | \theta^{(j-1)}, \mathbf{y})$ using **conditional SMC**, keeping the N 'th
45 trajectory fixed at the retained particle $\mathbf{h}_{1:T}^N = \mathbf{h}^{(j-1)}$. We then sample a new value for $\mathbf{h}^{(j)}$ from the
46 empirical distribution $\hat{\pi}_T(\mathbf{h}_{1:T}^{1:N})$. Finally we sample $\theta^{(j)} \sim p(\theta | \mathbf{h}^{(j)})$. For details, see [Andrieu2010].

1 Another variant, known as **particle Gibbs with ancestor sampling**, is discussed in [Lindsten2014];
2 it is particularly well-suited to state-space models.
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 III

Prediction

14 Predictive models: an overview

15 Generalized linear models

15.1 Variational inference for logistic regression

In this section we discuss a variational approach to Bayesian inference for logistic regression models based on local bounds to the likelihood. We will use a Gaussian prior, $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}_0, \mathbf{V}_0)$. We will create a “Gaussian-like” lower bound to the likelihood, which becomes conjugate to this prior. We then iteratively improve this lower bound.

15.1.1 Binary logistic regression

In this section, we discuss VI for binary logistic regression. Our presentation follows [BishopBook].

Let us first rewrite the likelihood for a single observation as follows:

$$p(y_n | \mathbf{x}_n, \mathbf{w}) = \sigma(\eta_n)^{y_n} (1 - \sigma(\eta_n))^{1-y_n} \quad (15.1)$$

$$= \left(\frac{1}{1 + e^{-\eta_n}} \right)^{y_n} \left(1 - \frac{1}{1 + e^{-\eta_n}} \right)^{1-y_n} \quad (15.2)$$

$$= e^{-\eta_n y_n} \frac{e^{-\eta_n}}{1 + e^{-\eta_n}} = e^{-\eta_n y_n} \sigma(-\eta_n) \quad (15.3)$$

where $\eta_n = \mathbf{w}^\top \mathbf{x}_n$ are the logits. This is not conjugate to the Gaussian prior. So we will use the following “Gaussian-like” variational lower bound to the sigmoid function, proposed in [Jaakkola96b; Jaakkola00]:

$$\sigma(\eta_n) \geq \sigma(\psi_n) \exp \left[(\eta_n - \psi_n)/2 - \lambda(\psi_n)(\eta_n^2 - \psi_n^2) \right] \quad (15.4)$$

where ψ_n is the variational parameter for datapoint n , and

$$\lambda(\psi) \triangleq \frac{1}{4\psi} \tanh(\psi/2) = \frac{1}{2\psi} \left[\sigma(\psi) - \frac{1}{2} \right] \quad (15.5)$$

We shall refer to this as the **JJ bound**, after its inventors, Jaakkola and Jordan. See Figure 15.1(a) for a plot, and see Section 6.3.4.2 for a derivation.

Using this bound, we can write

$$p(y_n | \mathbf{x}_n, \mathbf{w}) = e^{-\eta_n y_n} \sigma(-\eta_n) \geq e^{-\eta_n y_n} \sigma(\psi_n) \exp \left[(-\eta_n + \psi_n)/2 - \lambda(\psi_n)(\eta_n^2 - \psi_n^2) \right] \quad (15.6)$$

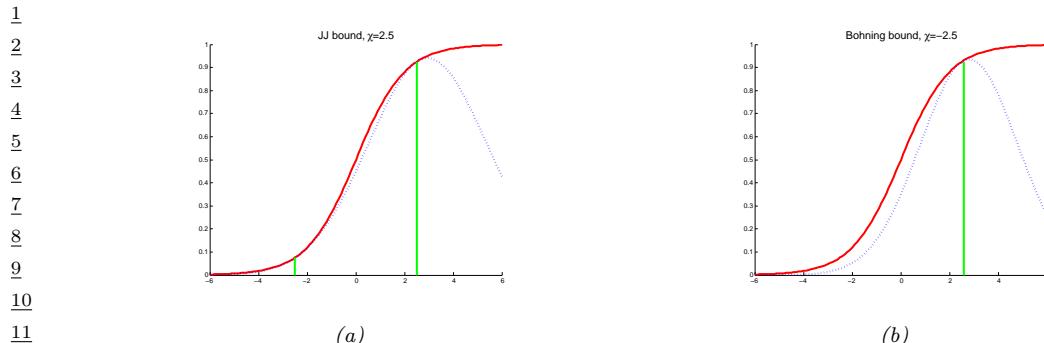


Figure 15.1: Quadratic lower bounds on the sigmoid (logistic) function. In solid red, we plot $\sigma(x)$ vs x . In dotted blue, we plot the lower bound $L(x, \psi)$ vs x for $\psi = 2.5$. (a) JJ bound. This is tight at $\psi = \pm 2.5$. (b) Bohning bound (Section 15.1.2.2). This is tight at $\psi = 2.5$. Generated by `sigmoid_lower_bounds.ipynb`.

We can now lower bound the log joint as follows:

$$\log p(\mathbf{y}|\mathbf{X}, \mathbf{w}) + \log p(\mathbf{w}) \geq -\frac{1}{2}(\mathbf{w} - \boldsymbol{\mu}_0)^\top \mathbf{V}_0^{-1}(\mathbf{w} - \boldsymbol{\mu}_0) \quad (15.7)$$

$$+ \sum_{n=1}^N [\eta_n(y_n - 1/2) - \lambda(\psi_n) \mathbf{w}^\top (\mathbf{x}_n \mathbf{x}_n^\top) \mathbf{w}] \quad (15.8)$$

²⁴ Since this is a quadratic function of w , we can derive a Gaussian posterior approximation as follows:

$$q(\mathbf{w}|\psi) = \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}_N, \mathbf{V}_N) \quad (15.9)$$

$$\boldsymbol{\mu}_N = \mathbf{V}_N \left(\mathbf{V}_0^{-1} \boldsymbol{\mu}_0 + \sum_{n=1}^N (y_n - 1/2) \boldsymbol{x}_n \right) \quad (15.10)$$

$$\frac{29}{30} \quad \mathbf{V}_N^{-1} = \mathbf{V}_0^{-1} + 2 \sum_{n=1}^N \lambda(\psi_n) \mathbf{x}_n \mathbf{x}_n^\top \quad (15.11)$$

³² This is more flexible than a Laplace approximation, since the variational parameters ψ can be used to optimize the curvature of the posterior covariance. To find the optimal ψ , we can maximize the ELBO, which is given by

$$\log p(\mathbf{y}|\mathbf{X}) = \log \int p(\mathbf{y}|\mathbf{X}, \mathbf{w})p(\mathbf{w})d\mathbf{w} \geq \log \int h(\mathbf{w}, \psi)p(\mathbf{w})d\mathbf{w} = \mathbb{L}(\psi) \quad (15.12)$$

where

$$h(\boldsymbol{w}, \boldsymbol{\psi}) = \prod_{n=1}^N \sigma(\boldsymbol{\psi}_n) \exp \left[\eta_n y_n - (\eta_n + \boldsymbol{\psi}_n)/2 - \lambda(\boldsymbol{\psi}_n)(\eta_n^2 - \boldsymbol{\psi}_n^2) \right] \quad (15.13)$$

⁴³ We can evaluate the lower bound analytically to get

$$\mathbb{L}(\boldsymbol{\psi}) = \frac{1}{2} \log \frac{|\mathbf{V}_N|}{|\mathbf{V}_0|} + \frac{1}{2} \boldsymbol{\mu}_N^\top \mathbf{V}_N^{-1} \boldsymbol{\mu}_N - \frac{1}{2} \boldsymbol{\mu}_0^\top \mathbf{V}_0^{-1} \boldsymbol{\mu}_0 + \sum_{n=1}^N \left[\log \sigma(\boldsymbol{\psi}_n) - \frac{1}{2} \boldsymbol{\psi}_n^\top + \lambda(\boldsymbol{\psi}_n) \boldsymbol{\psi}_n^2 \right] \quad (15.14)$$

If we solve for $\nabla_{\psi} \bar{L}(\psi) = \mathbf{0}$, we get the following iterative update equation for each variational parameter:

$$(\psi_n^{\text{new}})^2 = \mathbf{x}_n \mathbb{E} [\mathbf{w} \mathbf{w}^T] \mathbf{x}_n = \mathbf{x}_n (\mathbf{V}_N + \boldsymbol{\mu}_N \boldsymbol{\mu}_N^T) \mathbf{x}_n \quad (15.15)$$

One we have estimated ψ_n , we can plug it into the above Gaussian approximation $q(\mathbf{w}|\psi)$.

15.1.2 Multinomial logistic regression

In this section we discuss how to approximate the posterior $p(\mathbf{w}|\mathcal{D})$ for multinomial logistic regression using variational inference, extending the approach of Main Section 15.3.8 to the multi-class case. The key idea is to create a “Gaussian-like” lower bound on the multi-class logistic regression likelihood due to [Bohning92]. We can then compute the variational posterior in closed form. This will let us deterministically optimize the ELBO.

Let $\mathbf{y}_i \in \{0, 1\}^C$ be a one-hot label vector, and define the logits for example i to be

$$\boldsymbol{\eta}_i = [\mathbf{x}_i^T \mathbf{w}_1, \dots, \mathbf{x}_i^T \mathbf{w}_C] \quad (15.16)$$

If we define $\mathbf{X}_i = \mathbf{I} \otimes \mathbf{x}_i$, where \otimes is the kronecker product, and \mathbf{I} is $C \times C$ identity matrix, then we can write the logits as $\boldsymbol{\eta}_i = \mathbf{X}_i \mathbf{w}$. (For example, if $C = 2$ and $\mathbf{x}_i = [1, 2, 3]$, we have $\mathbf{X}_i = [1, 2, 3, 0, 0, 0; 0, 0, 0, 1, 2, 3]$.) Then the likelihood is given by

$$p(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \prod_{i=1}^N \exp[\mathbf{y}_i^T \boldsymbol{\eta}_i - \text{lse}(\boldsymbol{\eta}_i)] \quad (15.17)$$

where $\text{lse}()$ is the log-sum-exp function

$$\text{lse}(\boldsymbol{\eta}_i) \triangleq \log \left(\sum_{c=1}^C \exp(\eta_{ic}) \right) \quad (15.18)$$

For identifiability, we can set $\mathbf{w}_C = \mathbf{0}$, so

$$\text{lse}(\boldsymbol{\eta}_i) = \log \left(1 + \sum_{m=1}^M \exp(\eta_{im}) \right) \quad (15.19)$$

where $M = C - 1$. (We subtract 1 so that in the binary case, $M = 1$.)

15.1.2.1 Bohning’s quadratic bound to the log-sum-exp function

The above likelihood is not conjugate to the Gaussian prior. However, we will now can convert it to a quadratic form. Consider a Taylor series expansion of the log-sum-exp function around $\psi_i \in \mathbb{R}^M$:

$$\text{lse}(\boldsymbol{\eta}_i) = \text{lse}(\psi_i) + (\boldsymbol{\eta}_i - \psi_i)^T \mathbf{g}(\psi_i) + \frac{1}{2} (\boldsymbol{\eta}_i - \psi_i)^T \mathbf{H}(\psi_i) (\boldsymbol{\eta}_i - \psi_i) \quad (15.20)$$

$$\mathbf{g}(\psi_i) = \exp[\psi_i - \text{lse}(\psi_i)] = \text{softmax}(\psi_i) \quad (15.21)$$

$$\mathbf{H}(\psi_i) = \text{diag}(\mathbf{g}(\psi_i)) - \mathbf{g}(\psi_i) \mathbf{g}(\psi_i)^T \quad (15.22)$$

where \mathbf{g} and \mathbf{H} are the gradient and Hessian of lse, and $\psi_i \in \mathbb{R}^M$, where $M = C - 1$ is the number of classes minus 1. An upper bound to lse can be found by replacing the Hessian matrix $\mathbf{H}(\psi_i)$ with a matrix \mathbf{A}_i such that $\mathbf{A}_i \succeq \mathbf{H}(\psi_i)$ for all ψ_i . [Bohning92] showed that this can be achieved if we use the matrix $\mathbf{A}_i = \frac{1}{2} \left[\mathbf{I}_M - \frac{1}{M+1} \mathbf{1}_M \mathbf{1}_M^\top \right]$. In the binary case, this becomes $A_i = \frac{1}{2}(1 - \frac{1}{2}) = \frac{1}{4}$.

Note that \mathbf{A}_i is independent of ψ_i ; however, we still write it as \mathbf{A}_i (rather than dropping the i subscript), since other bounds that we consider below will have a data-dependent curvature term. The upper bound on lse therefore becomes

$$\text{lse}(\boldsymbol{\eta}_i) \leq \frac{1}{2} \boldsymbol{\eta}_i^\top \mathbf{A}_i \boldsymbol{\eta}_i - \mathbf{b}_i^\top \boldsymbol{\eta}_i + c_i \quad (15.23)$$

$$\mathbf{A}_i = \frac{1}{2} \left[\mathbf{I}_M - \frac{1}{M+1} \mathbf{1}_M \mathbf{1}_M^\top \right] \quad (15.24)$$

$$\mathbf{b}_i = \mathbf{A}_i \psi_i - \mathbf{g}(\psi_i) \quad (15.25)$$

$$c_i = \frac{1}{2} \psi_i^\top \mathbf{A}_i \psi_i - \mathbf{g}(\psi_i)^\top \psi_i + \text{lse}(\psi_i) \quad (15.26)$$

where $\psi_i \in \mathbb{R}^M$ is a vector of variational parameters.

We can use the above result to get the following lower bound on the softmax likelihood:

$$\log p(y_i = c | \mathbf{x}_i, \mathbf{w}) \geq \left[\mathbf{y}_i^\top \mathbf{X}_i \mathbf{w} - 4 \frac{1}{2} \mathbf{w}^\top \mathbf{X}_i \mathbf{A}_i \mathbf{X}_i \mathbf{w} + \mathbf{b}_i^\top \mathbf{X}_i \mathbf{w} - c_i \right]_c \quad (15.27)$$

To simplify notation, define the pseudo-measurement

$$\tilde{\mathbf{y}}_i \triangleq \mathbf{A}_i^{-1} (\mathbf{b}_i + \mathbf{y}_i) \quad (15.28)$$

Then we can get a “Gaussianized” version of the observation model:

$$p(\mathbf{y}_i | \mathbf{x}_i, \mathbf{w}) \geq f(\mathbf{x}_i, \psi_i) \mathcal{N}(\tilde{\mathbf{y}}_i | \mathbf{X}_i \mathbf{w}, \mathbf{A}_i^{-1}) \quad (15.29)$$

where $f(\mathbf{x}_i, \psi_i)$ is some function that does not depend on \mathbf{w} . Given this, it is easy to compute the posterior $q(\mathbf{w}) = \mathcal{N}(\mathbf{m}_N, \mathbf{V}_N)$, using Bayes rule for Gaussians.

Given the posterior, we can write the ELBO as follows:

$$\mathcal{L}(\psi) \triangleq -D_{\text{KL}}(q(\mathbf{w}) \| p(\mathbf{w})) + \mathbb{E}_q \left[\sum_{i=1}^N \log p(y_i | \mathbf{x}_i, \mathbf{w}) \right] \quad (15.30)$$

$$= -D_{\text{KL}}(q(\mathbf{w}) \| p(\mathbf{w})) + \mathbb{E}_q \left[\sum_{i=1}^N \mathbf{y}_i^\top \boldsymbol{\eta}_i - \text{lse}(\boldsymbol{\eta}_i) \right] \quad (15.31)$$

$$= -D_{\text{KL}}(q(\mathbf{w}) \| p(\mathbf{w})) + \sum_{i=1}^N \mathbf{y}_i^\top \mathbb{E}_q [\boldsymbol{\eta}_i] - \sum_{i=1}^N \mathbb{E}_q [\text{lse}(\boldsymbol{\eta}_i)] \quad (15.32)$$

where $p(\mathbf{w}) = \mathcal{N}(\mathbf{w} | \mathbf{m}_0, \mathbf{V}_0)$ is the prior and $q(\mathbf{w}) = \mathcal{N}(\mathbf{w} | \mathbf{m}_N, \mathbf{V}_N)$ is the approximate posterior.

The first term is just the KL divergence between two Gaussians, which is given by

$$\begin{aligned} -D_{\text{KL}}(\mathcal{N}(\mathbf{m}_N, \mathbf{V}_N) \| \mathcal{N}(\mathbf{m}_0, \mathbf{V}_0)) &= -\frac{1}{2} [\text{tr}(\mathbf{V}_N \mathbf{V}_0^{-1}) - \log |\mathbf{V}_N \mathbf{V}_0^{-1}| \\ &\quad + (\mathbf{m}_N - \mathbf{m}_0)^\top \mathbf{V}_0^{-1} (\mathbf{m}_N - \mathbf{m}_0) - DM] \end{aligned} \quad (15.33)$$

where DM is the dimensionality of the Gaussian, and we assume a prior of the form $p(\mathbf{w}) = \mathcal{N}(\mathbf{m}_0, \mathbf{V}_0)$, where typically $\mathbf{\mu}_0 = \mathbf{0}_{DM}$, and \mathbf{V}_0 is block diagonal. The second term is simply

$$\sum_{i=1}^N \mathbf{y}_i^\top \mathbb{E}_q [\boldsymbol{\eta}_i] = \sum_{i=1}^N \mathbf{y}_i^\top \tilde{\mathbf{m}}_i \quad (15.34)$$

where $\tilde{\mathbf{m}}_i \triangleq \mathbf{X}_i \mathbf{m}_N$. The final term can be lower bounded by taking expectations of our quadratic upper bound on lse as follows:

$$-\sum_{i=1}^N \mathbb{E}_q [\text{lse}(\boldsymbol{\eta}_i)] \geq -\frac{1}{2} \text{tr}(\mathbf{A}_i \tilde{\mathbf{V}}_i) - \frac{1}{2} \tilde{\mathbf{m}}_i^\top \mathbf{A}_i \tilde{\mathbf{m}}_i + \mathbf{b}_i^\top \tilde{\mathbf{m}}_i - c_i \quad (15.35)$$

where $\tilde{\mathbf{V}}_i \triangleq \mathbf{X}_i \mathbf{V}_N \mathbf{X}_i^\top$. Hence we have

$$\begin{aligned} \text{L}(\psi) &\geq -\frac{1}{2} [\text{tr}(\mathbf{V}_N \mathbf{V}_0^{-1}) - \log |\mathbf{V}_N \mathbf{V}_0^{-1}| + (\mathbf{m}_N - \mathbf{m}_0)^\top \mathbf{V}_0^{-1} (\mathbf{m}_N - \mathbf{m}_0)] \\ &\quad - \frac{1}{2} DM + \sum_{i=1}^N \mathbf{y}_i^\top \tilde{\mathbf{m}}_i - \frac{1}{2} \text{tr}(\mathbf{A}_i \tilde{\mathbf{V}}_i) - \frac{1}{2} \tilde{\mathbf{m}}_i^\top \mathbf{A}_i \tilde{\mathbf{m}}_i + \mathbf{b}_i^\top \tilde{\mathbf{m}}_i - c_i \end{aligned} \quad (15.36)$$

We will use coordinate ascent to optimize this lower bound. That is, we update the variational posterior parameters \mathbf{V}_N and \mathbf{m}_N , and then the variational likelihood parameters ψ_i . We leave the detailed derivation as an exercise, and just state the results. We have

$$\mathbf{V}_N = \left(\mathbf{V}_0 + \sum_{i=1}^N \mathbf{X}_i^\top \mathbf{A}_i \mathbf{X}_i \right)^{-1} \quad (15.37)$$

$$\mathbf{m}_N = \mathbf{V}_N \left(\mathbf{V}_0^{-1} \mathbf{m}_0 + \sum_{i=1}^N \mathbf{X}_i^\top (\mathbf{y}_i + \mathbf{b}_i) \right) \quad (15.38)$$

$$\psi_i = \tilde{\mathbf{m}}_i = \mathbf{X}_i \mathbf{m}_N \quad (15.39)$$

We can exploit the fact that \mathbf{A}_i is a constant matrix, plus the fact that \mathbf{X}_i has block structure, to simplify the first two terms as follows:

$$\mathbf{V}_N = \left(\mathbf{V}_0 + \mathbf{A} \otimes \sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^\top \right)^{-1} \quad (15.40)$$

$$\mathbf{m}_N = \mathbf{V}_N \left(\mathbf{V}_0^{-1} \mathbf{m}_0 + \sum_{i=1}^N (\mathbf{y}_i + \mathbf{b}_i) \otimes \mathbf{x}_i \right) \quad (15.41)$$

where \otimes denotes the kronecker product.

1
2 **15.1.2.2 Bohning's bound in the binary case**

3 If we have binary data, then $y_i \in \{0, 1\}$, $M = 1$ and $\eta_i = \mathbf{w}^\top \mathbf{x}_i$ where $\mathbf{w} \in \mathbb{R}^D$ is a weight vector
4 (not matrix). In this case, the Bohning bound becomes
5

6
$$\log(1 + e^\eta) \leq \frac{1}{2}a\eta^2 - b\eta + c \quad (15.42)$$

7

8
$$a = \frac{1}{4} \quad (15.43)$$

9

10
$$b = a\psi - (1 + e^{-\psi})^{-1} \quad (15.44)$$

11

12
$$c = \frac{1}{2}a\psi^2 - (1 + e^{-\psi})^{-1}\psi + \log(1 + e^\psi) \quad (15.45)$$

13

14 It is possible to derive an alternative quadratic bound for this case. as shown in Section 6.3.4.2.
15 This has the following form
16

17
$$\log(1 + e^\eta) \leq \lambda(\psi)(\eta^2 - \psi^2) + \frac{1}{2}(\eta - \psi) + \log(1 + e^\psi) \quad (15.46)$$

18

19
$$\lambda(\psi) \triangleq \frac{1}{4\psi} \tanh(\psi/2) = \frac{1}{2\psi} \left[\sigma(\psi) - \frac{1}{2} \right] \quad (15.47)$$

20

22 To facilitate comparison with Bohning's bound, let us rewrite the JJ bound as a quadratic form as
23 follows
24

25
$$\log(1 + e^\eta) \leq \frac{1}{2}a(\psi)\eta^2 - b(\psi)\eta + c(\psi) \quad (15.48)$$

26

27
$$a(\psi) = 2\lambda(\psi) \quad (15.49)$$

28
$$b(\psi) = -\frac{1}{2} \quad (15.50)$$

29

30
$$c(\psi) = -\lambda(\psi)\psi^2 - \frac{1}{2}\psi + \log(1 + e^\psi) \quad (15.51)$$

31

32 The JJ bound has an adaptive curvature term, since a depends on ψ . In addition, it is tight at two
33 points, as is evident from Figure 15.1(a). By contrast, the Bohning bound is a constant curvature
34 bound, and is only tight at one point, as is evident from Figure 15.1(b). Nevertheless, the Bohning
35 bound is simpler, and somewhat faster to compute, since \mathbf{V}_N is a constant, independent of the
36 variational parameters Ψ .
37

38

39 **15.1.2.3 Other bounds**
40

41 It is possible to devise bounds that are even more accurate than the JJ bound, and which work for
42 the multiclass case, by using a piecewise quadratic upper bound to lse, as described in [Marlin11].
43 By increasing the number of pieces, the bound can be made arbitrarily tight.

44 It is also possible to come up with approximations that are not bounds. For example, [Shekhovtsov2019]
45 gives a simple approximation for the output of a softmax layer when applied to a stochastic input
46 (characterized in terms of its first two moments).

47

15.2 Converting multinomial logistic regression to Poisson regression

It is possible to represent a multinomial logistic regression model with K outputs as K separate Poisson regression models. (Although the Poisson models are fit separately, they are implicitly coupled, since the counts must sum to N_n across all K outcomes.) This fact can enable more efficient training when the number of categories is large [Taddy2015].

To see why this relationship is true, we follow the presentation of [rethinking2]. We assume $K = 2$ for notational brevity (i.e., binomial regression). Assume we have m trials, with counts y_1 and y_2 of each outcome type. The multinomial likelihood has the form

$$p(y_1, y_2 | m, \mu_1, \mu_2) = \frac{m!}{y_1! y_2!} \mu_1^{y_1} \mu_2^{y_2} \quad (15.52)$$

Now consider a product of two Poisson likelihoods, for each set of counts:

$$p(y_1, y_2 | \lambda_1, \lambda_2) = p(y_1 | \lambda_1) p(y_2 | \lambda_2) = \frac{e^{-\lambda_1} \lambda_1^{y_1}}{y_1!} \frac{e^{-\lambda_2} \lambda_2^{y_2}}{y_2!} \quad (15.53)$$

We now show that these are equivalent, under a suitable setting of the parameters.

Let $\Lambda = \lambda_1 + \lambda_2$ be the expected total number of counts of any type, $\mu_1 = \lambda_1 / \Lambda$ and $\mu_2 = \lambda_2 / \Lambda$. Substituting into the binomial likelihood gives

$$p(y_1, y_2 | m, \mu_1, \mu_2) = \frac{m!}{y_1! y_2!} \left(\frac{\lambda_1}{\Lambda} \right)^{y_1} \left(\frac{\lambda_2}{\Lambda} \right)^{y_2} = \frac{m!}{\Lambda^{y_1} \Lambda^{y_2}} \frac{\lambda_1^{y_1} \lambda_2^{y_2}}{y_1! y_2!} \quad (15.54)$$

$$= \frac{m!}{\Lambda^m} \frac{e^{-\lambda_1}}{e^{-\lambda_1}} \frac{\lambda_1^{y_1}}{y_1!} \frac{e^{-\lambda_2}}{e^{-\lambda_2}} \frac{\lambda_2^{y_2}}{y_2!} \quad (15.55)$$

$$= \underbrace{\frac{m!}{e^{-\Lambda} \Lambda^m}}_{p(m)^{-1}} \underbrace{\frac{e^{-\lambda_1} \lambda_1^{y_1}}{y_1!}}_{p(y_1)} \underbrace{\frac{e^{-\lambda_2} \lambda_2^{y_2}}{y_2!}}_{p(y_2)} \quad (15.56)$$

The final expression says that $p(y_1, y_2 | m) = p(y_1)p(y_2)/p(m)$, which makes sense.

15.2.1 Beta-binomial logistic regression

In some cases, there is more variability in the observed counts than we might expect from just a binomial model, even after taking into account the observed predictors. This is called **over-dispersion**, and is usually due to unobserved factors that are omitted from the model. In such cases, we can use a **beta-binomial** model instead of a binomial model:

$$y_i \sim \text{BetaBinom}(m_i, \alpha_i, \beta_i) \quad (15.57)$$

$$\alpha_i = \pi_i \kappa \quad (15.58)$$

$$\beta_i = (1 - \pi_i) \kappa \quad (15.59)$$

$$\pi_i = \sigma(\mathbf{w}^\top \mathbf{x}_i) \quad (15.60)$$

Note that we have parameterized the model in terms of its mean rate,

$$\pi_i = \frac{\alpha_i}{\alpha_i + \beta_i} \quad (15.61)$$

1
2 and shape,

3
4 $\kappa_i = \alpha_i + \beta_i$ (15.62)

5 We choose to make the mean depend on the inputs (covariates), but to treat the shape (which is like
6 a precision term) as a shared constant.

7 As we discuss in [book1], the beta-binomial distribution as a continuous mixture distribution of
8 the following form:
9

10 $\text{BetaBinom}(y|m, \alpha, \beta) = \int \text{Bin}(y|m, \mu) \text{Beta}(\mu|\alpha, \beta) d\mu$ (15.63)

12 In the regression context, we can interpret this as follows: rather than just predicting the mean
13 directly, we predict the mean and variance. This allows for each individual example to have more
14 variability than we might otherwise expect.
15

16 If the shape parameter κ is less than 2, then the distribution is an inverted U-shape which strongly
17 favors probabilities of 0 or 1 (see Main Figure 2.3b). We generally want to avoid this, which we can
18 do by ensuring $\kappa > 2$.

19 Following [rethinking2], let us use this model to reanalyze the Berkeley admissions data from ??.
20 We saw that there was a lot of variability in the outcomes, due to the different admissions rates of each
21 department. Suppose we just regress on the gender, i.e., $\mathbf{x}_i = (\mathbb{I}(\text{GENDER}_i = 1), \mathbb{I}(\text{GENDER}_i = 2))$,
22 and $\mathbf{w} = (\alpha_1, \alpha_2)$ are the corresponding logits. If we use a binomial regression model, we can be
23 misled into thinking there is gender bias. But if we use the more robust beta-binomial model, we
24 avoid this false conclusion, as we show below.

25 We fit the following model:

26 $A_i \sim \text{BetaBinom}(N_i, \pi_i, \kappa)$ (15.64)

27 $\text{logit}(\pi_i) = \alpha_{\text{GENDER}[i]}$ (15.65)

29 $\alpha_j \sim \mathcal{N}(0, 1.5)$ (15.66)

30 $\kappa = \phi + 2$ (15.67)

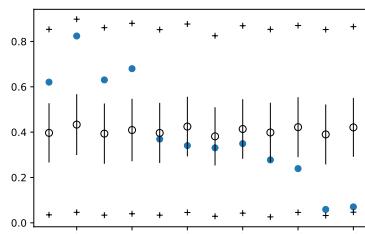
31 $\phi \sim \text{Expon}(1)$ (15.68)

33 (To ensure that $\kappa > 2$, we use a trick and define it as $\kappa = \phi + 2$, where we put an exponential prior
34 (which has a lower bound of 0) on ϕ .)

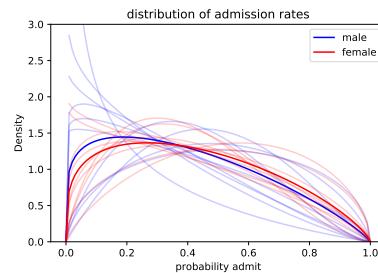
35 We fit this model (using HMC) and plot the results in Figure 15.2. In Figure 15.2a, we show the
36 posterior predictive distribution; we see that is quite broad, so the model is no longer overconfident.
37 In Figure 15.2b, we plot $p(\sigma(\alpha_j)|\mathcal{D})$, which is the posterior over the rate of admissions for men and
38 women. We see that there is considerable uncertainty in these value, so now we avoid the false
39 conclusion that one is significantly higher than the other. However, the model is so vague in its
40 predictions as to be useless. In Section 15.2.3, we fix this problem by using a multi-level logistic
41 regression model.
42

43 15.2.2 Poisson regression

45 Let us revisit the Berkeley admissions example from ?? using Poisson regression. We use a simplified
46 form of the model, in which we just model the outcome counts without using any features, such as
47



(a)



(b)

Figure 15.2: Results of fitting beta-binomial regression model to Berkeley admissions data. (b) Posterior predictive distribution (black) superimposed on empirical data (blue). The hollow circle is the posterior predicted mean acceptance rate, $\mathbb{E}[A_i|\mathcal{D}]$; the vertical lines are 1 standard deviation around this mean, $\text{std}[A_i|\mathcal{D}]$; the + signs indicate the 89% predictive interval. (b) Samples from the posterior distribution for the admissions rate for men (blue) and women (red). Thick curve is posterior mean. Adapted from Figure 12.1 of [rethinking2]. Generated by [logreg_ucb_admissions_numpyro.ipynb](#).

gender or department. That is, the model has the form

$$y_{j,n} \sim \text{Poi}(\lambda_j) \quad (15.69)$$

$$\lambda_j = e^{\alpha_j} \quad (15.70)$$

$$\alpha_j \sim \mathcal{N}(0, 1.5) \quad (15.71)$$

for $j = 1 : 2$ and $n = 1 : 12$. Let $\bar{\lambda}_i = \mathbb{E}[\lambda_i|\mathcal{D}_i]$, where $\mathcal{D}_1 = \mathbf{y}_{1,1:N}$ is the vector of admission counts, and $\mathcal{D}_2 = \mathbf{y}_{2,1:N}$ is the vector of rejection counts (so $m_n = y_{1,n} + y_{2,n}$ is the total number of applications for case n). The expected acceptance rate across the entire dataset is

$$\frac{\bar{\lambda}_1}{\bar{\lambda}_1 + \bar{\lambda}_2} = \frac{146.2}{146.2 + 230.9} = 0.38 \quad (15.72)$$

Let us compare this to a binomial regression model of the form

$$y_n \sim \text{Bin}(m_n, \mu) \quad (15.73)$$

$$\mu = \sigma(\alpha) \quad (15.74)$$

$$\alpha \sim \mathcal{N}(0, 1.5) \quad (15.75)$$

Let $\bar{\alpha} = \mathbb{E}[\alpha|\mathcal{D}]$, where $\mathcal{D} = (\mathbf{y}_{1,1:N}, \mathbf{m}_{1:N})$. The expected acceptance rate across the entire dataset is $\sigma(\bar{\alpha}) = 0.38$, which matches Equation (15.72). (See [logreg_ucb_admissions_numpyro.ipynb](#) for the code.)

15.2.3 GLMM (hierarchical Bayes) regression

Let us revisit the Berkeley admissions dataset from ??, where there are 12 examples, corresponding to male and female admissions to 6 departments. Thus the data is grouped both by gender and

1 department. Recall that A_i is the number of students admitted in example i , N_i is the number
 2 of applicants, μ_i is the expected rate of admissions (the variable of interest), and $\text{DEPT}[i]$ is the
 3 department (6 possible values). For pedagogical reasons, we replace the categorical variable $\text{GENDER}[i]$
 4 with the binary indicator $\text{MALE}[i]$. We can create a model with **varying intercept** and **varying**
 5 **slope** as follows:

$$A_i \sim \text{Bin}(N_i, \mu_i) \quad (15.76)$$

$$\text{logit}(\mu_i) = \alpha_{\text{DEPT}[i]} + \beta_{\text{DEPT}[i]} \times \text{MALE}[i] \quad (15.77)$$

10 This has 12 parameters, as does the original formulation in ???. However, these are not independent
 11 degrees of freedom. In particular, the intercept and slope are correlated, as we see in ?? (higher
 12 admissions means steeper slope). We can capture this using the following prior:

$$(\alpha_j, \beta_j) \sim \mathcal{N}\left(\begin{pmatrix} \bar{\alpha} \\ \bar{\beta} \end{pmatrix}, \Sigma\right) \quad (15.78)$$

$$\bar{\alpha} \sim \mathcal{N}(0, 4) \quad (15.79)$$

$$\bar{\beta} \sim \mathcal{N}(0, 1) \quad (15.80)$$

$$\Sigma = \text{diag}(\sigma) \mathbf{R} \text{diag}(\sigma) \quad (15.81)$$

$$\mathbf{R} \sim \text{LKJ}(2) \quad (15.82)$$

$$\sigma \sim \prod_{d=1}^2 \mathcal{N}_+(\sigma_d | 0, 1) \quad (15.83)$$

24 We can write this more compactly in the following way.¹ We define $\mathbf{u} = (\bar{\alpha}, \bar{\beta})$, and $\mathbf{w}_j = (\alpha_j, \beta_j)$,
 25 and then use this model:

$$\log(\mu_i) = w_{\text{DEPT}[i]}[0] + w_{\text{DEPT}[i]}[1] \times \text{MALE}[i] \quad (15.84)$$

$$\mathbf{w}_j \sim \mathcal{N}(\mathbf{u}, \Sigma) \quad (15.85)$$

$$\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \text{diag}(4, 1)) \quad (15.86)$$

31 See Figure 15.3(a) for the graphical model.

32 Following the discussion in Main Section 12.6.5, it is advisable to rewrite the model in a non-centered
 33 form. Thus we write

$$\mathbf{w}_j = \mathbf{u} + \sigma \mathbf{L} \mathbf{z}_j \quad (15.87)$$

36 where $\mathbf{L} = \text{chol}(\mathbf{R})$ is the Cholesky factor for the correlation matrix \mathbf{R} , and $\mathbf{z}_j \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_2)$. Thus the
 37 model becomes the following:²

$$\mathbf{z}_j \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_2) \quad (15.88)$$

$$\mathbf{v}_j = \text{diag}(\sigma) \mathbf{L} \mathbf{z}_j \quad (15.89)$$

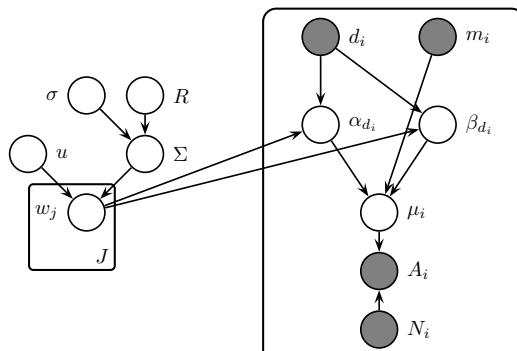
$$\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \text{diag}(4, 1)) \quad (15.90)$$

$$\log(\mu_i) = u[0] + v[\text{DEPT}[i], 0] + (u[1] + v[\text{DEPT}[i], 1]) \times \text{MALE}[i] \quad (15.91)$$

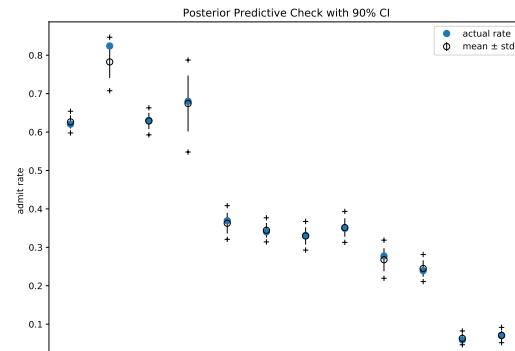
44 1. In <https://bit.ly/3mP1QWH>, this is referred to as glmm4. Note that we use \mathbf{w} instead of \mathbf{v} , and we use \mathbf{u} instead
 45 of \mathbf{v}_μ .

46 2. In <https://bit.ly/3mP1QWH>, this is referred to as glmm5.

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



(a)



(b)

Figure 15.3: (a) Generalized linear mixed model for inputs d_i (department) and m_i (male), and output A_i (number of admissions), given N_i (number of applicants). (b) Results of fitting this model to the UCB dataset. Generated by `logreg ucb admissions numpyro.ipynb`.

²¹ This is the version of the model that is implemented in the numypro code.

The results of fitting this model are shown in Figure 15.3(b). The fit is slightly better than in ??, especially for the second column (females in department 2), where the observed value is now inside the predictive interval.

39
40
41
42
43
44
45
46
47

16 Deep neural networks

16.1 More canonical examples of neural networks

16.1.1 Transformers

The high level structure is shown in Figure 16.1. We explain the encoder and decoder below.

16.1.1.1 Encoder

The details of the transformer encoder block are shown in Figure 16.2. The embedded input tokens \mathbf{X} are passed through an attention layer (typically multi-headed), and the output \mathbf{Z} is added to the input \mathbf{X} using a residual connection. More precisely, if the input $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ for $\mathbf{x}_i \in \mathbb{R}^d$, we compute the following [Yun2020iclr]:

$$\mathbf{x}_i = \mathbf{x}_i + \sum_{j=1}^n K_{ij} \mathbf{W}_V \mathbf{x}_j \quad (16.1)$$

where $\mathbf{K} = \text{softmax}(\mathbf{A})$, and $A_{ij} = (\mathbf{W}_Q \mathbf{x}_i)^\top (\mathbf{W}_K \mathbf{x}_j)$. (In [Sander2022], they explore a variant of this, known as **sinkformer**, where they use Sinkhorn’s algorithm to ensure \mathbf{K} is stocchastically normalized across columns as well as rows.) The output of self attention is then passed into a layer normalization layer, which normalizes and learns an affine transformation for each dimension, to ensure all hidden units have comparable magnitude. (This is necessary because the attention masks might upweight just a few locations, resulting in a skewed distribution of values.) Then the output vectors at each location are mapped through an MLP, composed of 1 linear layer, a skip connection and a normalization layer.

The overall encoder is N copies of this encoder block. The result is an encoding $\mathbf{H}_x \in \mathbb{R}^{T_x \times D}$ of the input, where T_x is the number of input tokens, and D is the dimensionality of the attention vectors.

16.1.1.2 Decoder

Once the input has been encoded, the output is generated by the decoder. The first part of the decoder is the decoder attention block, that attends to all previously generated tokens, $\mathbf{y}_{1:t-1}$, and computes the encoding $\mathbf{H}_y \in \mathbb{R}^{T_y \times D}$. This block uses masked attention, so that output t can only attend to locations prior to t in \mathbf{Y} .

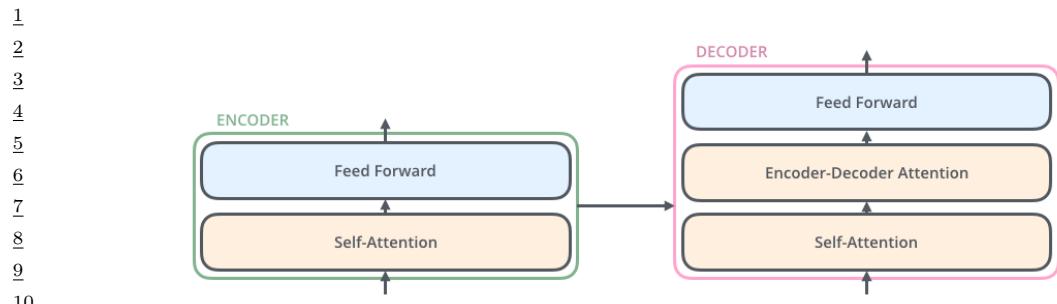


Figure 16.1: High level structure of the encoder-decoder transformer architecture. From <https://jalammar.github.io/illustrated-transformer/>. Used with kind permission of Jay Alammar.

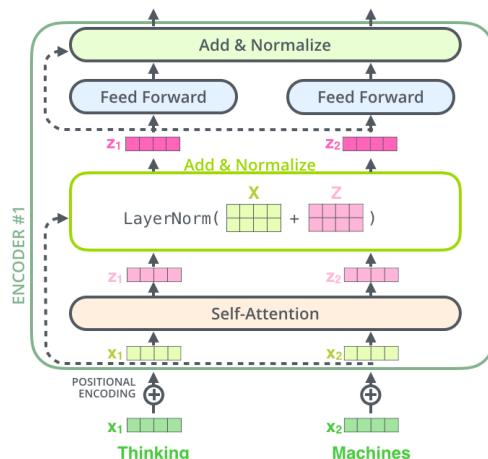


Figure 16.2: The encoder block of a transformer for two input tokens. From <https://jalammar.github.io/illustrated-transformer/>. Used with kind permission of Jay Alammar.

The second part of the decoder is the encoder-decoder attention block, that attends to both the encoding of the input, \mathbf{H}_x , and the previously generated outputs, \mathbf{H}_y . These are combined to compute $\mathbf{Z} = \text{Attn}(\mathbf{Q} = \mathbf{H}_y, \mathbf{K} = \mathbf{H}_x, \mathbf{V} = \mathbf{H}_x)$, which compares the output to the input. The joint encoding of the state \mathbf{Z} is then passed through an MLP layer. The full decoder repeats this decoder block N times.

At the end of the decoder, the final output is mapped to a sequence of T_y output logits via a final linear layer.

41

42 16.1.1.3 Putting it all together

We can combine the encoder and decoder as shown in Figure 16.3. There is one more detail we need to discuss. This concerns the fact that the attention operation pools information across all locations, so the transformer is invariant to the ordering of the inputs. To overcome this, it is standard to add

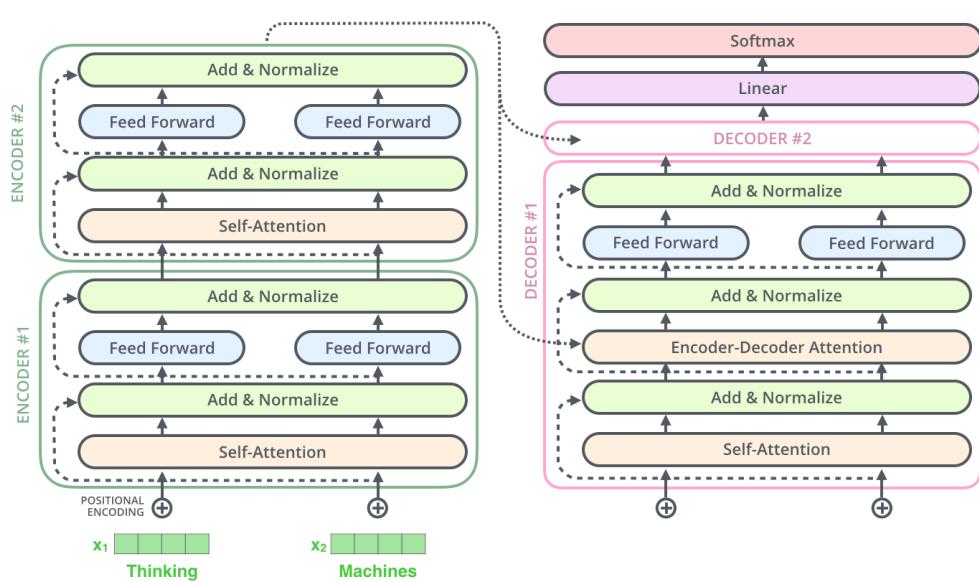


Figure 16.3: A transformer model where we use 2 encoder blocks and 2 decoder blocks. (The second decoder block is not expanded.) We assume there are 2 input and 2 output tokens. From <https://jalammar.github.io/illustrated-transformer/>. Used with kind permission of Jay Alammar.

positional encoding vectors to the input tokens $\mathbf{x} \in \mathbb{R}^{T_x \times D}$. That is, we replace \mathbf{x} with $\mathbf{x} + \mathbf{u}$, where $\mathbf{u} \in \mathbb{R}^{T_x \times D}$ is a (possibly learned) vector, where \mathbf{u}_i is some encoding of the fact that \mathbf{x}_i comes from the i 'th location in the N -dimensional sequence.

16.1.2 Graph neural networks (GNNs)

In this section, we discuss **graph neural networks** or **GNNs**. Our presentation is based on [Sanchez-lengeling2021], which in turn is a summary of the **message passing neural network** framework of [gilmer2017neural] and the **Graph Nets** framework of [battaglia2018relational].

We assume the graph is represent as a set of N nodes or vertices, each associated with a feature vector to create the matrix $\mathbf{V} \in \mathbb{R}^{N \times D_v}$; a set of E edges, each associated with a feature vector to create the matrix $\mathbf{E} \in \mathbb{R}^{E \times D_e}$; and a global feature vector $\mathbf{u} \in \mathbb{R}^{D_u}$, representing overall properties of the graph, such as its size. (We can think of \mathbf{u} as the features associated with a global or master node.) The topology of the graph can be represented as an $N \times N$ **adjacency matrix**, but since this is usually very sparse (see Figure 16.4 for an example), a more compact representation is just to store the list of edges in an **adjacency list** (see Figure 16.5 for an example).

16.1.2.1 Basics of GNNs

A GNN adopts a “graph in, graph out” philosophy, similar to how transformers map from sequences to sequences. A basic GNN layer updates the embedding vectors associated with the nodes, edges

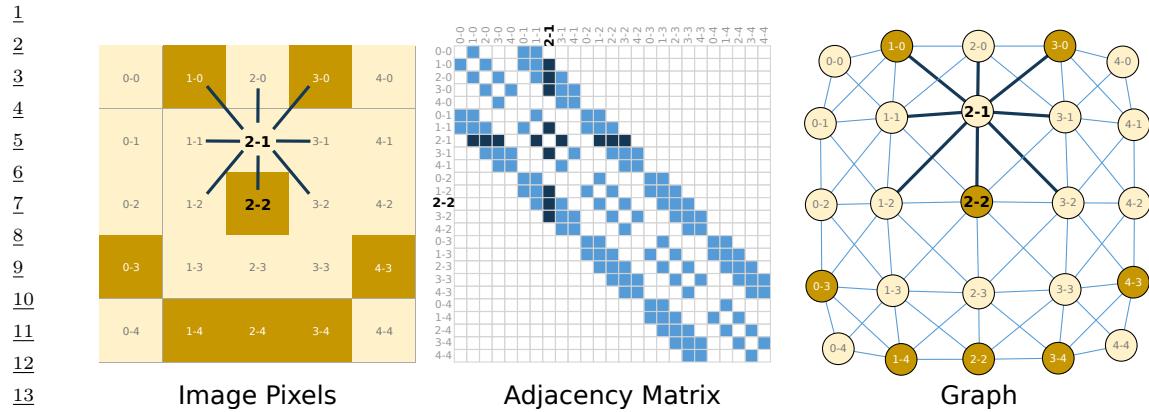


Figure 16.4: Left: Illustration of a 5×5 image, where each pixel is either off (light yellow) or on (dark yellow). Each non-border pixel has 8 nearest neighbors. We highlight the node at location $(2,1)$, where the top-left is $(0,0)$. Middle: The corresponding adjacency matrix, which is sparse and banded. Right: Visualization of the graph structure. Dark nodes correspond to pixels that are on, light nodes correspond to pixels that are off. Dark edges correspond to the neighbors of the node at $(2,1)$. From [Sanchez-lengeling2021]. Used with kind permission of Benjamin Sanchez-Lengeling.

20

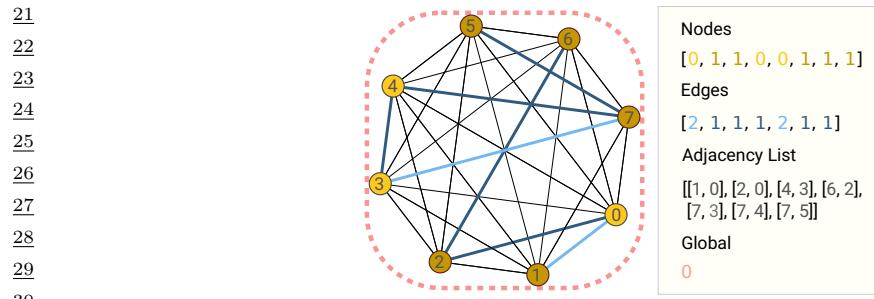


Figure 16.5: A simple graph where each node has 2 types (0=light yellow, 1=dark yellow), each edge has 2 types (1=gray, 2=blue), and the global feature vector is a constant (0=red). We represent the topology using an adjacency list. From [Sanchez-lengeling2021]. Used with kind permission of Benjamin Sanchez-Lengeling.

34

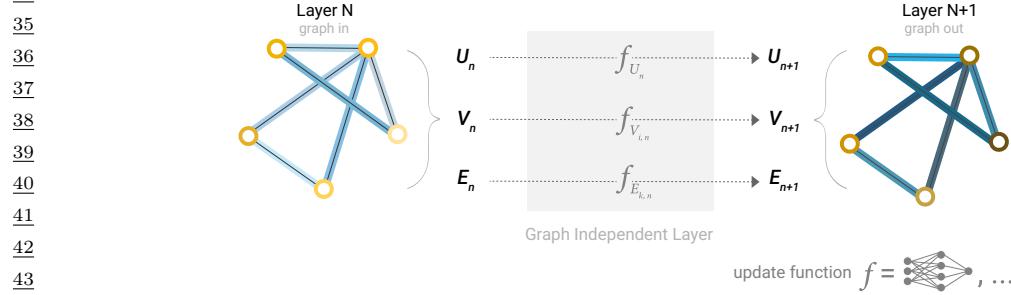


Figure 16.6: A basic GNN layer. We update the embedding vectors \mathbf{U} , \mathbf{V} and \mathbf{V} using the global, node and edge functions f . From [Sanchez-lengeling2021]. Used with kind permission of Benjamin Sanchez-Lengeling.

47

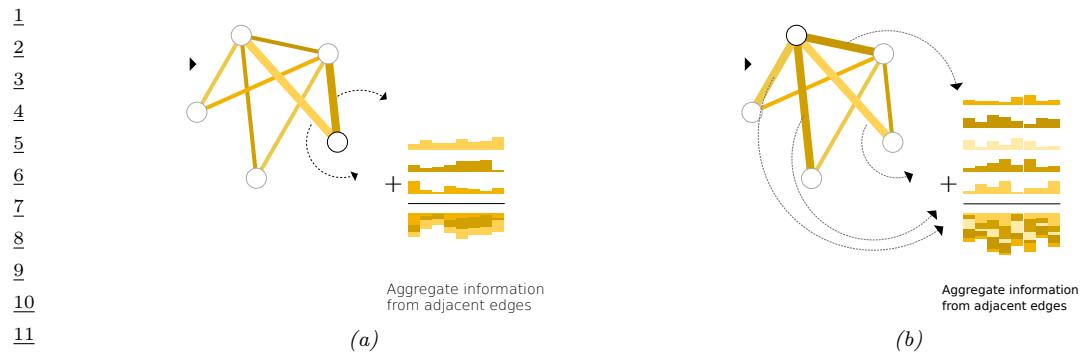


Figure 16.7: Aggregating edge information into two different nodes. From [Sanchez-lengeling2021]. Used with kind permission of Benjamin Sanchez-Lengeling.

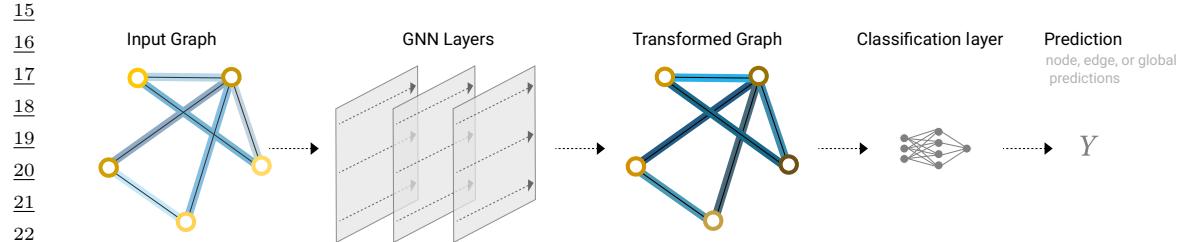


Figure 16.8: An end-to-end GNN classifier. From [Sanchez-lengeling2021]. Used with kind permission of Benjamin Sanchez-Lengeling.

and whole graph, as illustrated in Figure 16.6. The update functions are typically simple MLPs, that are applied independently to each embedding vector.

To leverage the graph structure, we can combine information using a **pooling** operation. That is, for each node n , we extract the feature vectors associated with its edges, and combine it with its local feature vector using a permutation invariant operation such as summation or averaging. See Figure 16.7 for an illustration. We denote this pooling operation by $\rho_{E_n \rightarrow V_n}$. We can similarly pool from nodes to edges, $\rho_{V_n \rightarrow E_n}$, or from nodes to globals, $\rho_{V_n \rightarrow U_n}$, etc.

The overall GNN is composed of GNN layers and pooling layers. At the end of the network, we can use the final embeddings to classify nodes, edges, or the whole graph. See Figure 16.8 for an illustration.

16.1.2.2 Message passing

Instead of transforming each vector independently and then pooling, we can first pool the information for each node (or edge) and then update its vector representation. That is, for node i , we **gather** information from all neighboring nodes, $\{h_j : j \in nbr(i)\}$; we **aggregate** these vectors with the local vector using an operation such as sum; and then we compute the new state using an **update**

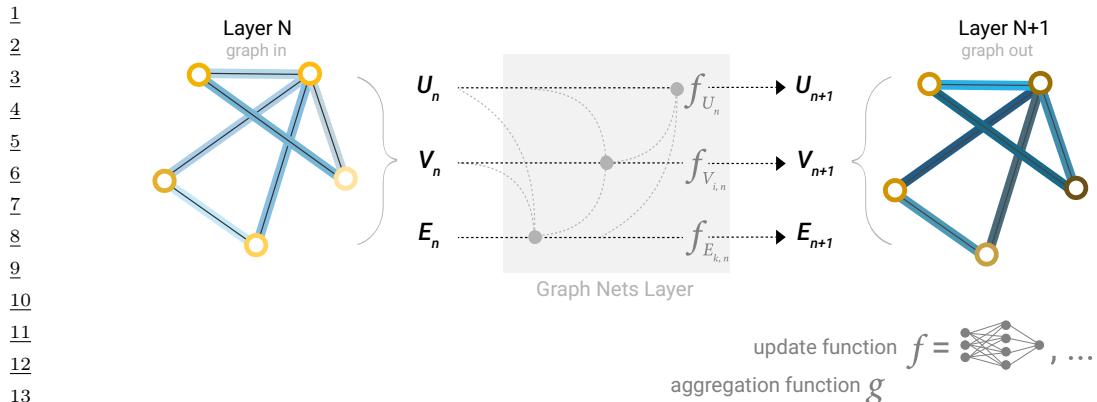


Figure 16.9: Message passing in one layer of a GNN. First the global node U_n and the local nodes V_n send messages to the edges E_n , which get updated to give E_{n+1} . Then the nodes get updated to give V_{n+1} . Finally the global node gets updated to give U_{n+1} . From [Sanchez-lengeling2021]. Used with kind permission of Benjamin Sanchez-Lengeling.

function, such as

$$\mathbf{h}'_i = \text{ReLU}(\mathbf{U}\mathbf{h}_i + \sum_{j \in \text{nbr}(i)} \mathbf{V}\mathbf{h}_j) \quad (16.2)$$

See Figure 16.11a for a visualization.

The above operation can be viewed as a form of “**message passing**”, in which the values of neighboring nodes \mathbf{h}_j are sent to node i and then combined. It is more general than belief propagation (Main Section 9.3), since the messages are not restricted to represent probability distributions (see Main Section 9.4.10 for more discussion).

After K message passing layers, each node will have received information from neighbors which are K steps away in the graph. This can be “short circuited” by sending messages through the global node, which acts as a kind of bottleneck. See Figure 16.9 for an illustration.

16.1.2.3 More complex types of graphs

We can easily generalize this framework to handle other graph types. For example, **multigraphs** have multiple edge types between each pair of nodes. For example, in a **knowledge graph**, we might have edge types “spouse-of”, “employed-by” or “born-in”. See Figure 16.10(left) for an example. In **hypergraphs**, each edge may connect more than two nodes. For example, in a knowledge graph, we might want to specify the three-way relation “parents-of(c, m, f)”, for child c , mother m and father f . We can “reify” such hyperedges into hypernodes, as shown in Figure 16.10(right).

16.1.2.4 Graph attention networks

When performing message passing, we can generalize the linear combination used in Equation (16.2) to use a weighted combination instead, where the weights are computed an attention mecha-

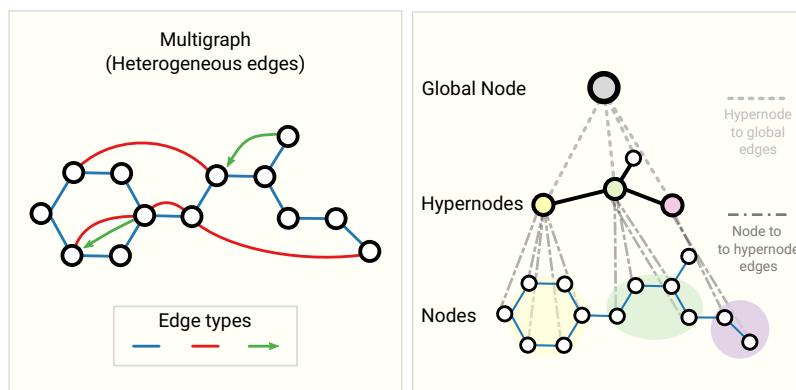


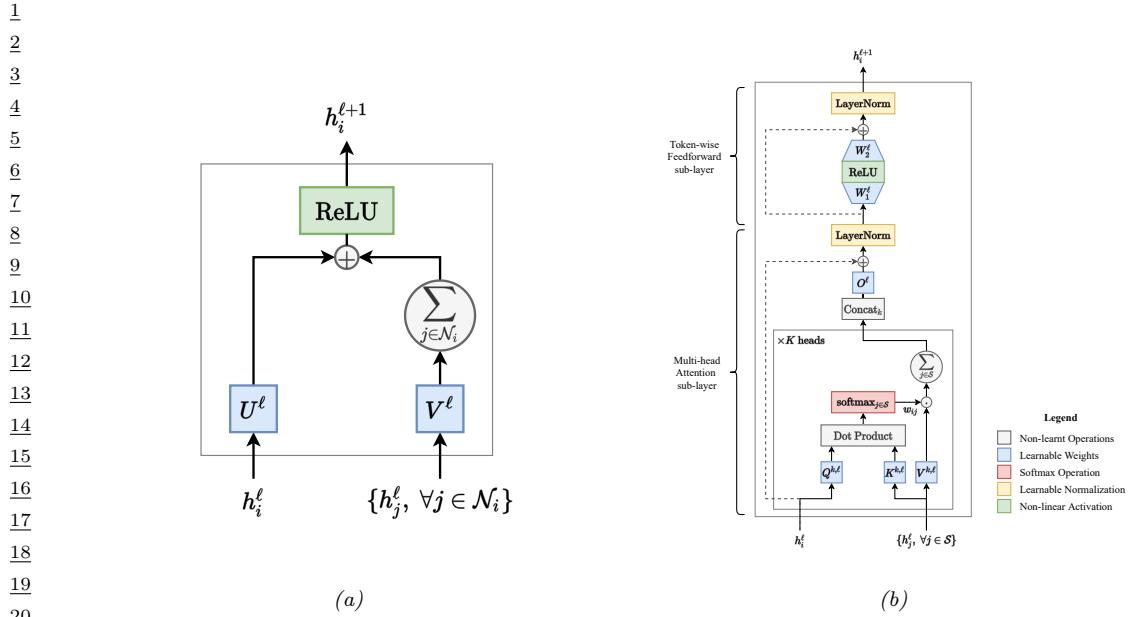
Figure 16.10: Left: a multigraph can have different edge types. Right: a hypergraph can have edges which connect multiple nodes. From [Sanchez-lengeling2021]. Used with kind permission of Benjamin Sanchez-Lengeling.

nism (Main Section 16.2.7). The resulting model is called a **graph attention network** or **GAT** [velickovic2017graph]. This allows the effective topology of the graph to be context dependent.

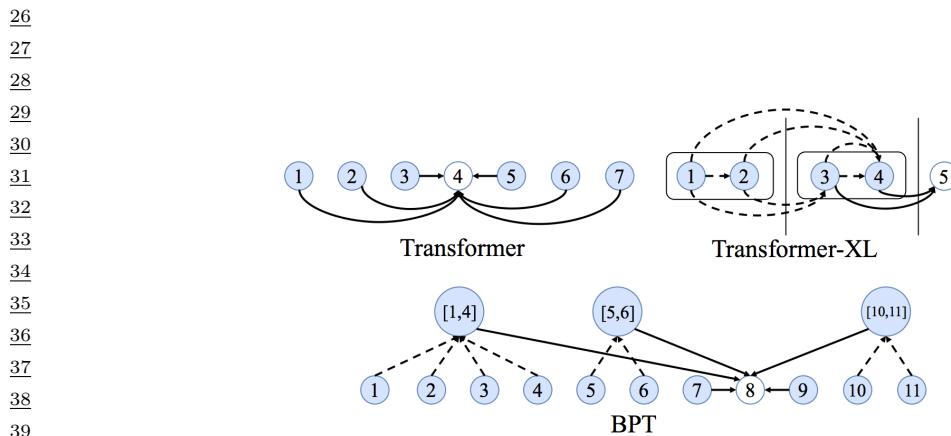
16.1.2.5 Transformers are fully connected GNNs

Suppose we create a fully connected graph in which each node represents a word in a sentence. Let us use this to construct a GNN composed of GAT layers, where we use multi-headed scaled dot product attention. Suppose we combine each GAT block with layer normalization and an MLP. The resulting block is shown in Figure 16.11b. We see that this is identical to the transformer encoder block shown in Figure 16.2. This construction shows that transformers are just a special case of GNNs [Joshi2020].

The advantage of this observation is that it naturally suggests ways to overcome the $O(N^2)$ complexity of transformers. For example, in **Transformer-XL** [transformer-xl], we create blocks of nodes, and connect these together, as shown in Figure 16.12(top right). In **binary partition transformer** or **BPT** [BPT], we also create blocks of nodes, but add them as virtual ‘‘hypernodes’’, as shown in Figure 16.12(bottom). There are many other approaches to reducing the $O(N^2)$ cost (see e.g., [book1]), but the GNN perspective is a helpful one.



21 *Figure 16.11: (a) A graph neural network aggregation block. Here h_i^ℓ is the hidden representation for node i
22 in layer ℓ , and $\mathcal{N}(i)$ are i's neighbors. The output is given by $h_i^{\ell+1} = \text{ReLU}(\mathbf{U}^\ell h_i + \sum_{j \in \text{nbr}(i)} \mathbf{V}^\ell h_j^\ell)$. (b) A
23 transformer encoder block. Here h_i^ℓ is the hidden representation for word i in layer ℓ , and \mathcal{S} are all the words
24 in the sentence. The output is given by $h_i^{\ell+1} = \text{Attn}(\mathbf{Q}^\ell h_i^\ell, \{\mathbf{K}^\ell h_j, \mathbf{V}^\ell h_j^\ell\})$. From [Joshi2020]. Used with
25 kind permission of Chaitanya Joshi.*



41 *Figure 16.12: Graph connectivity for different types of transformer. Top left: in a vanilla Transformer,
42 every node is connected to every other node. Top right: in Transformer-XL, nodes are grouped into blocks.
43 Bottom: in BPT, we use a binary partitioning of the graph to create virtual node clusters. From <https://graphdeeplearning.github.io/post/transfomers-are-gnns/>. Used with kind permission of Chaitanya
44 Joshi.*

17 Bayesian neural networks

17.1 More details on EKF for training MLPs

The suggestion to use the EKF to train MLPs was first made in [Singhal1988]. We give a summary below, based on [Puskorius2003].

17.1.1 Global EKF

On the left, we use notation from [Puskorius2003], and on the right we use our notation.

$$\hat{\mathbf{w}}_k = \boldsymbol{\mu}_{k|k-1} \quad (17.1)$$

$$\mathbf{P}_k = \boldsymbol{\Sigma}_{k|k-1} \quad (17.2)$$

$$\mathbf{H}_k = \text{Jac}(\mathbf{h})(\boldsymbol{\mu}_{k|k-1}) \quad (17.3)$$

$$\mathbf{A}_k = \mathbf{S}_k^{-1} = (\mathbf{R}_k + \mathbf{H}_k \boldsymbol{\Sigma}_{k|t-1} \mathbf{H}_k^\top)^{-1} \quad (17.4)$$

$$\mathbf{K}_k = \boldsymbol{\Sigma}_{k|k-1} \mathbf{H}_k^\top \mathbf{S}_k^{-1} \quad (17.5)$$

$$\boldsymbol{\xi}_k = \mathbf{y}_k - \mathbf{h}(\boldsymbol{\mu}_{k|k-1}) \quad (17.6)$$

$$\hat{\mathbf{w}}_{k+1} = \boldsymbol{\mu}_{k+1|k} = \boldsymbol{\mu}_{k|k} = \boldsymbol{\mu}_{k|k-1} + \mathbf{K}_k \boldsymbol{\xi}_k \quad (17.7)$$

$$\mathbf{P}_{k+1} = \boldsymbol{\Sigma}_{k+1|k} = \boldsymbol{\Sigma}_{k|k} + \mathbf{Q}_k = \boldsymbol{\Sigma}_{k|k-1} - \mathbf{K}_k \mathbf{H}_k \boldsymbol{\Sigma}_{k|k-1} + \mathbf{Q}_k \quad (17.8)$$

Suppose there are N outputs and M parameters (size of \mathbf{w}_k), so \mathbf{H}_k is $N \times M$. Computing the matrix inversion \mathbf{S}_k^{-1} takes $O(N^3)$ time, computing the matrix multiplication $\mathbf{H}_k^\top \mathbf{S}_k^{-1}$ takes $O(MN^2)$ time, computing the matrix multiplication $\boldsymbol{\Sigma}_{k|k-1}(\mathbf{H}_k^\top \mathbf{S}_k^{-1})$ takes $O(M^2N)$ time, and computing the matrix multiplication $\mathbf{H}_k \boldsymbol{\Sigma}_{k|t-1} \mathbf{H}_k^\top$ takes $O(N^2M + NM^2)$ time. Computing the Jacobian takes $O(NM)$ time, which is N times slower than standard backprop (which uses a scalar output representing the loss). The total time is therefore $O(N^3 + N^2M + NM^2)$. The memory usage is $O(M^2)$.

The learning rate of the algorithm is controlled by the artificial process noise, $\mathbf{Q}_k = q\mathbf{I}$. [Puskorius2003] recommend annealing this from a large to a small value over time, to ensure more rapid convergence. (We should keep $q > 0$ to ensure the posterior is always positive definite.)

17.1.2 Decoupled EKF

We can speed up the method using the “decoupled EKF” [Puskorius1991; Murtuza1994], which partitions the posterior covariance into groups. We give a summary below, based on [Puskorius2003].

1 Suppose there are g groups, and let $\boldsymbol{\mu}_{t|t}^i$ represent the mean of the i 'th group (of size M_i), $\boldsymbol{\Sigma}_{k|k}^i$ its
2 covariance, \mathbf{H}_k^i the Jacobian wrt the i 'th groups weights (of size $N \times M_i$) and \mathbf{K}_k^i the corresponding
3 Kalman gain. Then we have (in our notation)

4

$$\boldsymbol{\mu}_{k|k-1}^i = \boldsymbol{\mu}_{k-1|k-1}^i \quad (17.9)$$

5

$$\boldsymbol{\Sigma}_{k|k-1}^i = \boldsymbol{\Sigma}_{k-1|k-1}^i + \mathbf{Q}_{k-1}^i \quad (17.10)$$

6

$$\mathbf{S}_k = \mathbf{R}_k + \sum_{j=1}^g (\mathbf{H}_k^j)^T \boldsymbol{\Sigma}_{k|k-1}^j \mathbf{H}_k^j \quad (17.11)$$

7

$$\mathbf{K}_k^i = \boldsymbol{\Sigma}_{k|k-1}^i \mathbf{H}_k^i \mathbf{S}_k^{-1} \quad (17.12)$$

8

$$\boldsymbol{\xi}_k = \mathbf{y}_k - \mathbf{h}(\boldsymbol{\mu}_{k|k-1}) \quad (17.13)$$

9

$$\boldsymbol{\mu}_{k|k}^i = \boldsymbol{\mu}_{k|k-1}^i + \mathbf{K}_k^i \boldsymbol{\xi}_k \quad (17.14)$$

10

$$\boldsymbol{\Sigma}_{k|k}^i = \boldsymbol{\Sigma}_{k|k-1}^i - \mathbf{K}_k^i \mathbf{H}_k^i \boldsymbol{\Sigma}_{k|k-1}^i \quad (17.15)$$

11

$$\boldsymbol{\Sigma}_{k|k}^i = \boldsymbol{\Sigma}_{k|k-1}^i - \mathbf{K}_k^i \mathbf{H}_k^i \boldsymbol{\Sigma}_{k|k-1}^i \quad (17.16)$$

12 The time complexity is reduced to $O(N^3 + N^2M + N \sum_{i=1}^g M_i^2)$, and the space complexity
13 is $O(\sum_{i=1}^g M_i^2)$. The term “fully decoupled” refers to a diagonal approximation to the posterior
14 covariance, which is similar in spirit to diagonal pre-conditioning methods such as Adam. The term
15 “node decoupled EKF” refers to a block diagonal approximation, where the blocks correspond to all
16 the weights feeding into a single neuron (since these are highly correlated).

17 In [Puskorius2003], they give a serial scheme for reducing the complexity when N is large (e.g.,
18 multi-class classification). The new time complexity is $O(N^2G + \sum_{i=1}^G M_i^2)$, where G is the number
19 of nodes in the network.

20

21 17.1.3 Mini-batch EKF

22

23 A “multi-stream” (i.e., minibatch) extension was presented in [Feldkamp1994]. As explained in
24 [Puskorius2003], this amounts to stacking N_s observations into a single large observation vector,
25 denoted $\mathbf{y}_{k:l}$, where $l = k + N_s - 1$, and then stacking the Jacobians $\mathbf{H}_{k:l}$. We then perform the
26 update (possibly decoupled) as above. Note that this is more expensive than just averaging gradients,
27 as is done by mini-batch SGD.

28 Minibatch EKF is potentially less accurate than updating after each example, since the linearization
29 is computed at the previous posterior, $\boldsymbol{\mu}_{k-1}$, even for examples at the end of the minibatch, namely
30 at time $l \gg k$. Furthermore, it may be more expensive, due to the need to invert \mathbf{S}_k , which has size
31 $NN_s \times NN_s$. However, it may be less sensitive to the ordering of the data.

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

18 Gaussian processes

18.1 Deep GPs

A **deep Gaussian process** or **DGP** is a composition of GPs [Damianou2013]. (See [Jakkala2021] for a recent survey.) More formally, a DGP of L layers is a hierarchical model of the form

$$\text{DGP}(\mathbf{x}) = f_L \circ \cdots \circ f_1(\mathbf{x}), \quad f_i(\cdot) = [f_i^{(1)}(\cdot), \dots, f_i^{(H_i)}(\cdot)], \quad f_i^{(j)} \sim \text{GP}(0, \mathcal{K}_i(\cdot, \cdot)) \quad (18.1)$$

This is similar to a deep neural network, except the hidden nodes are now hidden functions.

A natural question is: what is gained by this approach compared to a standard GP? Although conventional single-layer GPs are nonparametric, and can model any function (assuming the use of a non-degenerate kernel) with enough data, in practice their performance is limited by the choice of kernel. This can be partially overcome by using a DGP, as we show in Section 18.1.0.2. Unfortunately, posterior inference in DGPs is challenging, as we discuss in Section 18.1.0.3.

In Section 18.1.0.4, we discuss the expressive power of infinitely wide DGPs, and in Section 18.1.0.5 we discuss connections with DNNs.

18.1.0.1 Construction of a deep GP

In this section we give an example of a 2 layer DGP, following the presentation in [Pleiss2021]. Let $f_j^{(j)} \sim \text{GP}(0, \mathcal{K}_j)$ for $j = 1 : H_1$, where H_1 is the number of hidden units, and $f_2 \sim \text{GP}(0, \mathcal{K}_2)$. Assume we have labeled training data $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$ and $\mathbf{y} = (y_1, \dots, y_N)$. Define $\mathbf{F}_1 = [\mathbf{f}_1(\mathbf{x}_1), \dots, \mathbf{f}_1(\mathbf{x}_N)]$ and $\mathbf{f}_2 = [f_2(\mathbf{f}_1(\mathbf{x}_1)), \dots, f_2(\mathbf{f}_1(\mathbf{x}_N))]$. Let \mathbf{x}^* be a test input and define $\mathbf{f}_1^* = \mathbf{f}_1(\mathbf{x}^*)$ and $\mathbf{f}_2^* = f_2(\mathbf{f}_1(\mathbf{x}^*))$. The corresponding joint distribution over all the random variables is given by

$$p(f_2^*, \mathbf{f}_2, \mathbf{F}_1, \mathbf{f}_1, \mathbf{y}) = p(f_2^* | \mathbf{f}_2, \mathbf{f}_1^*, \mathbf{F}_1)p(\mathbf{f}_2 | \mathbf{F}_1, \mathbf{f}_1^*)p(\mathbf{f}_1^*, \mathbf{F}_1)p(\mathbf{y} | \mathbf{f}_2) \quad (18.2)$$

where we drop the dependence on \mathbf{X} and \mathbf{x}^* for brevity. This is illustrated by the graphical model in Figure 18.1, where we define $\mathbf{K}_2 = \mathcal{K}_2(\mathbf{F}_1, \mathbf{F}_1)$, $\mathbf{k}_{2*} = \mathcal{K}_2(\mathbf{F}_1, \mathbf{f}_1^*)$, and $\mathbf{k}_{2**} = \mathcal{K}_2(\mathbf{f}_1^*, \mathbf{f}_1^*)$.

18.1.0.2 Example: 1d step function

Suppose we have data from a piecewise constant function. (This can often happen when modeling certain physical processes, which can exhibit saturation effects.) Figure 18.2a shows what happens if we fit data from such a step function using a standard GP with an RBF (Gaussian) kernel. Obviously

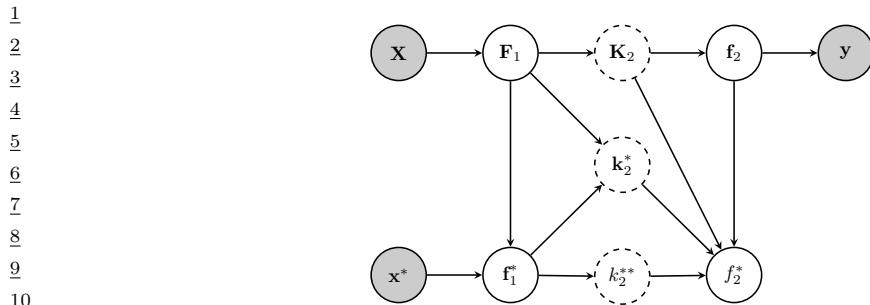


Figure 18.1: Graphical model corresponding to a deep GP with 2 layers. The dashed nodes are deterministic functions of their parents, and represent kernel matrices. The shaded nodes are observed, the unshaded nodes are hidden. Adapted from Figure 5 of [Pleiss2021].

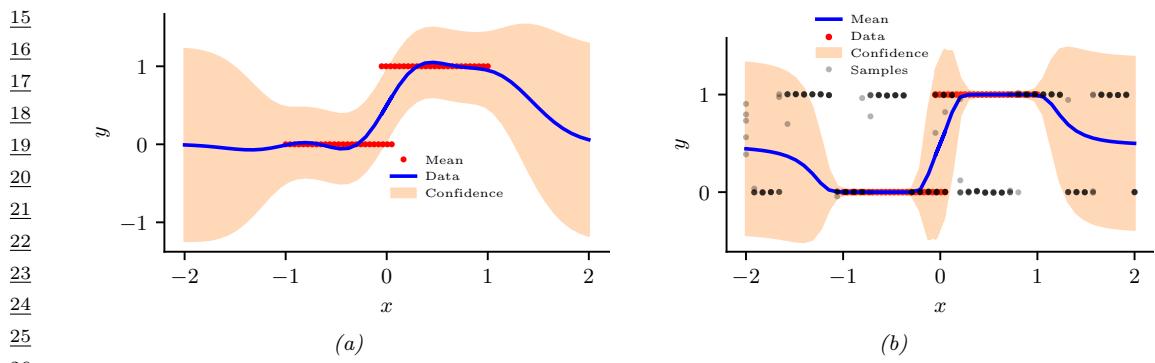


Figure 18.2: Some data (red points) sampled from a step function fit with (a) a standard GP with RBF kernel and (b) a deep GP with 4 layers of RBF kernels. The solid blue line is the posterior mean. The pink shaded area represents the posterior variance ($\mu(x) \pm 2\sigma(x)$). The thin blue dots in (b) represent posterior samples. Generated by `deepgp_stepdata`.

this method oversmooths the function and does not “pick up on” the underlying discontinuity. It is possible to learn kernels that can capture such discontinuous (non-stationary) behavior by learning to warp the input with a neural net before passing into the RBF kernel (see Main Figure 18.26).

Another approach is to learn a sequence of smooth mappings which together capture the overall complex behavior, analogous to the approach in deep learning. Suppose we fit a 4 layer DGP with a single hidden unit at each layer; we will use an RBF kernel. Thus the kernel at level 1 is $\mathcal{K}_1(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2/(2D))$, the kernel at level 2 is $\mathcal{K}_2(\mathbf{f}_1(\mathbf{x}), \mathbf{f}_1(\mathbf{x}')) = \exp(-\|\mathbf{f}_1(\mathbf{x}) - \mathbf{f}_1(\mathbf{x}')\|^2/(2H_1))$, etc.

We can perform posterior inference in this model to compute $p(\mathbf{f}_* | \mathbf{x}_*, \mathbf{X}, \mathbf{y})$ for a set of test points \mathbf{x}_* (see Section 18.1.0.3 for the details). Figure 18.2b shows the resulting posterior predictive distribution. We see that the predictions away from the data capture two plausible modes: either the signal continues at the level $y = 0$ or at $y = 1$. (The posterior mean, shown by the solid blue line, is a poor summary of the predictive distribution in this case, since it lies between these two modes.) This is an example of non-trivial extrapolation behavior outside of the support of the data.

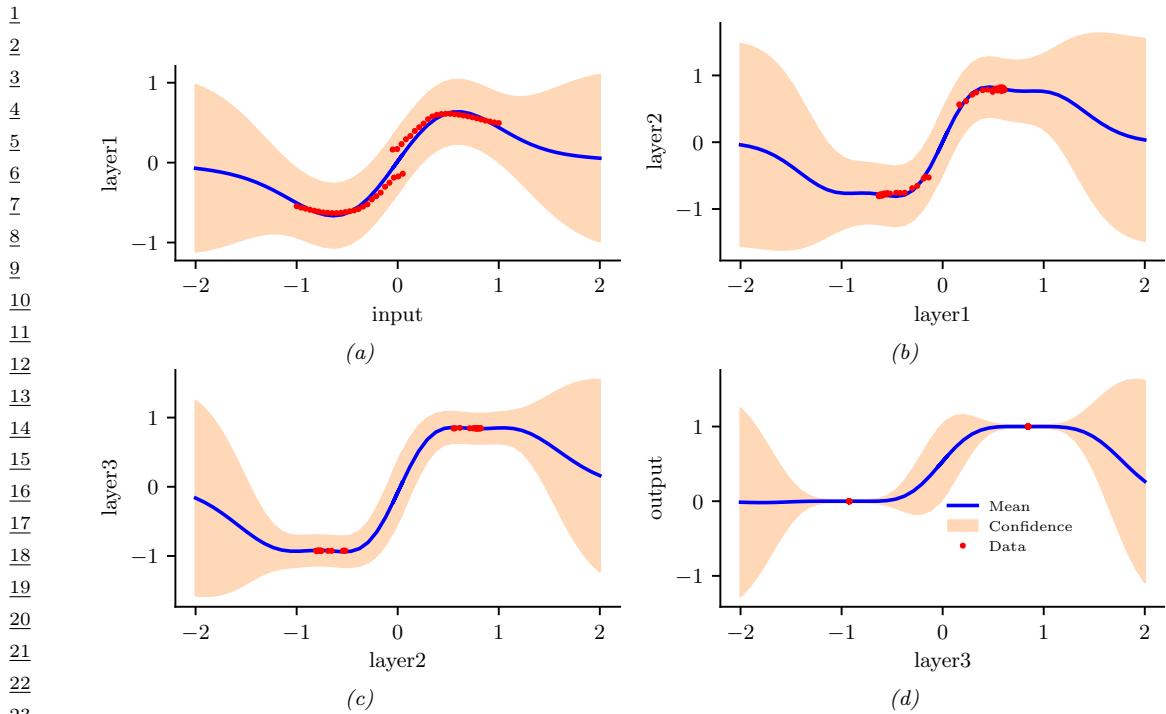


Figure 18.3: Illustration of the functions learned at each layer of the DGP. (a) Input to layer 1. (b) Layer 2 to layer 1. (c) Layer 2 to layer 3. (d) Layer 3 to output. Generated by [deeppg_stepdata.ipynb](#)

Figure 18.3 shows the individual functions learned at each layer (these are all maps from 1d to 1d). We see that the functions are individually smooth (since they are derived from an RBF kernel), but collectively they define non-smooth behavior.

18.1.0.3 Posterior inference

In Equation (18.2), we defined the joint distribution defined by a (2 layer) DGP. We can condition on \mathbf{y} to convert this into a joint posterior, as follows:

$$p(f_2^*, f_2, \mathbf{F}_1, f_1 | \mathbf{y}) = p(f_2^* | f_2, f_1^*, \mathbf{F}_1, \mathbf{y}) p(f_2 | \mathbf{F}_1, f_1^*, \mathbf{y}) p(f_1^* | \mathbf{F}_1, \mathbf{y}) \quad (18.3)$$

$$= p(f_2^* | f_2, f_1^*, \mathbf{F}_1) p(f_2 | \mathbf{F}_1, \mathbf{y}) p(f_1^* | \mathbf{F}_1, \mathbf{y}) \quad (18.4)$$

where the simplifications in the second line follow from the conditional independencies encoded in Figure 18.1. Note that f_2 and f_2^* depend on \mathbf{F}_1 and f_1^* only through \mathbf{K}_2 , k_2^* and k_2^{**} , where

$$p(f_2 | \mathbf{K}_2) \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_2), \quad p(f_2^* | k_2^{**}, k_2^*, \mathbf{K}_2, f_2) \sim \mathcal{N}((k_2^*)^\top \mathbf{K}_2^{-1} f_2, k_2^{**} - (k_2^*)^\top \mathbf{K}_2^{-1} k_2^*) \quad (18.5)$$

Hence

$$p(f_2^*, f_2, \mathbf{F}_1, f_1 | \mathbf{y}) = p(f_2^* | f_2, \mathbf{K}_2, k_2^*, k_2^{**}) p(f_2 | \mathbf{K}_2, \mathbf{y}) p(\mathbf{K}_2, k_2^*, k_2^{**} | \mathbf{y}) \quad (18.6)$$

For prediction we only care about f_2^* , so we marginalize out the other variables. The posterior mean is given by

$$\mathbb{E}_{f_2^*|\mathbf{y}} [f_2^*] = \mathbb{E}_{\mathbf{K}_2, \mathbf{k}_2^*, k_2^{**}|\mathbf{y}} [\mathbb{E}_{f_2|\mathbf{K}_2, \mathbf{y}} [\mathbb{E}_{f_2^*|f_2, \mathbf{K}_2, \mathbf{k}_2^*, k_2^{**}} [f_2^*]]] \quad (18.7)$$

$$= \mathbb{E}_{\mathbf{K}_2, \mathbf{k}_2^*, k_2^{**}|\mathbf{y}} [\mathbb{E}_{f_2|\mathbf{K}_2, \mathbf{y}} [(k_2^*)^\top \mathbf{K}_2^{-1} f_2]] \quad (18.8)$$

$$= \mathbb{E}_{\mathbf{K}_2, \mathbf{k}_2^*|\mathbf{y}} \left[\mathbf{k}_2^* \underbrace{\mathbf{K}_2^{-1} \mathbb{E}_{f_2|\mathbf{K}_2, \mathbf{y}} [f_2]}_{\alpha} \right] \quad (18.9)$$

Since \mathbf{K}_2 and \mathbf{k}_2^* are deterministic transformations of $\mathbf{f}_1(\mathbf{x}^*), \mathbf{f}_1(\mathbf{x}_1), \dots, \mathbf{f}_1(\mathbf{x}_N)$, we can rewrite this as

$$\mathbb{E}_{f_2^*|\mathbf{y}} [f_2^*] = \mathbb{E}_{\mathbf{f}_1(\mathbf{x}^*), \mathbf{f}_1(\mathbf{x}_1), \dots, \mathbf{f}_1(\mathbf{x}_N)|\mathbf{y}} \left[\sum_{i=1}^N \alpha_i \mathcal{K}_2(\mathbf{f}_1(\mathbf{x}_i), \mathbf{f}_1(\mathbf{x}^*)) \right] \quad (18.10)$$

We see from the above that inference in a DGP is, in general, very expensive, due to the need to marginalize over a lot of variables, corresponding to all the hidden function values at each layer at each data point. In [Salimbeni2017], they propose an approach to approximate inference in DGPs based on the sparse variational method of Main Section 10.1.1.1. The key assumption is that each layer has a set of inducing points, along with corresponding inducing values, that simplifies the dependence between unknown function values within each layer. However, the dependence between layers is modeled exactly. In [Dutordoir2021] they show that the posterior mean of such a sparse variational approximation can be computed by performing a forwards pass through a ReLU DNN.

18.1.0.4 Behavior in the limit of infinite width

Consider the case of a DGP where the depth is 2. The posterior mean of the predicted output at a test point is given by Equation (18.10). We see that this is a mixture of data-dependent kernel functions, since both \mathbf{K}_2 and \mathbf{k}_2 depend on the data \mathbf{y} . This is what makes deep GPs more expressive than single layer GPs, where the kernel is fixed. However, [Pleiss2021] show that, in the limit $H_1 \rightarrow \infty$, the posterior over the kernels for the layer 2 features becomes independent of the data, i.e., $p(\mathbf{K}_2, \mathbf{k}_2^*|\mathbf{y}) = \delta(\mathbf{K}_2 - \mathbf{K}_{\lim})\delta(\mathbf{k}_2^* - \mathbf{k}_{\lim}^*)$, where $\mathbf{K}_{\lim} = \mathbb{E}[f_2 f_2^\top]$ and $\mathbf{k}_{\lim}^* = \mathbb{E}[f_2 f_2^*]$, where the expectations depend on \mathbf{X} but not \mathbf{y} . Consequently the posterior predictive mean reduces to

$$\lim_{H_1 \rightarrow \infty} \mathbb{E}_{f_2^*|\mathbf{y}} [f_2^*] = \sum_{i=1}^N \alpha_i \mathcal{K}_{\lim}(\mathbf{x}_i, \mathbf{x}_*) \quad (18.11)$$

which is the same form as a single layer GP.

As a concrete example, consider a 2 layer DGP with an RBF kernel at each layer. Thus the kernel at level 1 is $\mathcal{K}_1(\mathbf{x}, \mathbf{x}') = \exp(-||\mathbf{x} - \mathbf{x}'||^2/(2D))$, and the kernel at level 2 is $\mathcal{K}_2(\mathbf{f}_1(\mathbf{x}), \mathbf{f}_1(\mathbf{x}')) = \exp(-||\mathbf{f}_1(\mathbf{x}) - \mathbf{f}_1(\mathbf{x}')||^2/(2H_1))$. Let us fit this model to a noisy step function. In Figure 18.4 we show the results as we increase the width of the hidden layer. When the width is 1, we see that the covariance of the resulting DGP, $\mathcal{K}_2(\mathbf{f}_1(\mathbf{x}), \mathbf{f}_1(\mathbf{x}'))$, is nonstationary. In particular, there are long-range correlations near $\mathbf{x} = \pm 1$ (since the function is constant in this region), but short range correlations near $\mathbf{x} = 0$ (since the function is changing rapidly in this region). However, as the width

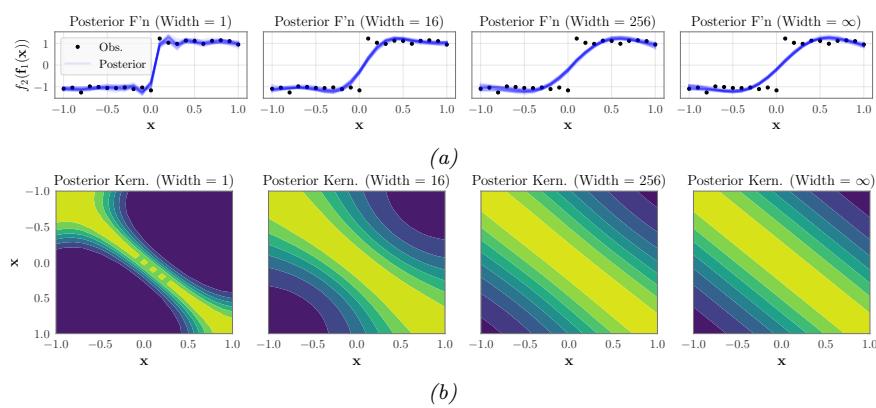


Figure 18.4: (a) Posterior of 2-layer RBF deep GP fit to a noisy step function. Columns represent width of 1, 16, 256 and infinity. (b) Average posterior covariance of the DGP, given by $\mathbb{E}_{\mathbf{f}_1(\mathbf{x}), \mathbf{f}_1(\mathbf{x}') | \mathbf{y}} [\mathcal{K}_2(\mathbf{f}_1(\mathbf{x}), \mathbf{f}_1(\mathbf{x}'))]$. As the width increases, the covariance becomes stationary, as shown by the kernel's constant diagonals. From Figure 1 of [Pleiss2021]. Used with kind permission of Geoff Pleiss.

increases, we lose this nonstationarity, as shown by the constant diagonals of the kernel matrix. Indeed, in [Pleiss2021] they prove that the limiting kernel is $\mathcal{K}_{\lim}(\mathbf{x}, \mathbf{x}') = \exp(\exp(-\|\mathbf{x} - \mathbf{x}'\|^2/(2D)) - 1)$, which is stationary.

In [Pleiss2021], they also show that increasing the width makes the marginals more Gaussian, due to central-limit like behavior. However, increasing the depth makes the marginals less Gaussian, and causes them to have sharper peaks and heavier tails. Thus one often gets best results with a deep GP if it is deep but narrow.

18.1.0.5 Connection with Bayesian neural networks

A Bayesian neural network (BNN) is a DNN in which we place priors over the parameters (see Main Section 17.1). One can show (see e.g., [Ober2021]) that BNNs are a degenerate form of deep GPs. For example, consider a 2 layer MLP, $f_2(f_1(\mathbf{x}))$, with $f_1 : \mathbb{R}^D \rightarrow \mathbb{R}^{H_1}$ and $f_2 : \mathbb{R}^{H_1} \rightarrow \mathbb{R}$, defined by

$$f_1^{(i)}(\mathbf{x}) = (\mathbf{w}_1^{(i)})^\top \mathbf{x} + \beta \mathbf{b}_1, \quad f_2(\mathbf{z}) = \frac{1}{\sqrt{H_1}} \mathbf{w}_2^\top \varphi(\mathbf{z}) + \beta b_2 \quad (18.12)$$

where $\beta > 0$ is a scaling constant, and $\mathbf{W}_1, \mathbf{b}_1, \mathbf{w}_2, b_2$ are Gaussian. The first layer is a linear regression model, and hence (from the results in Main Section 18.3.3) corresponds to a GP with a linear kernel of the form $\mathcal{K}_1(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$. The second layer is also a linear regression model but applied to features $\varphi(\mathbf{z})$. Hence (from the results in Main Section 18.3.3) this corresponds to a GP with a linear kernel of the form $\mathcal{K}_2(\mathbf{z}, \mathbf{z}') = \varphi(\mathbf{z})^\top \varphi(\mathbf{z}')$. Thus each layer of the model corresponds to a (degenerate) GP, and hence the overall model is a (degenerate) DGP. (The term “degenerate” refers to the fact that the covariance matrices only have a finite number of non-zero eigenvalues, due to the use of a finite set of basis functions.) Consequently we can use the results from Section 18.1.0.4 to conclude that infinitely wide DNNs also reduce to a single layer GP, as we already established in Main Section 18.7.1.

1 In practice we use finite-width DNNs. The width should be wide enough to approximate a standard
2 GP at each layer, but should not be too wide, otherwise the corresponding kernels of the resulting
3 deep GP will no longer be adapted to the data, i.e., there will not be any “feature learning”. See e.g.,
4 [Aitchison2020; Pleiss2021; Zavatone-Veth2021] for details.
5

6

7 18.2 GPs and SSMs

8

9 Consider a Matern kernel of order $\nu = \frac{3}{2}$ with length scale ℓ and variance σ^2 :
10

$$\mathcal{K}(\tau; \frac{3}{2}, \ell) = \sigma^2 \left(1 + \frac{\sqrt{3}\tau}{\ell} \right) \exp \left(-\frac{\sqrt{3}\tau}{\ell} \right) \quad (18.13)$$

14

For this kernel, we define

15

$$\mathbf{F} = \begin{pmatrix} 0 & 1 \\ -\lambda^2 & -2\lambda \end{pmatrix}, \mathbf{L} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \mathbf{H} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad (18.14)$$

18

19 Consider a Matern kernel of order $\nu = \frac{3}{2}$ with length scale ℓ and variance σ^2 :
20

$$\mathcal{K}(\tau; \frac{5}{2}, \ell) = \sigma^2 \left(1 + \frac{\sqrt{5}\tau}{\ell} + \frac{5\tau^2}{3\ell^2} \right) \exp \left(-\frac{\sqrt{5}\tau}{\ell} \right) \quad (18.15)$$

24

We define $\lambda = \frac{\sqrt{2\nu}}{\ell}$, $p = \nu - \frac{1}{2}$, and

25

$$\mathbf{F} = \begin{pmatrix} 0 & 1 & 0 \\ -\lambda^3 & -3\lambda^2 & -3\lambda \end{pmatrix}, \mathbf{L} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \mathbf{H} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad (18.16)$$

29

$$q = \frac{2\sigma^2\pi^{\frac{1}{2}}\lambda^{2p+1}\Gamma(p+1)}{\Gamma(p+\frac{1}{2})} \quad (18.17)$$

32

If $\Delta_k = t_k - t_{k-1}$, the LG-SSM becomes

33

$$\mathbf{z}_k = \mathbf{A}_{k-1}\mathbf{z}_{k-1} + \mathcal{N}(\mathbf{0}, \mathbf{Q}_{k-1}) \quad (18.18)$$

36

$$\mathbf{y}_k = \mathbf{H}\mathbf{z}_k + \mathcal{N}(0, \sigma_n^2) \quad (18.19)$$

37

where

38

$$\Phi(\tau) = \text{expm}(\mathbf{F}\tau) \quad (18.20)$$

41

$$\mathbf{A}_{k-1} = \Phi(\Delta_k) \quad (18.21)$$

42

$$\mathbf{Q}_{k-1} = \int_0^{\Delta_k} \Phi(\Delta_k - \tau) \mathbf{L} q \mathbf{L}^\top \Phi(\Delta_k - \tau)^\top d\tau \quad (18.22)$$

44454647

19 Beyond the iid assumption

PART IV

Generation

20 Generative models: an overview

21 Variational autoencoders

21.0.1 VAEs with missing data

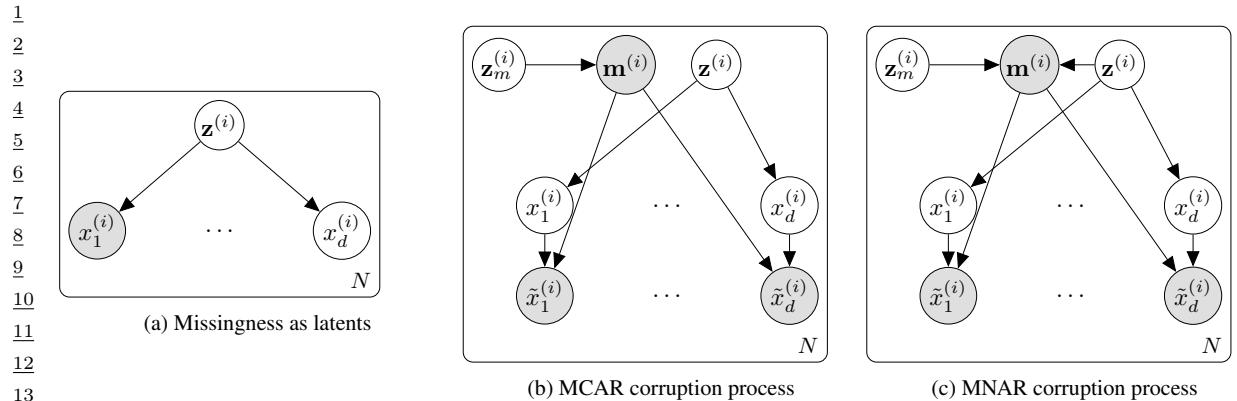
Sometimes we may have **missing data**, in which parts of the data vector $\mathbf{x} \in \mathbb{R}^D$ may be unknown. In ?? we saw a special case of this when we discussed multimodal VAEs. In this section we allow for arbitrary patterns of missingness.

To model the missing data, let $\mathbf{m} \in \{0, 1\}^D$ be a binary vector where $m_j = 1$ if x_j is missing, and $m_j = 0$ otherwise. Let $\mathbf{X} = \{\mathbf{x}^{(n)}\}$ and $\mathbf{M} = \{\mathbf{m}^{(n)}\}$ be $N \times D$ matrices. Furthermore, let \mathbf{X}_o be the observed parts of \mathbf{X} and \mathbf{X}_h be the hidden parts. If we assume $p(\mathbf{M}|\mathbf{X}_o, \mathbf{X}_h) = p(\mathbf{M})$, we say the data is **missing completely at random** or **MCAR**, since the missingness does not depend on the hidden or observed features. If we assume $p(\mathbf{M}|\mathbf{X}_o, \mathbf{X}_h) = p(\mathbf{M}|\mathbf{X}_o)$, we say the data is **missing at random** or **MAR**, since the missingness does not depend on the hidden features, but may depend on the visible features. If neither of these assumptions hold, we say the data is **not missing at random** or **NMAR**.

In the MCAR and MAR cases, we can ignore the missingness mechanism, since it tells us nothing about the hidden features. However, in the NMAR case, we need to model the **missing data mechanism**, since the lack of information may be informative. For example, the fact that someone did not fill out an answer to a sensitive question on a survey (e.g., “Do you have COVID?”) could be informative about the underlying value. See e.g., [Little87; Marlin08] for more information on missing data models.

In the context of VAEs, we can model the MCAR scenario by treating the missing values as latent variables. This is illustrated in Figure 21.1(a). Since missing leaf nodes in a directed graphical model do not affect their parents, we can simply ignore them when computing the posterior $p(\mathbf{z}^{(i)}|\mathbf{x}_o^{(i)})$, where $\mathbf{x}_o^{(i)}$ are the observed parts of example i . However, when using an amortized inference network, it can be difficult to handle missing inputs, since the model is usually trained to compute $p(\mathbf{z}^{(i)}|\mathbf{x}_{1:d}^{(i)})$. One solution to this is to use the product of experts approach discussed in the context of multi-modal VAEs in ???. However, this is designed for the case where whole blocks (corresponding to different modalities) are missing, and will not work well if there are arbitrary missing patterns (e.g., pixels that get dropped out due to occlusion or scratches on the lens). In addition, this method will not work for the NMAR case.

An alternative approach, proposed in [Collier2020], is to explicitly include the missingness indicators into the model, as shown in Figure 21.1(b). We assume the model always generates each \mathbf{x}_j for $j = 1 : d$, but we only get to see the “corrupted” versions $\tilde{\mathbf{x}}_j$. If $m_j = 0$ then $\tilde{\mathbf{x}}_j = \mathbf{x}_j$, but if $m_j = 1$, then $\tilde{\mathbf{x}}_j$ is a special value, such as 0, unrelated to \mathbf{x}_j . We can model any correlation between the missingness elements (components of \mathbf{m}) by using another latent variable \mathbf{z}_m . This model can



14 *Figure 21.1: Illustration of different VAE variants for handling missing data. From Figure 1 of [Collier2020].*
15 Used with kind permission of Mark Collier.

16
17
18 easily be extended to the NMAR case by letting \mathbf{m} depend on the latent factors for the observed
19 data, \mathbf{z} , as well as the usual missingness latent factors \mathbf{z}_m , as shown in Figure 21.1(c).
20

21 We modify the VAE to be conditional on the missingness pattern, so the VAE decoder has the
22 form $p(\mathbf{x}_o | \mathbf{z}, \mathbf{m})$, and the encoder has the form $q(\mathbf{z} | \mathbf{x}_o, \mathbf{m})$. However, we assume the prior is $p(\mathbf{z})$
23 as usual, independent of \mathbf{m} . We can compute a lower bound on the log marginal likelihood of the
24 observed data, given the missingness, as follows:

25
$$\log p(\mathbf{x}_o | \mathbf{m}) = \log \int \int p(\mathbf{x}_o, \mathbf{x}_m | \mathbf{z}, \mathbf{m}) p(\mathbf{z}) d\mathbf{x}_m d\mathbf{z} \quad (21.1)$$

26
$$= \log \int p(\mathbf{x}_o | \mathbf{z}, \mathbf{m}) p(\mathbf{z}) d\mathbf{z} \quad (21.2)$$

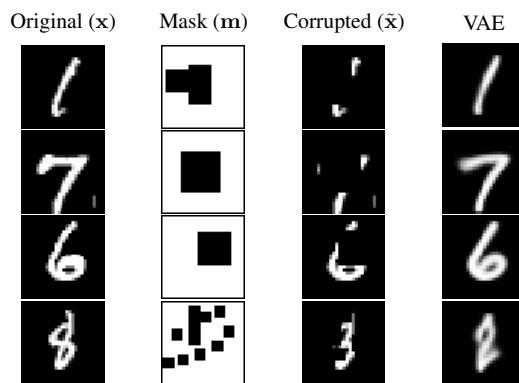
27
$$= \log \int p(\mathbf{x}_o | \mathbf{z}, \mathbf{m}) p(\mathbf{z}) \frac{q(\mathbf{z} | \tilde{\mathbf{x}}, \mathbf{m})}{q(\mathbf{z} | \tilde{\mathbf{x}}, \mathbf{m})} d\mathbf{z} \quad (21.3)$$

28
$$= \log \mathbb{E}_{q(\mathbf{z} | \tilde{\mathbf{x}}, \mathbf{m})} \left[p(\mathbf{x}_o | \mathbf{z}, \mathbf{m}) \frac{p(\mathbf{z})}{q(\mathbf{z} | \tilde{\mathbf{x}}, \mathbf{m})} \right] \quad (21.4)$$

29
$$\geq \mathbb{E}_{q(\mathbf{z} | \tilde{\mathbf{x}}, \mathbf{m})} [\log p(\mathbf{x}_o | \mathbf{z}, \mathbf{m})] - D_{\text{KL}}(q(\mathbf{z} | \tilde{\mathbf{x}}, \mathbf{m}) \| p(\mathbf{z})) \quad (21.5)$$

30 We can fit this model in the usual way.
31
32
33
34
35

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



22 Auto-regressive models

23 Normalizing flows

24 Energy-based models

25 Denoising diffusion models

26 Generative adversarial networks

PART V

Discovery

27 Discovery methods: an overview

28 Latent factor models

28.1 Inference in topic models

In this section, we discuss some methods for performing inference in LDA (Latent Dirichlet Allocation) models, defined in Main Section 28.5.1.

28.1.1 Collapsed Gibbs sampling for LDA

In this section, we discuss how to perform inference using MCMC.

The simplest approach is to use Gibbs sampling. The full conditionals are as follows:

$$p(m_{il} = k | \cdot) \propto \exp[\log \pi_{ik} + \log w_{k,x_{il}}] \quad (28.1)$$

$$p(\boldsymbol{\pi}_i | \cdot) = \text{Dir}(\{\alpha_k + \sum_l \mathbb{I}(m_{il} = k)\}) \quad (28.2)$$

$$p(\boldsymbol{w}_k | \cdot) = \text{Dir}(\{\gamma_v + \sum_i \sum_l \mathbb{I}(x_{il} = v, m_{il} = k)\}) \quad (28.3)$$

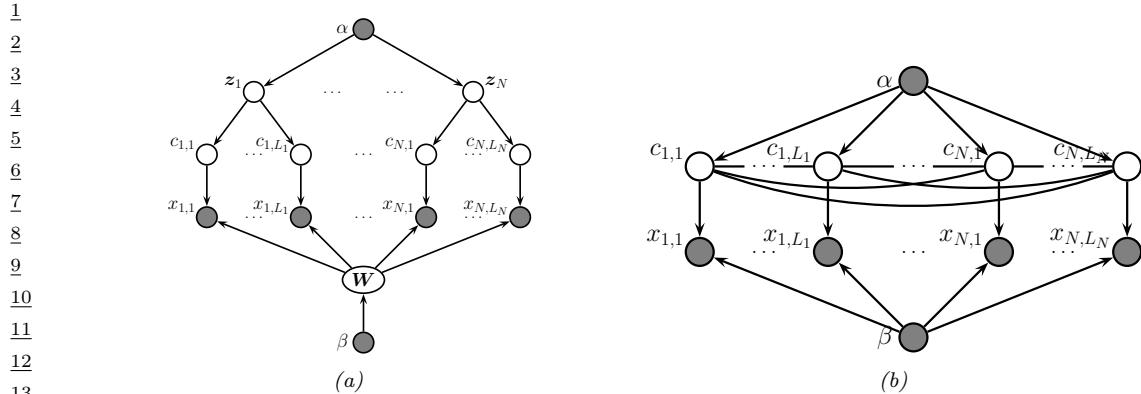
However, one can get better performance by analytically integrating out the $\boldsymbol{\pi}_i$'s and the \boldsymbol{w}_k 's, both of which have a Dirichlet distribution, and just sampling the discrete m_{il} 's. This approach was first suggested in [Griffiths04], and is an example of collapsed Gibbs sampling. Figure 28.1(b) shows that now all the m_{il} variables are fully correlated. However, we can sample them one at a time, as we explain below.

First, we need some notation. Let $N_{ivk} = \sum_{l=1}^{L_i} \mathbb{I}(m_{il} = k, x_{il} = v)$ be the number of times word v is assigned to topic k in document i . Let $N_{ik} = \sum_v N_{ivk}$ be the number of times any word from document i has been assigned to topic k . Let $N_{vk} = \sum_i N_{ivk}$ be the number of times word v has been assigned to topic k in any document. Let $N_k = \sum_v N_{vk}$ be the number of words assigned to topic k . Finally, let $L_i = \sum_k N_{ik}$ be the number of words in document i ; this is observed.

We can now derive the marginal prior. By applying Main Equation (3.94), one can show that

$$p(\boldsymbol{m} | \boldsymbol{\alpha}) = \prod_i \int \left[\prod_{l=1}^{L_i} \text{Cat}(m_{il} | \boldsymbol{\pi}_i) \right] \text{Dir}(\boldsymbol{\pi}_i | \boldsymbol{\alpha} \mathbf{1}_K) d\boldsymbol{\pi}_i \quad (28.4)$$

$$= \left(\frac{\Gamma(K\alpha)}{\Gamma(\alpha)^K} \right)^N \prod_{i=1}^N \frac{\prod_{k=1}^K \Gamma(N_{ik} + \alpha)}{\Gamma(L_i + K\alpha)} \quad (28.5)$$



14 Figure 28.1: (a) LDA unrolled for N documents. (b) Collapsed LDA, where we integrate out the continuous
15 latents z_n and the continuous topic parameters \mathbf{W} .

16

17

18 By similar reasoning, one can show

19

$$\begin{aligned} p(\mathbf{x}|\mathbf{m}, \beta) &= \prod_k \int \left[\prod_{il: m_{il}=k} \text{Cat}(x_{il}|\mathbf{w}_k) \right] \text{Dir}(\mathbf{w}_k|\beta\mathbf{1}_V) d\mathbf{w}_k \end{aligned} \quad (28.6)$$

20

$$= \left(\frac{\Gamma(V\beta)}{\Gamma(\beta)^V} \right)^K \prod_{k=1}^K \frac{\prod_{v=1}^V \Gamma(N_{vk} + \beta)}{\Gamma(N_k + V\beta)} \quad (28.7)$$

21

22 From the above equations, and using the fact that $\Gamma(x+1)/\Gamma(x) = x$, we can derive the full
23 conditional for $p(m_{il}|\mathbf{m}_{-i,l})$. Define N_{ivk}^- to be the same as N_{ivk} except it is computed by summing
24 over all locations in document i except for m_{il} . Also, let $x_{il} = v$. Then
25

26

$$p(m_{i,l} = k|\mathbf{m}_{-i,l}, \mathbf{y}, \alpha, \beta) \propto \frac{N_{v,k}^- + \beta}{N_k^- + V\beta} \frac{N_{i,k}^- + \alpha}{L_i + K\alpha} \quad (28.8)$$

27

28 We see that a word in a document is assigned to a topic based both on how often that word is
29 generated by the topic (first term), and also on how often that topic is used in that document (second
30 term).

31

32 Given Equation (28.8), we can implement the collapsed Gibbs sampler as follows. We randomly
33 assign a topic to each word, $m_{il} \in \{1, \dots, K\}$. We can then sample a new topic as follows: for a
34 given word in the corpus, decrement the relevant counts, based on the topic assigned to the current
35 word; draw a new topic from Equation (28.8), update the count matrices; and repeat. This algorithm
36 can be made efficient since the count matrices are very sparse [Li2014lda].

37

38 This process is illustrated in Figure 28.2 on a small example with two topics, and five words.
39 The left part of the figure illustrates 16 documents that were sampled from the LDA model using
40 $p(\text{money}|k=1) = p(\text{loan}|k=1) = p(\text{bank}|k=1) = 1/3$ and $p(\text{river}|k=2) = p(\text{stream}|k=2) =$
41 $p(\text{bank}|k=2) = 1/3$. For example, we see that the first document contains the word “bank” 4 times
42 (indicated by the four dots in row 1 of the “bank” column), as well as various other financial terms.
43 The right part of the figure shows the state of the Gibbs sampler after 64 iterations. The “correct”
44

45

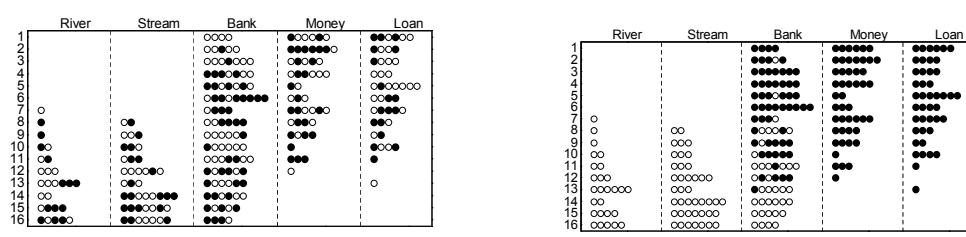


Figure 28.2: Illustration of (collapsed) Gibbs sampling applied to a small LDA example. There are $N = 16$ documents, each containing a variable number of words drawn from a vocabulary of $V = 5$ words. There are two topics. A white dot means word the word is assigned to topic 1, a black dot means the word is assigned to topic 2. (a) The initial random assignment of states. (b) A sample from the posterior after 64 steps of Gibbs sampling. From Figure 7 of [Steyvers07]. Used with kind permission of Tom Griffiths.

topic has been assigned to each token in most cases. For example, in document 1, we see that the word “bank” has been correctly assigned to the financial topic, based on the presence of the words “money” and “loan”. The posterior mean estimate of the parameters is given by $\hat{p}(\text{money}|k=1) = 0.32$, $\hat{p}(\text{loan}|k=1) = 0.29$, $\hat{p}(\text{bank}|k=1) = 0.39$, $\hat{p}(\text{river}|k=2) = 0.25$, $\hat{p}(\text{stream}|k=2) = 0.4$, and $\hat{p}(\text{bank}|k=2) = 0.35$, which is impressively accurate, given that there are only 16 training examples.

28.1.2 Variational inference for LDA

A faster alternative to MCMC is to use variational EM, which we discuss in general terms in Main Section 6.5.6.1. There are several ways to apply this to LDA, which we discuss in the following sections.

28.1.2.1 Sequence version

In this section, we focus on a version in which we unroll the model, and work with a latent variable for each word. Following [Blei03], we will use a fully factorized (mean field) approximation of the form

$$q(\mathbf{z}_n, \mathbf{s}_n) = \text{Dir}(\mathbf{z}_n | \tilde{\mathbf{z}}_n) \prod_l \text{Cat}(s_{nl} | \tilde{\mathbf{N}}_{nl}) \quad (28.9)$$

where $\tilde{\mathbf{z}}_n$ are the variational parameters for the approximate posterior over \mathbf{z}_n , and $\tilde{\mathbf{N}}_{nl}$ are the variational parameters for the approximate posterior over s_{nl} . We will follow the usual mean field recipe. For $q(s_{nl})$, we use Bayes’ rule, but where we need to take expectations over the prior:

$$\tilde{N}_{nlk} \propto w_{d,k} \exp(\mathbb{E}[\log z_{nk}]) \quad (28.10)$$

where $d = x_{nl}$, and

$$\mathbb{E}[\log z_{nk}] = \psi_k(\tilde{\mathbf{z}}_n) \triangleq \Psi(\tilde{z}_{nk}) - \psi\left(\sum_{k'} \tilde{z}_{nk'}\right) \quad (28.11)$$

1 where ψ is the digamma function. The update for $q(\mathbf{z}_n)$ is obtained by adding up the expected
2 counts:
3

4 $\tilde{z}_{nk} = \alpha_k + \sum_l \tilde{N}_{nlk}$ (28.12)
5
6

7 The M step is obtained by adding up the expected counts and normalizing:
8

9 $\hat{w}_{dk} \propto \beta_d + \sum_{n=1}^N \sum_{l=1}^{L_n} \tilde{N}_{nlk} \mathbb{I}(x_{nl} = d)$ (28.13)
10
11
12

13 28.1.2.2 Count version

14 Note that the E step takes $O((\sum_n L_n) N_w N_z)$ space to store the \tilde{N}_{nlk} . It is much more space efficient
15 to perform inference in the mPCA version of the model, which works with counts; these only take
16 $O(N N_w N_z)$ space, which is a big savings if documents are long. (By contrast, the collapsed Gibbs
17 sampler must work explicitly with the s_{nl} variables.)
18

19 Following the discussion in Main Section 28.4.1, we will work with the variables \mathbf{z}_n and \mathbf{N}_n , where
20 $\mathbf{N}_n = [N_{ndk}]$ is the matrix of counts, which can be derived from $\mathbf{s}_{n,1:L_n}$. We will again use a fully
21 factorized (mean field) approximation of the form

22 $q(\mathbf{z}_n, \mathbf{N}_n) = \text{Dir}(\mathbf{z}_n | \tilde{\mathbf{z}}_n) \prod_d \mathcal{M}(\mathbf{N}_{nd} | x_{nd}, \tilde{N}_{nd})$ (28.14)
23
24

25 where $x_{nd} = \sum_{l=1}^{L_n} \mathbb{I}(x_{nl} = d)$ is the total number of times token d occurs in document n .
26

27 The E step becomes

28 $\tilde{z}_{nk} = \alpha_k + \sum_d x_{nd} \tilde{N}_{ndk}$ (28.15)
29
30

31 $\tilde{N}_{ndk} \propto w_{dk} \exp(\mathbb{E}[\log z_{nk}])$ (28.16)
32

33 The M step becomes

34 $\hat{w}_{dk} \propto \beta_d + \sum_n x_{nd} \tilde{N}_{ndk}$ (28.17)
35
36

37 28.1.2.3 Bayesian version

38 We now modify the algorithm to use variational Bayes (VB) instead of EM, i.e., we infer the
39 parameters as well as the latent variables. There are two advantages to this. First, by setting $\beta \ll 1$,
40 VB will encourage \mathbf{W} to be sparse (as in Main Section 10.3.6.6). Second, we will be able to generalize
41 this to the online learning setting, as we discuss below.
42

43 Our new posterior approximation becomes

44 $q(\mathbf{z}_n, \mathbf{N}_n, \mathbf{W}) = \text{Dir}(\mathbf{z}_n | \tilde{\mathbf{z}}_n) \prod_d \mathcal{M}(\mathbf{N}_{nd} | x_{nd}, \tilde{N}_{nd}) \prod_k \text{Dir}(\mathbf{w}_k | \tilde{\mathbf{w}}_k)$ (28.18)
45
46

1

2 **Algorithm 28.1:** Batch VB for LDA

3 1 Input: $\{x_{nd}\}$, N_z , α , β

4 2 Estimate \tilde{w}_{dk} using EM for multinomial mixtures

5 3 **while** not converged **do**

6 4 // E step

7 5 $a_{dk} = 0$ // expected sufficient statistics

8 6 **for** each document $n = 1 : N$ **do**

9 7 $(\tilde{z}_n, \tilde{\mathbf{N}}_n) = \text{VB-Estep}(\mathbf{x}_n, \tilde{\mathbf{W}}, \alpha)$

10 8 $a_{dk} += x_{nd} \tilde{N}_{ndk}$

11 9 // M step

12 10 **for** each topic $k = 1 : N_z$ **do**

13 11 $\tilde{w}_{dk} = \beta_d + a_{dk}$

14 12 function $(\tilde{z}_n, \tilde{\mathbf{N}}_n) = \text{VB-Estep}(\mathbf{x}_n, \tilde{\mathbf{W}}, \alpha)$

15 13 Initialize $\tilde{z}_{nk} = \alpha_k$

16 14 repeat

17 15 $\tilde{z}_n^{old} = \tilde{z}_n$, $\tilde{z}_{nk} = \alpha_k$

18 16 **for** each word $d = 1 : N_w$ **do**

19 17 **for** each topic $k = 1 : N_z$ **do**

20 18 $\tilde{N}_{ndk} = \exp(\psi_k(\tilde{w}_d) + \psi_k(\tilde{z}_n^{old}))$

21 19 $\tilde{\mathbf{N}}_{nd} = \text{normalize}(\tilde{\mathbf{N}}_{nd})$

22 20 $\tilde{z}_n += x_{nd} \tilde{\mathbf{N}}_{nd}$

23 21 until Converged

27

28

29 The update for \tilde{N}_{ndk} changes, to the following:

30

$$\tilde{N}_{ndk} \propto \exp(\mathbb{E}[\log w_{dk}] + \mathbb{E}[\log z_{nk}]) \quad (28.19)$$

33 The M step is the same as before:

34

$$\hat{w}_{dk} \propto \beta_d + \sum_n x_{nd} \tilde{N}_{ndk} \quad (28.20)$$

35

37 No normalization is required, since we are just updating the pseudocounts. The overall algorithm is
38 summarized in Algorithm 28.1.

39

40 **28.1.2.4 Online (SVI) version**

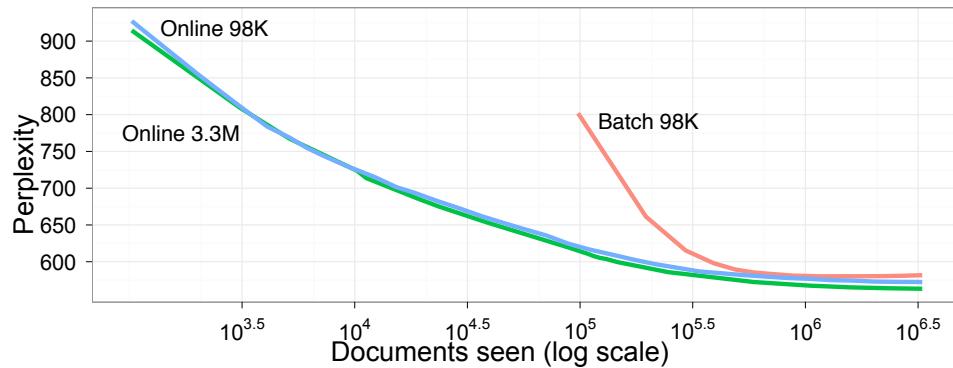
42 In the batch version, the E step takes $O(NN_z N_w)$ per mean field update. This can be slow if we
43 have many documents. This can be reduced by using stochastic variational inference, as discussed
44 in Main Section 10.1.4. We perform an E step in the usual way. We then compute the variational
45 parameters for \mathbf{W} treating the expected sufficient statistics from the single data case as if the whole
46 data set had those statistics. Finally, we make a partial update for the variational parameters for
47

```

Algorithm 28.2: Online VB for LDA

1 Input:  $\{x_{nd}\}$ ,  $N_z$ ,  $\alpha$ ,  $\beta$ , LR schedule
2 Initialize  $\tilde{w}_{dk}$  randomly
3 for  $t = 1 : \infty$  do
4   Set step size  $\eta_t$ 
5   Pick document  $n$ ;
6    $(\tilde{z}_n, \tilde{\mathbf{N}}_n) = \text{VB-Estep}(\mathbf{x}_n, \tilde{\mathbf{W}}, \alpha)$ 
7    $\tilde{w}_{dk}^{new} = \beta_d + N x_{nd} \tilde{N}_{ndk}$ 
8    $\tilde{w}_{dk} = (1 - \eta_t) \tilde{w}_{dk} + \eta_t \tilde{w}_{dk}^{new}$ 

```



²⁶ Figure 28.3: Test perplexity vs number of training documents for batch and online VB-LDA. From Figure 1 of [Hoffman10]. Used with kind permission of David Blei.

³⁰ **W**, putting weight η_t on the new estimate and weight $1 - \eta_t$ on the old estimate. The step size η_t decays over time, according to some schedule, as in SGD. The overall algorithm is summarized in ³² Algorithm 28.2. In practice, we should use mini-batches. In [Hoffman10], they used a batch of size ³³ 256–4096.

³⁴ Figure 28.3 plots the perplexity on a test set of size 1000 vs number of analyzed documents (E
³⁵ steps), where the data is drawn from (English) Wikipedia. The figure shows that online variational
³⁶ inference is much faster than offline inference, yet produces similar results.

29 State-space models

29.1 Continuous time SSMs

In this section, we briefly discuss continuous time dynamical systems.

29.1.1 Ordinary differential equations

We first consider a 1d system, whose state at time t is $z(t)$. We assume this evolves according to the following **nonlinear differential equation**:

$$\frac{dz}{dt} = f(t, z) \quad (29.1)$$

We assume the observations occur at discrete time steps t_i ; we can estimate the hidden state at these time steps, and then evolve the system dynamics in continuous time until the next measurement arrives. To compute z_i from z_{i-1} , we use

$$z_{i+1} = z_i + \int_{t_{i-1}}^{t_i} f(t, z_i) dt \quad (29.2)$$

To compute the integral, we will use the second-order **Runge-Kutta method**, with a step size of $\Delta = t_i - t_{i-1}$ be the sampling frequency. This gives rise to the following update:

$$k_1 = f(t_i, z_i) \quad (29.3)$$

$$k_2 = f(t_i + \Delta, z_i + k_1 \Delta) \quad (29.4)$$

$$z_{i+1} = z_i + \frac{\Delta}{2}(k_1 + k_2) \quad (29.5)$$

The term k_1 is the slope at z_i , and the term k_2 is the slope at z_{i+1} , so $\frac{1}{2}(k_1 + k_2)$ is the average slope. Thus z_{i+1} is the initial value z_i plus the step size Δ times the average slope, as illustrated in Figure 29.1.

When we have a vector-valued state-space, we need to solve a multidimensional integral. However, if the components are conditionally independent given the previous state, we can reduce this to a set of independent 1d integrals.

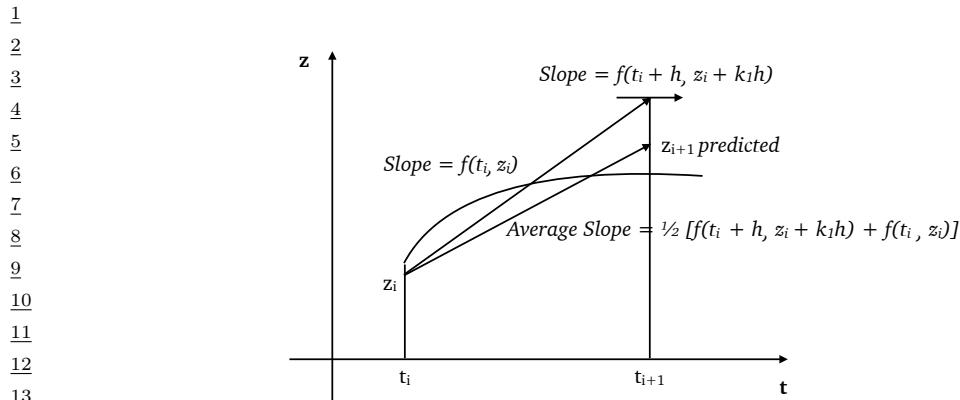


Figure 29.1: Illustration of one step of second-order Runge-Kutta method with step size h .

29.1.2 Example: Noiseless 1d spring-mass system

In this section, we consider an example from Wikipedia¹ of a **spring mass system** operating in 1d. Like many physical systems, this is best modeled in **continuous time**, although we will later discretize it.

Let $x(t)$ be the position of an object which is attached by a spring to a wall, and let $\dot{x}(t)$ and $\ddot{x}(t)$ be its velocity and acceleration. By **Newton's laws of motion**, we have the following **ordinary differential equation**:

$$m\ddot{x}(t) = u(t) - b\dot{x}(t) - cx(t) \quad (29.6)$$

where $u(t)$ is an externally applied force (e.g., someone tugging on the object), b is the viscous friction coefficient, c is the spring constant, and m is the mass of the object. See Figure 29.3 for the setup. We assume that we only observe the position, and not the velocity.

We now proceed to represent this as a first order Markov system. For simplicity, we ignore the noise ($\mathbf{Q}_t = \mathbf{R}_t = \mathbf{0}$). We define the state space to contain the position and velocity, $\mathbf{z}(t) = [x(t), \dot{x}(t)]$. Thus the model becomes

$$\dot{\mathbf{z}}(t) = \mathbf{F}\mathbf{z}(t) + \mathbf{B}u(t) \quad (29.7)$$

$$\mathbf{y}(t) = \mathbf{H}\mathbf{z}(t) + \mathbf{D}u(t) \quad (29.8)$$

where

$$\begin{pmatrix} \dot{x}(t) \\ \ddot{x}(t) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -\frac{c}{m} & -\frac{b}{m} \end{pmatrix} \begin{pmatrix} x(t) \\ \dot{x}(t) \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{m} \end{pmatrix} u(t) \quad (29.9)$$

$$\mathbf{y}(t) = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} x(t) \\ \dot{x}(t) \end{pmatrix} \quad (29.10)$$

1. https://en.wikipedia.org/wiki/State-space_representation#Moving_object_example

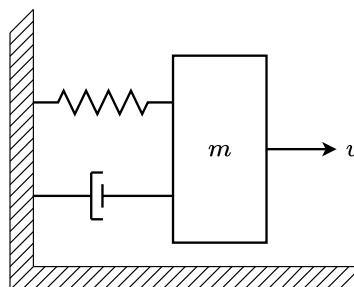


Figure 29.2: Illustration of the spring mass system.

To simulate from this system, we need to evaluate the state at a set of discrete time intervals, $t_k = k\Delta$, where Δ is the sampling rate or step size. There are many ways to discretize an ODE.² Here we discuss the **generalized bilinear transform** [Zhang2007]. In this approach, we specify a step size Δ and compute

$$\underline{z}_k = \underbrace{(\mathbf{I} - \alpha\Delta\mathbf{F})^{-1}(\mathbf{I} + (1 - \alpha)\Delta\mathbf{F})}_{\mathbf{A}} \underline{z}_{k-1} + \underbrace{\Delta(\mathbf{I} - \alpha\Delta\mathbf{F})^{-1}\mathbf{B}}_{\mathbf{B}} \mathbf{u}_k \quad (29.11)$$

If we set $\alpha = 0$, we recover **Euler's method**, which simplifies to

$$\underline{z}_k = \underbrace{(\mathbf{I} + \Delta\mathbf{F})}_{\mathbf{A}} \underline{z}_{k-1} + \underbrace{\Delta\mathbf{B}}_{\mathbf{B}} \mathbf{u}_k \quad (29.12)$$

If we set $\alpha = 1$, we recover the **backward Euler method**. If we set $\alpha = \frac{1}{2}$ we get the **bilinear method**, which preserves the stability of the system [Zhang2007]; we will use this in Section 29.2. Regardless of how we do the discretization, the resulting discrete time SSM becomes

$$\underline{z}_k = \bar{\mathbf{F}} \underline{z}_{k-1} + \bar{\mathbf{B}} \mathbf{u}_k \quad (29.13)$$

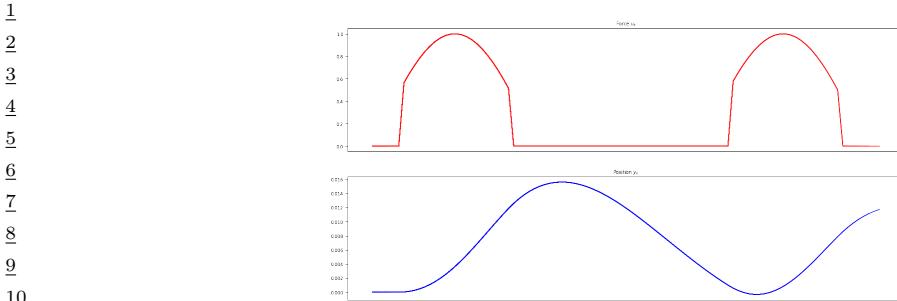
$$\mathbf{y}_k = \mathbf{H} \underline{z}_k + \mathbf{D} \mathbf{u}_k \quad (29.14)$$

Now consider simulating a system where we periodically “tug” on the object, so the force increases and then decreases for a short period, as shown in the top row of Figure 29.3. We can discretize the dynamics and compute the corresponding state and observation at integer time points. The result is shown in the bottom row of Figure 29.3. We see that the object’s location changes smoothly, since it integrates the force over time.

29.1.3 Example: tracking a moving object in continuous time

In this section, we consider a variant of the example in Section 8.1.1. We modify the dynamics so that energy is conserved, by ensuring the velocity is constant, and by working in continuous time.

². See discussion at <https://en.wikipedia.org/wiki/Discretization>.



11 *Figure 29.3: Signals generated by the spring mass system. Top row shows the input force. Bottom row*
12 *shows the observed location of the end-effector. Adapted from A figure by Sasha Rush. Generated by*
13 *[ssm_spring_demo.ipynb](#).*

14

15

16 Thus the particle moves in a circle, rather than spiralling in towards the origin. (See [**Strogatz2015**]
17 for details.) The dynamics model becomes

18

$$\underline{19} \quad \mathbf{z}_t = \mathbf{F}\mathbf{z}_{t-1} + \boldsymbol{\epsilon}_t \quad (29.15)$$

20

$$\underline{21} \quad \mathbf{F} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \quad (29.16)$$

22

23 where $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ is the system noise. We set $\mathbf{Q} = 0.001\mathbf{I}$, so the noise is negligible.

24 To see why this results in circular dynamics, we follow a derivation from Gerardo Durán-Martín.

25 We can represent a point in the plane using polar coordinates, (r, θ) , or Euclidean coordinates, (x, y) .

26 We can switch between these using standard trigonometric identities:

27

$$\underline{28} \quad x_t = r_t \cos \theta_t, y_t = r_t \sin \theta_t \quad (29.17)$$

29

30 The (noiseless) dynamical system has the following form:

$$\underline{31} \quad \begin{pmatrix} \dot{x}_t \\ \dot{y}_t \end{pmatrix} = \begin{pmatrix} y_t \\ -x_t \end{pmatrix} \quad (29.18)$$

32

33 We now show that this implies that

34

$$\underline{35} \quad \dot{r} = 0, \dot{\theta} = -1 \quad (29.19)$$

36

37 which means the radius is constant, and the angle changes at a constant rate. To see why, first note

38 that

39

$$\underline{40} \quad r_t^2 = x_t^2 + y_t^2 \quad (29.20)$$

41

$$\underline{42} \quad \frac{d}{dt} r_t^2 = \frac{d}{dt} (x_t^2 + y_t^2) \quad (29.21)$$

43

$$\underline{44} \quad 2r_t \dot{r}_t = 2x_t \dot{x}_t + 2y_t \dot{y}_t \quad (29.22)$$

45

$$\underline{46} \quad \dot{r}_t = \frac{x_t \dot{x}_t + y_t \dot{y}_t}{r_t} \quad (29.23)$$

47

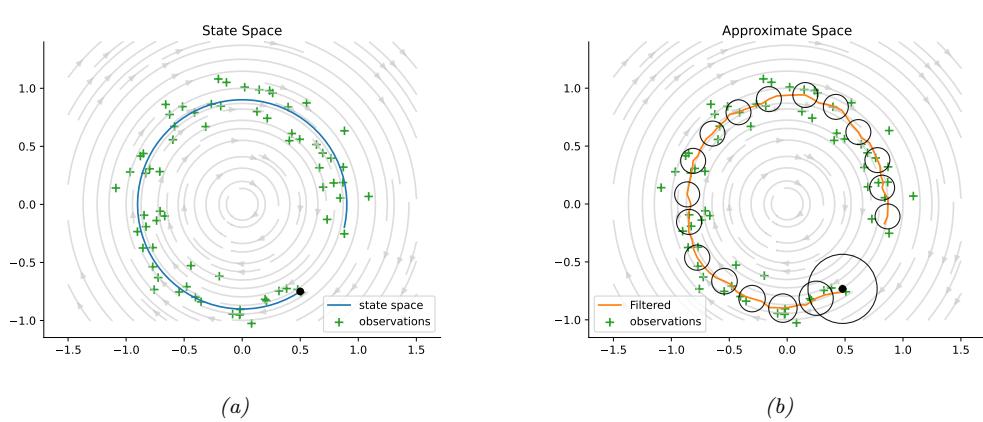


Figure 29.4: Illustration of Kalman filtering applied to a 2d linear dynamical system in continuous time. (a) True underlying state and observed data. (b) Estimated state. Generated by `kf_continuous_circle.ipynb`.

Also,

$$\tan \theta_t = \frac{y_t}{x_t} \quad (29.24)$$

$$\frac{d}{dt} \tan \theta_t = \frac{d}{dt} \frac{y_t}{x_t} \quad (29.25)$$

$$\dot{\theta}_t \sec^2(\theta_t) = \frac{1}{x_t} \dot{y}_t - \frac{y_t}{x_t^2} \dot{x}_t = \frac{x_t \dot{y}_t - \dot{x}_t y_t}{x_t^2} \quad (29.26)$$

$$\dot{\theta}_t = \frac{x_t \dot{y}_t - \dot{x}_t y_t}{r_t^2} \quad (29.27)$$

Plugging into Equation (29.18), and using the fact that $\cos^2 + \sin^2 = 1$, we have

$$\dot{r} = \frac{(r \cos \theta)(r \sin \theta) - (r \sin \theta)(r \cos \theta)}{r^2} = 0 \quad (29.28)$$

$$\dot{\theta} = \frac{-(r \cos \theta)(r \cos \theta) - (r \sin \theta)(r \sin \theta)}{r^2} = \frac{-r^2(\cos^2 \theta + \sin^2 \theta)}{r^2} = -1 \quad (29.29)$$

In most applications, we cannot directly see the underlying states. Instead, we just get noisy observations. Thus we will assume the following observation model:

$$u_t = H z_t + \delta_t \quad (29.30)$$

$$\mathbf{H} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad (29.31)$$

where $\delta \sim \mathcal{N}(0, B)$ is the measurement noise. We set $B = 0.01I$

We sample from this model and apply the Kalman filter to the resulting synthetic data, to estimate the underlying hidden states. To ensure energy conservation, we integrate the dynamics between

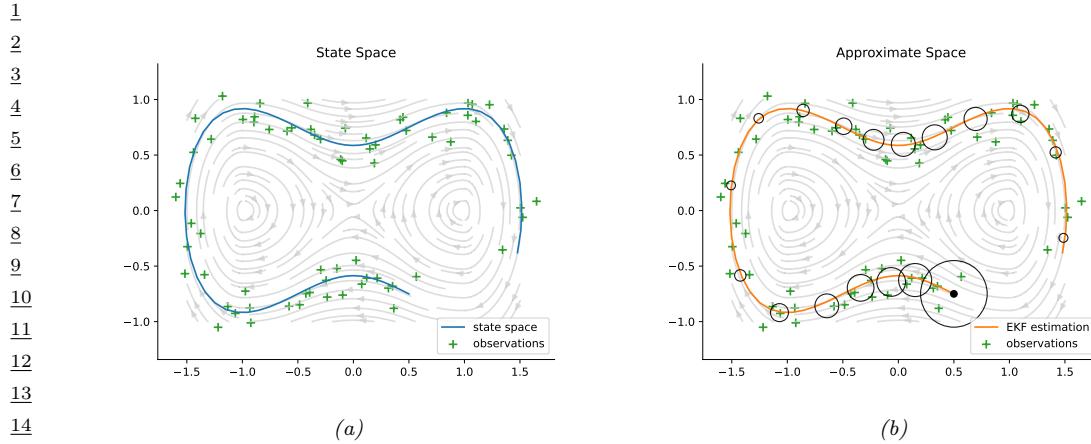


Figure 29.5: Illustration of extended Kalman filtering applied to a 2d nonlinear dynamical system. (a) True underlying state and observed data. (b) Estimated state. Generated by [ekf_continuous.ipynb](#).

each observation in continuous time, using the RK2 method in Section 29.1.1. The result is shown in Figure 29.4. We see that the method is able to filter out the noise, and track the underlying hidden state.

29.1.4 Example: tracking a particle in 2d

Consider a particle moving in continuous time in 2d space with the following nonlinear dynamics:

$$\mathbf{z}' = f_{\mathbf{z}}(\mathbf{z}) = \begin{pmatrix} z_2 \\ z_1 - z_1^3 \end{pmatrix} \quad (29.32)$$

This is a mildly nonlinear version of the model in Section 29.1.3. We can use the RK2 method of Section 29.1.1 applied to each component separately to compute the latent trajectory of the particle. We use a step size of $h = dt = 0.01$ and simulate from $t = 0$ to $T = 7.5$. Thus the number of integration steps is $N = T/h = 750$.

We sample the system at $K = 70$ evenly spaced time steps, and generate noisy observations using $\mathbf{y}_t = \mathbf{h}(\mathbf{z}_t) + \mathcal{N}(\mathbf{0}, \mathbf{R})$, where $\mathbf{h}(\mathbf{z}) = \mathbf{z}$ is the identity function. See Figure 29.5(a) for the hidden trajectory and corresponding noisy observations.

In Figure 29.5(b), we condition on the noisy observations, and compute $p(\mathbf{z}_t | \mathbf{y}_{1:t})$ using the EKF. We see that the method can successfully filter out the noise.

29.2 Structured State Space Sequence model (S4)

In this section, we briefly discuss a new sequence-to-sequence model known as the **Structured State Space Sequence model** or **S4** [S4; hippo; Goel2022]. Our presentation of S4 is based in part on the excellent tutorial [RushS4].

An S4 model is basically a deep stack of (noiseless) linear SSMs (Main Section 29.6). In between each layer we add pointwise nonlinearities and a linear mapping. Because SSMs are recurrent first-

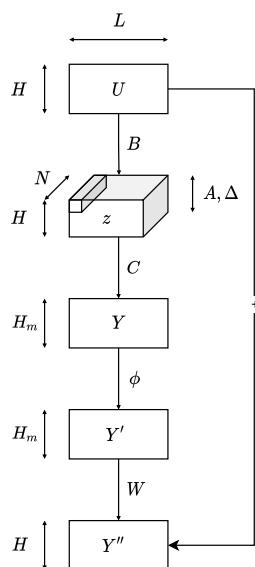


Figure 29.6: Illustration of one S4 block. The input \mathbf{X} has H sequences, each of length L . These get mapped (in parallel for each of the sequences) to the output \mathbf{Y} by a (noiseless) linear SSM with state size N and parameters $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$. The output \mathbf{Y} is mapped pointwise through a nonlinearity ϕ to create \mathbf{Y}' . The channels are then linearly combined by weight matrix \mathbf{W} and added to the input (via a skip connection) to get the final output \mathbf{Y}'' , which is another set of H sequences of length L .

order models, we can easily generate (sample) from them in $O(L)$ time, where L is the length of the sequence. This is much faster than the $O(L^2)$ required by standard transformers (Main Section 16.3.5). However, because these SSMs are linear, it turns out that we compute all the hidden representations given known inputs in parallel using convolution; this makes the models fast to train. Finally, since S4 models are derived from an underlying continuous time process, they can easily be applied to observations at different temporal frequencies. Empirically S4 has been found to be much better at modeling long range dependencies compared to transformers, which (at the time of writing, namely January 2022) are considered state of the art.

The basic building block, known as a **Linear State Space Layer (LSSL)**, is the following continuous time linear dynamical system that maps an input sequence $\mathbf{u}(t) \in \mathbb{R}$ to an output sequence $\mathbf{y}(t) \in \mathbb{R}^1$ via a sequence of hidden states $\mathbf{z}(t) \in \mathbb{R}^N$:

$$\dot{\mathbf{z}}(t) = \mathbf{A}\mathbf{z}(t) + \mathbf{B}\mathbf{u}(t) \quad (29.33)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{z}(t) + \mathbf{D}\mathbf{u}(t) \quad (29.34)$$

Henceforth we will omit the skip connection corresponding to the \mathbf{D} term for brevity. We can convert this to a discrete time system using the generalized bilinear transform discussed in Section 29.1.2. If we set $\alpha = \frac{1}{2}$ in Equation (29.11) we get the **bilinear method**, which preserves the stability of the

1 system [Zhang2007]. The result is
2

$$\underline{3} \quad \underline{z}_t = \bar{\mathbf{A}}\underline{z}_{t-1} + \bar{\mathbf{B}}\underline{u}_t \quad (29.35)$$

$$\underline{4} \quad \underline{y}_t = \bar{\mathbf{C}}\underline{z}_t \quad (29.36)$$

$$\underline{5} \quad \bar{\mathbf{A}} = (\mathbf{I} - \Delta/2 \cdot \mathbf{A})^{-1}(\mathbf{I} + \Delta/ \cdot \mathbf{A}) \in \mathbb{R}^{N \times N} \quad (29.37)$$

$$\underline{6} \quad \bar{\mathbf{B}} = (\mathbf{I} - \Delta/2 \cdot \mathbf{A})^{-1}\Delta\mathbf{B} \in \mathbb{R}^N \quad (29.38)$$

$$\underline{7} \quad \bar{\mathbf{C}} = \mathbf{C} \in \mathbb{R}^{N \times 1} \quad (29.39)$$

10 We now discuss what is happening inside the LSSL layer. Let us assume the initial state is $\underline{z}_{-1} = \mathbf{0}$.
11 We can unroll the recursion to get

$$\underline{13} \quad \underline{z}_0 = \bar{\mathbf{B}}\underline{u}_0, \quad \underline{z}_1 = \bar{\mathbf{A}}\bar{\mathbf{B}}\underline{u}_0 + \bar{\mathbf{B}}\underline{u}_1, \quad \underline{z}_2 = \bar{\mathbf{A}}^2\bar{\mathbf{B}}\underline{u}_0 + \bar{\mathbf{A}}\bar{\mathbf{B}}\underline{u}_1 + \bar{\mathbf{B}}\underline{u}_2, \dots \quad (29.40)$$

$$\underline{14} \quad \underline{y}_0 = \bar{\mathbf{C}}\bar{\mathbf{B}}\underline{u}_0, \quad \underline{y}_1 = \bar{\mathbf{C}}\bar{\mathbf{A}}\bar{\mathbf{B}}\underline{u}_0 + \bar{\mathbf{C}}\bar{\mathbf{B}}\underline{u}_1, \quad \underline{y}_2 = \bar{\mathbf{C}}\bar{\mathbf{A}}^2\bar{\mathbf{B}}\underline{u}_0 + \bar{\mathbf{C}}\bar{\mathbf{A}}\bar{\mathbf{B}}\underline{u}_1 + \bar{\mathbf{C}}\bar{\mathbf{B}}\underline{u}_2, \dots \quad (29.41)$$

17 We see that \underline{z}_t is computing a weighted sum of all the past inputs, where the weights are controlled
18 by powers of the \mathbf{A} matrix. (See also the discussion of subspace identification techniques in
19 Main Section 29.8.2.) It turns out that we can define \mathbf{A} to have a special structure, known as
20 **Hippo** (High-order Polynomial Projection Operator) [hippo], such that (1) it ensures \underline{z}_t embeds
21 “relevant” parts of the past history in a compact manner, (2) enables recursive computation of $\underline{z}_{1:L}$
22 in $O(N)$ time per step, instead of the naive $O(N^2)$ time for the matrix vector multiply; and (3) only
23 has $O(3N)$ (complex-valued) parameters to learn.

24 However, recursive computation still takes time linear in L . At training time, when all inputs to
25 each location are already available, we can further speed thing up by recognizing that the output
26 sequence can be computed in parallel using a convolution:

$$\underline{28} \quad \underline{y}_k = \bar{\mathbf{C}}\bar{\mathbf{A}}^k\bar{\mathbf{B}}\underline{u}_0 + \bar{\mathbf{C}}\bar{\mathbf{A}}^{k-1}\bar{\mathbf{B}}\underline{u}_1 + \dots + \bar{\mathbf{C}}\bar{\mathbf{B}}\underline{u}_k \quad (29.42)$$

$$\underline{29} \quad \underline{y} = \bar{\mathbf{K}} \odot \underline{u} \quad (29.43)$$

$$\underline{30} \quad \bar{\mathbf{K}} = (\bar{\mathbf{C}}\bar{\mathbf{B}}, \bar{\mathbf{C}}\bar{\mathbf{A}}\bar{\mathbf{B}}, \dots, \bar{\mathbf{C}}\bar{\mathbf{A}}^{L-1}\bar{\mathbf{B}}) \quad (29.44)$$

33 We can compute the convolution kernel $\bar{\mathbf{K}}$ matrix in $O(N + L)$ time and space, using the S4
34 representation, and then use FFT to efficiently compute the output. Unfortunately the details of
35 how to do this are rather complicated, so we refer the reader to [S4].

36 Once we have constructed an LSSL layer, we can process a stack of H sequences independently
37 in parallel by replicating the above process with different parameters, for $h = 1 : H$. If we let each
38 of the H \mathbf{C} matrices be of size $\mathbf{N} \times \mathbf{M}$, so they return a vector of channels instead of a scalar at
39 each location, the overall mapping is from $\underline{u}_{1:L} \in \mathbb{R}^{H \times L}$ to $\underline{y}_{1:L} \in \mathbb{R}^{HM \times L}$. We can add a pointwise
40 nonlinearity to the output and then apply a projection matrix $\mathbf{W} \in \mathbb{R}^{MH \times H}$ to linearly combine the
41 channels and map the result back to size $\underline{y}_{1:L} \in \mathbb{R}^{H \times L}$, as shown in Figure 29.6. This overall block
42 can then be repeated a desired number of times. The input to the whole model is an encoder matrix
43 which embeds each token, and the output is a decoder that creates the softmax layer at each location,
44 as in the transformer. We can now learn the \mathbf{A} , \mathbf{B} , \mathbf{C} , \mathbf{D} , and \mathbf{W} matrices (as well as the step size
45 Δ) for each layer using backpropagation, using whatever loss function we want on the output layer.
46

30 Graph learning

30.1 Latent variable models for graphs

Graphs arise in many application areas, such as modeling social networks, protein-protein interaction networks, or patterns of disease transmission between people or animals. There are usually two primary goals when analyzing such data: first, try to discover some “interesting structure” in the graph, such as clusters or communities; second, try to predict which links might occur in the future (e.g., who will make friends with whom). In this section, we focus on the former. More precisely, we will consider a variety of latent variable models for observed graphs.

30.1.1 Stochastic block model

In Figure 30.1(a) we show a directed graph on 9 nodes. There is no apparent structure. However, if we look more deeply, we see it is possible to partition the nodes into three groups or blocks, $B_1 = \{1, 4, 6\}$, $B_2 = \{2, 3, 5, 8\}$, and $B_3 = \{7, 9\}$, such that most of the connections go from nodes in B_1 to B_2 , or from B_2 to B_3 , or from B_3 to B_1 . This is illustrated in Figure 30.1(b).

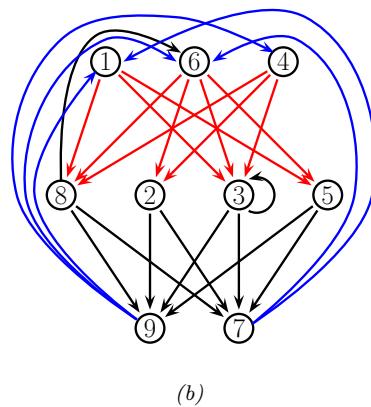
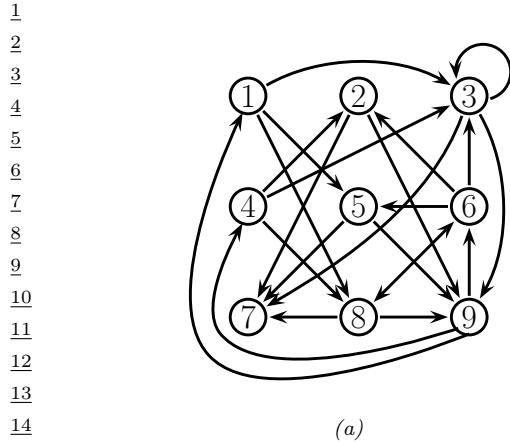
The problem is easier to understand if we plot the adjacency matrices. Figure 30.2(a) shows the matrix for the graph with the nodes in their original ordering. Figure 30.2(b) shows the matrix for the graph with the nodes in their permuted ordering. It is clear that there is block structure.

We can make a generative model of block structured graphs as follows. First, for every node, sample a latent block $q_i \sim \text{Cat}(\boldsymbol{\pi})$, where π_k is the probability of choosing block k , for $k = 1 : K$. Second, choose the probability of connecting group a to group b , for all pairs of groups; let us denote this probability by $\eta_{a,b}$. This can come from a beta prior. Finally, generate each edge R_{ij} using the following model:

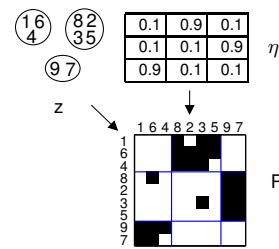
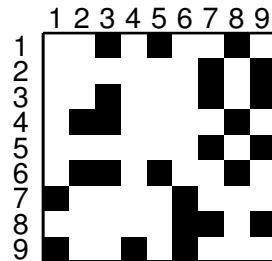
$$p(R_{ij} = r | q_i = a, q_j = b, \boldsymbol{\eta}) = \text{Ber}(r | \eta_{a,b}) \quad (30.1)$$

This is called the **stochastic block model** [Nowicki01]. Figure 30.4(a) illustrates the model as a DGM, and Figure 30.2 illustrates how this model can be used to cluster the nodes in our example.

Note that this is quite different from a conventional clustering problem. For example, we see that all the nodes in block 3 are grouped together, even though there are no connections between them. What they share is the property that they “like to” connect to nodes in block 1, and to receive connections from nodes in block 2. Figure 30.3 illustrates the power of the model for generating many different kinds of graph structure. For example, some social networks have hierarchical structure, which can be modeled by clustering people into different social strata, whereas others consist of a set of cliques.



16 Figure 30.1: (a) A directed graph. (b) The same graph, with the nodes partitioned into 3 groups, making the
17 block structure more apparent.



33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

Figure 30.2: (a) Adjacency matrix for the graph in Figure 30.1(a). (b) Rows and columns are shown permuted to show the block structure. We also show how the stochastic block model can generate this graph. From Figure 1 of [Kemp06]. Used with kind permission of Charles Kemp.

Unlike a standard mixture model, it is not possible to fit this model using exact EM, because all the latent q_i variables become correlated. However, one can use variational EM [Airoldi08], collapsed Gibbs sampling [Kemp06], etc. We omit the details (which are similar to the LDA case).

In [Kemp06], they lifted the restriction that the number of blocks K be fixed, by replacing the Dirichlet prior on π by a Dirichlet process (see Supplementary ??). This is known as the infinite relational model. See Section 30.1.3 for details.

If we have features associated with each node, we can make a discriminative version of this model, for example by defining

$$p(R_{ij} = r | q_i = a, q_j = b, \mathbf{x}_i, \mathbf{x}_j, \boldsymbol{\theta}) = \text{Ber}(r | \mathbf{w}_{a,b}^T f(\mathbf{x}_i, \mathbf{x}_j)) \quad (30.2)$$

where $f(\mathbf{x}_i, \mathbf{x}_j)$ is some way of combining the feature vectors. For example, we could use concatenation, $[\mathbf{x}_i, \mathbf{x}_j]$, or elementwise product $\mathbf{x}_i \otimes \mathbf{x}_j$ as in supervised LDA. The overall model is like a relational

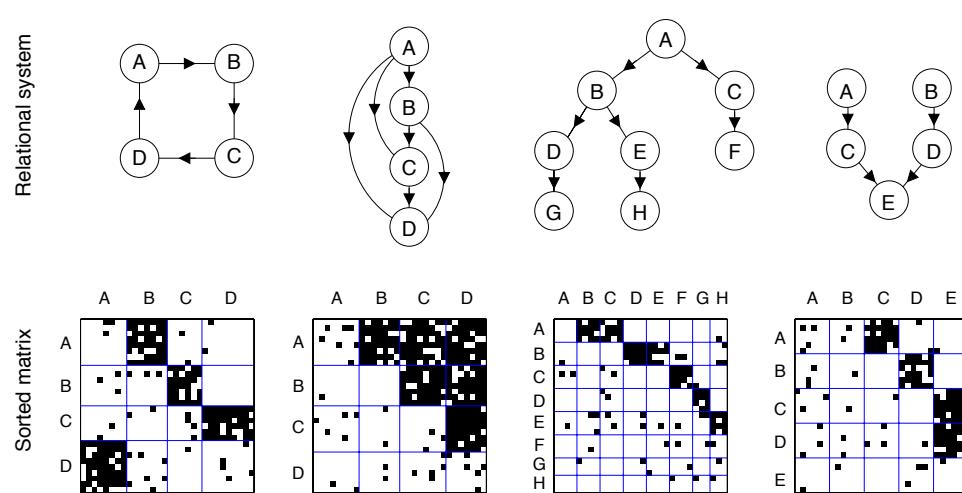


Figure 30.3: Some examples of graphs generated using the stochastic block model with different kinds of connectivity patterns between the blocks. The abstract graph (between blocks) represent a ring, a dominance hierarchy, a common-cause structure, and a common-effect structure. From Figure 4 of [Kemp10]. Used with kind permission of Charles Kemp.

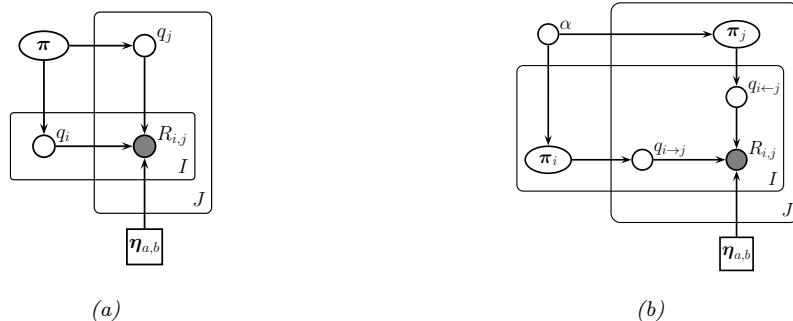


Figure 30.4: (a) Stochastic block model. (b) Mixed membership stochastic block model.

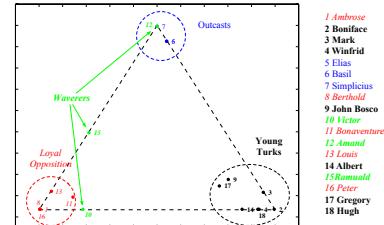
extension of the mixture of experts model.

30.1.2 Mixed membership stochastic block model

In [Airoldi08], they lifted the restriction that each node only belong to one cluster. That is, they replaced $q_i \in \{1, \dots, K\}$ with $\pi_i \in S_K$. This is known as the **mixed membership stochastic block model**, and is similar in spirit to **fuzzy clustering** or **soft clustering**. Note that π_{ik} is not the same as $p(z_i = k | \mathcal{D})$; the former represents **ontological uncertainty** (to what degree does each object belong to a cluster) whereas the latter represents **epistemological uncertainty** (which cluster does an object belong to). If we want to combine epistemological and ontological uncertainty,



(a)



(b)

Figure 30.5: (a) Who-likes-whom graph for Sampson’s monks. (b) Mixed membership of each monk in one of three groups. From Figures 2-3 of [Airoldi08]. Used with kind permission of Edo Airoldi.

we can compute $p(\boldsymbol{\pi}_i | \mathcal{D})$.

In more detail, the generative process is as follows. First, each node picks a distribution over blocks, $\boldsymbol{\pi}_i \sim \text{Dir}(\boldsymbol{\alpha})$. Second, choose the probability of connecting group a to group b , for all pairs of groups, $\eta_{a,b} \sim \beta(\alpha, \beta)$. Third, for each edge, sample two discrete variables, one for each direction:

$$q_{i \rightarrow j} \sim \text{Cat}(\boldsymbol{\pi}_i), q_{i \leftarrow j} \sim \text{Cat}(\boldsymbol{\pi}_j) \quad (30.3)$$

Finally, generate each edge R_{ij} using the following model:

$$p(R_{ij} = 1 | q_{i \rightarrow j} = a, q_{i \leftarrow j} = b, \boldsymbol{\eta}) = \eta_{a,b} \quad (30.4)$$

See Figure 30.4(b) for the DGM.

Unlike the regular stochastic block model, each node can play a different role, depending on who it is connecting to. As an illustration of this, we will consider a dataset that is widely used in the social networks analysis literature. The data concerns who-likes-whom amongst of group of 18 monks. It was collected by hand in 1968 by Sampson [Sampson68] over a period of months. (These days, in the era of social media such as Facebook, a social network with only 18 people is trivially small, but the methods we are discussing can be made to scale.) Figure 30.5(a) plots the raw data, and Figure 30.5(b) plots $\mathbb{E}[\boldsymbol{\pi}]_i$ for each monk, where $K = 3$. We see that most of the monks belong to one of the three clusters, known as the “young turks”, the “outcasts” and the “loyal opposition”. However, some individuals, notably monk 15, belong to two clusters; Sampson called these monks the “waverers”. It is interesting to see that the model can recover the same kinds of insights as Sampson derived by hand.

One prevalent problem in social network analysis is missing data. For example, if $R_{ij} = 0$, it may be due to the fact that person i and j have not had an opportunity to interact, or that data is not available for that interaction, as opposed to the fact that these people don’t want to interact. In other words, *absence of evidence is not evidence of absence*. We can model this by modifying the observation model so that with probability ρ , we generate a 0 from the background model, and we only force the model to explain observed 0s with probability $1 - \rho$. In other words, we robustify the observation model to allow for outliers, as follows:

$$p(R_{ij} = r | q_{i \rightarrow j} = a, q_{i \leftarrow j} = b, \boldsymbol{\eta}) = \rho \delta_0(r) + (1 - \rho) \text{Ber}(r | \eta_{a,b}) \quad (30.5)$$

1 See [Airoldi08] for details.
2

3 30.1.3 Infinite relational model

4 The stochastic block model is defined for graphs, in which each pair of edges may or may not have
5 an edge. We can easily extend this to hyper-graphs, which is useful for modeling relational data. For
6 example, suppose we want to model a family tree. We might write $R_1(i, j, k) = 1$ if adults i and j
7 are the parents of child k , where R_1 is the “parent-of” relation. Here i and j are entities of type T^1
8 (adults), and j is an entity of type T^2 (child), so the type signature of R_1 is $T^1 \times T^1 \times T^2 \rightarrow \{0, 1\}$.
9

10 To define the probability of relations holding between entities, we can associate a latent cluster
11 variable $q_i^t \in \{1, \dots, K_t\}$ with each entity i of each type t . We then define the probability of the
12 relation holding between specific entities by looking up the probability of the relation holding between
13 the corresponding entity clusters. Continuing our example above, we have
14

$$\underline{15} \quad p(R_1(i, j, k)|q_i^1 = a, q_j^1 = b, q_k^2 = c, \boldsymbol{\eta}) = \text{Ber}(R_1(i, j, k)|\eta_{a,b,c}) \quad (30.6)$$

17 We can also have real-valued relations, where each edge has a weight. For example, we can write
18

$$\underline{19} \quad p(R_1(i, j, k)|q_i^1 = a, q_j^1 = b, q_k^2 = c, \boldsymbol{\mu}) = \mathcal{N}(R_1(i, j, k)|\mu_{a,b,c}, \sigma^2), \quad (30.7)$$

21 where $\mu_{a,b,c}$ captures the average response for that group of clusters. We can also add entity-specific
22 offset terms:

$$\underline{23} \quad p(R_1(i, j, k)|q_i^1 = a, q_j^1 = b, q_k^2 = c, \boldsymbol{\mu}) = \mathcal{N}(R_1(i, j, k)|\mu_{a,b,c} + \mu_i + \mu_j + \mu_k, \sigma^2), \quad (30.8)$$

25 This model was proposed in [Banerjee07], who fit the model using an alternating minimization
26 procedure.

27 If we allow the number of clusters K_t for each type of entity to be unbounded, by using a Dirichlet
28 process, the model is called the **infinite relational model** (IRM) [Kemp06], also known as an
29 **infinite hidden relational model** (IHRM) [Xu06]. We can fit this model with variational Bayes
30 [Xu06; Xu07] or collapsed Gibbs sampling [Kemp06]. Rather than go into algorithmic detail, we
31 just sketch some interesting applications.
32

33 30.1.3.1 Learning ontologies

35 An **ontology** refers to an organisation of knowledge. In AI, ontologies are often built by hand (see
36 e.g., [Russell10]), but it is interesting to try and learn them from data. In [Kemp06], they show
37 how this can be done using the IRM.

38 The data comes from the Unified Medical Language System [McCray03], which defines a semantic
39 network with 135 concepts (such as “disease or syndrome”, “diagnostic procedure”, “animal”), and
40 49 binary predicates (such as “affects”, “prevents”). We can represent this as a ternary relation
41 $R : T^1 \times T^1 \times T^2 \rightarrow \{0, 1\}$, where T^1 is the set of concepts and T^2 is the set of binary predicates.
42 The result is a 3d cube. We can then apply the IRM to partition the cube into regions of roughly
43 homogeneous response. The system found 14 concept clusters and 21 predicate clusters. Some of these
44 are shown in Figure 30.6. The system learns, for example, that biological functions affect organisms
45 (since $\eta_{a,b,c} \approx 1$ where a represents the biological function cluster, b represents the organism cluster,
46 and c represents the affects cluster).
47

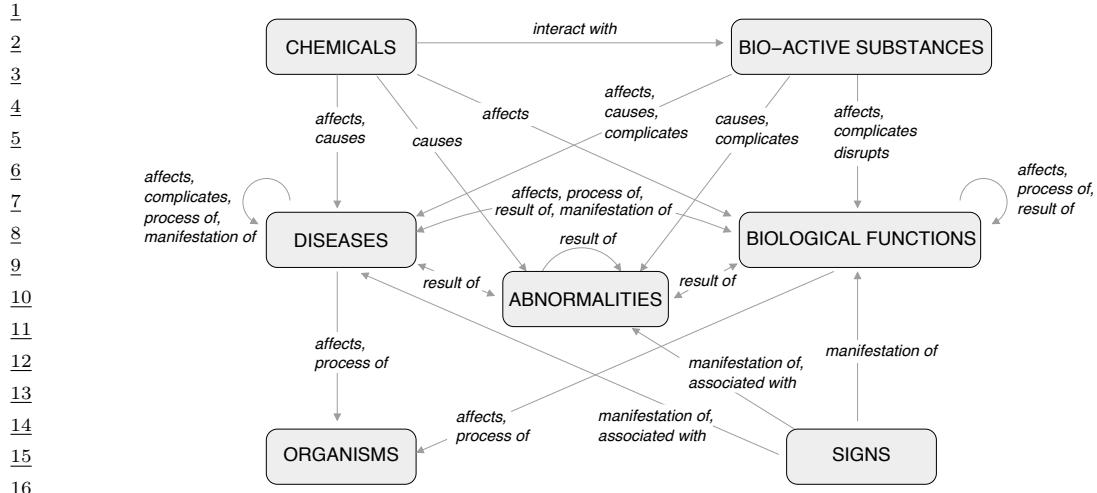


Figure 30.6: Illustration of an ontology learned by IRM applied to the Unified Medical Language System. The boxes represent 7 of the 14 concept clusters. Predicates that belong to the same cluster are grouped together, and associated with edges to which they pertain. All links with weight above 0.8 have been included. From Figure 9 of [Kemp10]. Used with kind permission of Charles Kemp.

30.1.3.2 Clustering based on relations and features

We can also use IRM to cluster objects based on their relations and their features. For example, [Kemp06] consider a political dataset (from 1965) consisting of 14 countries, 54 binary predicates representing interaction types between countries (e.g., “sends tourists to”, “economic aid”), and 90 features (e.g., “communist”, “monarchy”). To create a binary dataset, real-valued features were thresholded at their mean, and categorical variables were dummy-encoded. The data has 3 types: T^1 represents countries, T^2 represents interactions, and T^3 represents features. We have two relations: $R^1 : T^1 \times T^1 \times T^2 \rightarrow \{0, 1\}$, and $R^2 : T^1 \times T^3 \rightarrow \{0, 1\}$. (This problem therefore combines aspects of both the biclustering model and the ontology discovery model.) When given multiple relations, the IRM treats them as conditionally independent. In this case, we have

$$p(\mathbf{R}^1, \mathbf{R}^2 | \mathbf{q}^1, \mathbf{q}^2, \mathbf{q}^3, \boldsymbol{\theta}) = p(\mathbf{R}^1 | \mathbf{q}^1, \boldsymbol{\theta})p(\mathbf{R}^2 | \mathbf{q}^1, \mathbf{q}^3, \boldsymbol{\theta}) \quad (30.9)$$

The results are shown in Figure 30.7. The IRM divides the 90 features into 5 clusters, the first of which contains “noncommunist”, which captures one of the most important aspects of this Cold-War era dataset. It also clusters the 14 countries into 5 clusters, reflecting natural geo-political groupings (e.g., USA and UK, or the Communist Bloc), and the 54 predicates into 18 clusters, reflecting similar relationships (e.g., “negative behavior” and “accusations”).

30.2 Learning tree structures

Since the problem of structure learning for general graphs is NP-hard [Chickering96np], we start by considering the special case of trees. Trees are special because we can learn their structure efficiently,

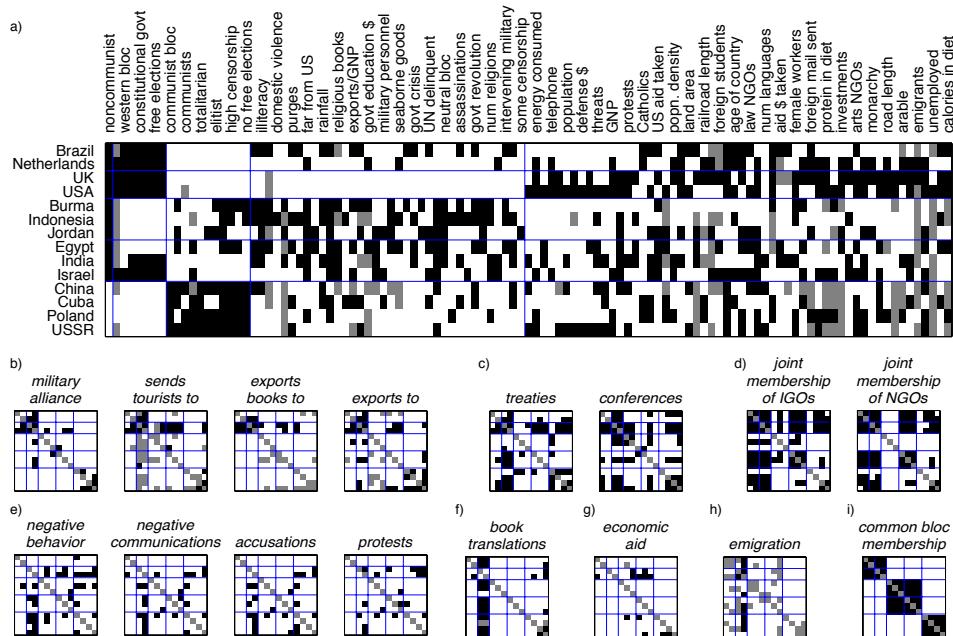


Figure 30.7: Illustration of IRM applied to some political data containing features and pairwise interactions. Top row (a): the partition of the countries into 5 clusters and the features into 5 clusters. Every second column is labelled with the name of the corresponding feature. Small squares at bottom (b-i): these are 8 of the 18 clusters of interaction types. From Figure 6 of [Kemp06]. Used with kind permission of Charles Kemp.

as we discuss below, and because, once we have learned the tree, we can use them for efficient exact inference, as discussed in Main Section 9.3.2.

30.2.1 Chow-Liu algorithm

In this section, we consider undirected trees with pairwise potentials. The likelihood can be represented as follows:

$$p(\mathbf{x}|T) = \prod_{t \in V} p(x_t) \prod_{(s,t) \in E} \frac{p(x_s, x_t)}{p(x_s)p(x_t)} \quad (30.10)$$

where $p(x_s, x_t)$ is an edge marginal and $p(x_t)$ is a node marginal. Hence we can write the log-likelihood for a tree as follows:

$$\begin{aligned} \log p(\mathcal{D}|\boldsymbol{\theta}, T) &= \sum_t \sum_k N_{tk} \log p(x_t = k|\boldsymbol{\theta}) \\ &\quad + \sum_{s,t} \sum_{j,k} N_{stjk} \log \frac{p(x_s = j, x_t = k|\boldsymbol{\theta})}{p(x_s = j|\boldsymbol{\theta})p(x_t = k|\boldsymbol{\theta})} \end{aligned} \quad (30.11)$$

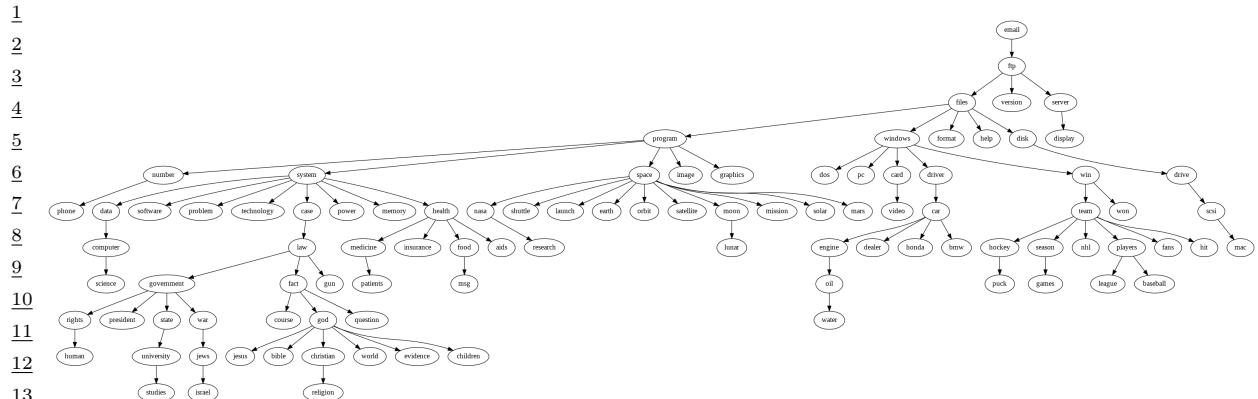


Figure 30.8: The MLE tree estimated from the 20-newsgroup data. Generated by chow_liu_tree_demo.ipynb.

¹⁸ where N_{stjk} is the number of times node s is in state j and node t is in state k , and N_{tk} is the number
¹⁹ of times node t is in state k . We can rewrite these counts in terms of the empirical distribution:
²⁰ $N_{stjk} = Np_{\mathcal{D}}(x_s = j, x_t = k)$ and $N_{tk} = Np_{\mathcal{D}}(x_t = k)$. Setting θ to the MLEs, this becomes
²¹ $N_{stjk} = Np_{\mathcal{D}}(x_s = j, x_t = k)$ and $N_{tk} = Np_{\mathcal{D}}(x_t = k)$. Setting θ to the MLEs, this becomes

$$\frac{\log p(\mathcal{D}|\theta, T)}{N} = \sum_{t \in \mathcal{V}} \sum_k p_{\mathcal{D}}(x_t = k) \log p_{\mathcal{D}}(x_t = k) \quad (30.12)$$

$$+ \sum_{(s,t) \in \mathcal{E}(T)} \mathbb{I}(x_s, x_t | \hat{\theta}_{st}) \quad (30.13)$$

where $\mathbb{I}(x_s, x_t | \hat{\theta}_{st}) \geq 0$ is the mutual information between x_s and x_t given the empirical distribution:

$$\mathbb{I}(x_s, x_t | \hat{\theta}_{st}) = \sum_j \sum_k p_{\mathcal{D}}(x_s = j, x_t = k) \log \frac{p_{\mathcal{D}}(x_s = j, x_t = k)}{p_{\mathcal{D}}(x_s = j)p_{\mathcal{D}}(x_t = k)} \quad (30.14)$$

³⁵ Since the first term in Equation (30.13) is independent of the topology T , we can ignore it when learning structure. Thus the tree topology that maximizes the likelihood can be found by computing the maximum weight spanning tree, where the edge weights are the pairwise mutual informations, $\mathbb{I}(y_s, y_t | \theta_{st})$. This is called the **Chow-Liu algorithm** [Chow68].

39 There are several algorithms for finding a max spanning tree (MST). The two best known are
 40 Prim's algorithm and Kruskal's algorithm. Both can be implemented to run in $O(E \log V)$ time,
 41 where $E = V^2$ is the number of edges and V is the number of nodes. See e.g., [Sedgewick11] for
 42 details. Thus the overall running time is $O(NV^2 + V^2 \log V)$, where the first term is the cost of
 43 computing the sufficient statistics.

⁴⁴ Figure 30.8 gives an example of the method in action, applied to the binary 20 newsgroups data
⁴⁵ shown in Main Figure 5.8. The tree has been arbitrarily rooted at the node representing “email”.
⁴⁶ The connections that are learned seem intuitively reasonable.

30.2.2 Finding the MAP forest

Since all trees have the same number of parameters, we can safely use the maximum likelihood score as a model selection criterion without worrying about overfitting. However, sometimes we may want to fit a **forest** rather than a single tree, since inference in a forest is much faster than in a tree (we can run belief propagation in each tree in the forest in parallel). The MLE criterion will never choose to omit an edge. However, if we use the marginal likelihood or a penalized likelihood (such as BIC), the optimal solution may be a forest. Below we give the details for the marginal likelihood case.

In Section 30.3.3.2, we explain how to compute the marginal likelihood of any DAG using a Dirichlet prior for the CPTs. The resulting expression can be written as follows:

$$\log p(\mathcal{D}|T) = \sum_{t \in \mathcal{V}} \log \int \prod_{i=1}^N p(x_{it} | \mathbf{x}_{i,\text{pa}(t)} | \boldsymbol{\theta}_t) p(\boldsymbol{\theta}_t) d\boldsymbol{\theta}_t = \sum_t \text{score}(\mathbf{N}_{t,\text{pa}(t)}) \quad (30.15)$$

where $\mathbf{N}_{t,\text{pa}(t)}$ are the counts (sufficient statistics) for node t and its parents, and score is defined in Equation (30.28).

Now suppose we only allow DAGs with at most one parent. Following [Heckerman95c], let us associate a weight with each $s \rightarrow t$ edge, $w_{s,t} \triangleq \text{score}(t|s) - \text{score}(t|0)$, where $\text{score}(t|0)$ is the score when t has no parents. Note that the weights might be negative (unlike the MLE case, where edge weights are always non-negative because they correspond to mutual information). Then we can rewrite the objective as follows:

$$\log p(\mathcal{D}|T) = \sum_t \text{score}(t|\text{pa}(t)) = \sum_t w_{\text{pa}(t),t} + \sum_t \text{score}(t|0) \quad (30.16)$$

The last term is the same for all trees T , so we can ignore it. Thus finding the most probable tree amounts to finding a **maximal branching** in the corresponding weighted directed graph. This can be found using the algorithm in [Gabow84].

If the scoring function is prior and likelihood equivalent (these terms are explained in Section 30.3.3.3), we have

$$\text{score}(s|t) + \text{score}(t|0) = \text{score}(t|s) + \text{score}(s|0) \quad (30.17)$$

and hence the weight matrix is symmetric. In this case, the maximal branching is the same as the maximal weight forest. We can apply a slightly modified version of the MST algorithm to find this [Edwards10]. To see this, let $G = (V, E)$ be a graph with both positive and negative edge weights. Now let G' be a graph obtained by omitting all the negative edges from G . This cannot reduce the total weight, so we can find the maximum weight forest of G by finding the MST for each connected component of G' . We can do this by running Kruskal's algorithm directly on G' : there is no need to find the connected components explicitly.

30.2.3 Mixtures of trees

A single tree is rather limited in its expressive power. Later in this chapter we discuss ways to learn more general graphs. However, the resulting graphs can be expensive to do inference in. An interesting alternative is to learn a **mixture of trees** [Meila00b], where each mixture component may have a different tree topology. This is like an unsupervised version of the TAN classifier discussed

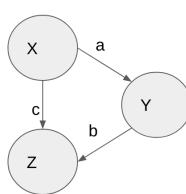


Figure 30.9: A simple linear Gaussian model. .

in Main Section 4.2.8.3. We can fit a mixture of trees by using EM: in the E step, we compute the responsibilities of each cluster for each data point, and in the M step, we use a weighted version of the Chow-Liu algorithm. See [Meila00b] for details.

In fact, it is possible to create an “infinite mixture of trees”, by integrating out over all possible trees. Remarkably, this can be done in N_G^3 time using the matrix tree theorem. This allows us to perform exact Bayesian inference of posterior edge marginals etc. However, it is not tractable to use this infinite mixture for inference of hidden nodes. See [Meila06] for details.

30.3 Learning DAG structures

In this section, we discuss how to estimate the structure of directed graphical models from observational data. This is often called **Bayes net structure learning**. We can only do this if we make the faithfulness assumption, which we explain in Section 30.3.1. Furthermore our output will be a set of equivalent DAGs, rather than a single unique DAG, as we explain in Section 30.3.2. After introducing these restrictions, we discuss some statistical and algorithmic techniques. If the DAG is interpreted causal, these techniques can be used for **causal discovery**, although this relies on additional assumptions about non-confounding. For more details, see e.g., [Glymour2019].

30.3.1 Faithfulness

The Markov assumption allows us to infer CI properties of a distribution p from a graph G . To go in the opposite direction, we need to assume that the generating distribution p is **faithful** to the generating DAG G . This means that all the conditional independence (CI) properties of p are exactly captured by the graphical structure, so $I(p) = I(G)$; this means there cannot be any CI properties in p that are due to particular settings of the parameters (such as zeros in a regression matrix) that are not graphically explicit. (For this reason, a faithful distribution is also called a **stable** distribution.)

Let us consider an example of a non-faithful distribution (from [Peters2017]). Consider a linear Gaussian model of the form

$$X = E_X, \quad E_X \sim \mathcal{N}(0, \sigma_X^2) \tag{30.18}$$

$$Y = aX + E_Y, \quad E_Y \sim \mathcal{N}(0, \sigma_Y^2) \tag{30.19}$$

$$Z = bY + cX + E_Z, \quad E_Z \sim \mathcal{N}(0, \sigma_Z^2) \tag{30.20}$$

where the error terms are independent. If $ab + c = 0$, then $X \perp Z$, even though this is not implied by the DAG in Figure 30.9. Fortunately, this kind of accidental cancellation happens with zero

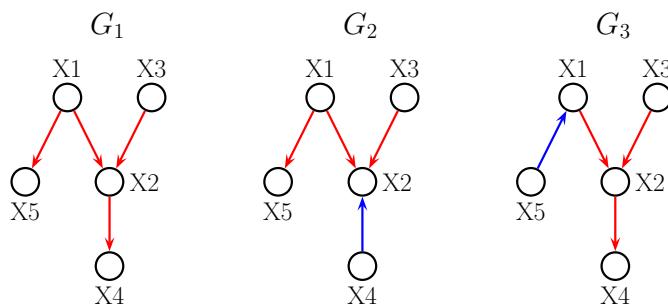


Figure 30.10: Three DAGs. G_1 and G_3 are Markov equivalent, G_2 is not.

probability if the coefficients are drawn randomly from positive densities [SpirtesBook].

30.3.2 Markov equivalence

Even with the faithfulness assumption, we cannot always uniquely identify a DAG from a joint distribution. To see this, consider the following 3 DGMs: $X \rightarrow Y \rightarrow Z$, $X \leftarrow Y \leftarrow Z$ and $X \leftarrow Y \rightarrow Z$. These all represent the same set of CI statements, namely

$$X \perp Z|Y, \quad X \not\perp Z \quad (30.21)$$

We say these graphs are **Markov equivalent**, since they encode the same set of CI assumptions. That is, they all belong to the same Markov **equivalence class**. However, the DAG $X \rightarrow Y \leftarrow Z$ encodes $X \perp Z$ and $X \not\perp Z|Y$, so corresponds to a different distribution.

In [Verma90], they prove the following theorem.

Theorem 30.3.1. *Two structures are Markov equivalent iff they have the same skeleton, i.e., the have the same edges (disregarding direction) and they have the same set of v-structures (colliders whose parents are not adjacent).*

For example, referring to Figure 30.10, we see that $G_1 \not\equiv G_2$, since reversing the $2 \rightarrow 4$ arc creates a new v-structure. However, $G_1 \equiv G_3$, since reversing the $1 \rightarrow 5$ arc does not create a new v-structure.

We can represent a Markov equivalence class using a single **partially directed acyclic graph** or **PDAG** (also called an **essential graph** or **pattern**), in which some edges are directed and some undirected (see Main Section 4.5.4.1). The undirected edges represent reversible edges; any combination is possible so long as no new v-structures are created. The directed edges are called **compelled edges**, since changing their orientation would change the v-structures and hence change the equivalence class. For example, the PDAG $X - Y - Z$ represents $\{X \rightarrow Y \rightarrow Z, X \leftarrow Y \leftarrow Z, X \leftarrow Y \rightarrow Z\}$ which encodes $X \not\perp Z$ and $X \perp Z|Y$. See Figure 30.10 for another example.

The significance of the above theorem is that, when we learn the DAG structure from data, we will not be able to uniquely identify all of the edge directions, even given an infinite amount of data. We say that we can learn DAG structure “up to Markov equivalence”. This also cautions us

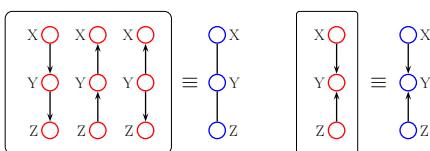


Figure 30.11: PDAG representation of Markov equivalent DAGs.

not to read too much into the meaning of particular edge orientations, since we can often change them without changing the model in any observable way. (If we want to distinguish between edge orientations within a PDAG (e.g., if we want to imbue a causal interpretation on the edges), we can use interventional data, as we discuss in Section 30.5.2.)

30.3.3 Bayesian model selection: statistical foundations

In this section, we discuss how to compute the exact posterior over graphs, $p(G|\mathcal{D})$, ignoring for now the issue of computational tractability. We assume there is no missing data, and that there are no hidden variables. This is called the **complete data assumption**.

For simplicity, we will focus on the case where all the variables are categorical and all the CPDs are tables. Our presentation is based in part on [Heckerman95c], although we will follow the notation of Main Section 4.2.7.3. In particular, let $x_{it} \in \{1, \dots, K_t\}$ be the value of node t in case i , where K_t is the number of states for node t . Let $\theta_{tck} \triangleq p(x_{it} = k | \mathbf{x}_{\text{pa}(t)} = c)$, for $k = 1 : K_t$, and $c = 1 : C_t$, where C_t is the number of parent combinations (possible conditioning cases). For notational simplicity, we will often assume $K_t = K$, so all nodes have the same number of states. We will also let $d_t = \dim(\text{pa}(t))$ be the degree or fan-in of node t , so that $C_t = K^{d_t}$.

30.3.3.1 Deriving the likelihood

Assuming there is no missing data, and that all CPDs are tabular, the likelihood can be written as follows:

$$p(\mathcal{D}|G, \boldsymbol{\theta}) = \prod_{i=1}^N \prod_{t=1}^{N_G} \text{Cat}(x_{it} | \mathbf{x}_{i,\text{pa}(t)}, \boldsymbol{\theta}_t) \quad (30.22)$$

$$= \prod_{i=1}^N \prod_{t=1}^{N_G} \prod_{c=1}^{C_t} \prod_{k=1}^{K_t} \theta_{tck}^{\mathbb{I}(x_{i,t} = k, \mathbf{x}_{i,\text{pa}(t)} = c)} = \prod_{t=1}^{N_G} \prod_{c=1}^{C_t} \prod_{k=1}^{K_t} \theta_{tck}^{N_{tck}} \quad (30.23)$$

where N_{tck} is the number of times node t is in state k and its parents are in state c . (Technically these counts depend on the graph structure G , but we drop this from the notation.)

30.3.3.2 Deriving the marginal likelihood

Choosing the graph with the maximum likelihood will always pick a fully connected graph (subject to the acyclicity constraint), since this maximizes the number of parameters. To avoid such overfitting,

we will choose the graph with the maximum marginal likelihood, $p(\mathcal{D}|G)$, where we integrate out the parameters; the magic of the Bayesian Occam's razor (Main Section 3.8.1) will then penalize overly complex graphs.

To compute the marginal likelihood, we need to specify priors on the parameters. We will make two standard assumptions. First, we assume **global prior parameter independence**, which means $p(\boldsymbol{\theta}) = \prod_{t=1}^{N_G} p(\boldsymbol{\theta}_t)$. Second, we assume **local prior parameter independence**, which means $p(\boldsymbol{\theta}_t) = \prod_{c=1}^{C_t} p(\boldsymbol{\theta}_{tc})$ for each t . It turns out that these assumptions imply that the prior for each row of each CPT must be a Dirichlet [Geiger97], that is, $p(\boldsymbol{\theta}_{tc}) = \text{Dir}(\boldsymbol{\theta}_{tc}|\boldsymbol{\alpha}_{tc})$. Given these assumptions, and using the results of Main Equation (3.94), we can write down the marginal likelihood of any DAG as follows:

$$p(\mathcal{D}|G) = \prod_{t=1}^{N_G} \prod_{c=1}^{C_t} \int \left[\prod_{i:x_i,\text{pa}(t)=c} \text{Cat}(x_{it}|\boldsymbol{\theta}_{tc}) \right] \text{Dir}(\boldsymbol{\theta}_{tc}) d\boldsymbol{\theta}_{tc} \quad (30.24)$$

$$= \prod_{t=1}^{N_G} \prod_{c=1}^{C_t} \frac{B(\mathbf{N}_{tc} + \boldsymbol{\alpha}_{tc})}{B(\boldsymbol{\alpha}_{tc})} \quad (30.25)$$

$$= \prod_{t=1}^{N_G} \prod_{c=1}^{C_t} \frac{\Gamma(N_{tc})}{\Gamma(N_{tc} + \alpha_{tc})} \prod_{k=1}^{K_t} \frac{\Gamma(N_{tck} + \alpha_{tck}^G)}{\Gamma(\alpha_{tck}^G)} \quad (30.26)$$

$$= \prod_{t=1}^{N_G} \text{score}(\mathbf{N}_{t,\text{pa}(t)}) \quad (30.27)$$

where $N_{tc} = \sum_k N_{tck}$, $\alpha_{tc} = \sum_k \alpha_{tck}$, $\mathbf{N}_{t,\text{pa}(t)}$ is the vector of counts (sufficient statistics) for node t and its parents, and $\text{score}()$ is a local scoring function defined by

$$\text{score}(\mathbf{N}_{t,\text{pa}(t)}) \triangleq \prod_{c=1}^{C_t} \frac{B(\mathbf{N}_{tc} + \boldsymbol{\alpha}_{tc})}{B(\boldsymbol{\alpha}_{tc})} \quad (30.28)$$

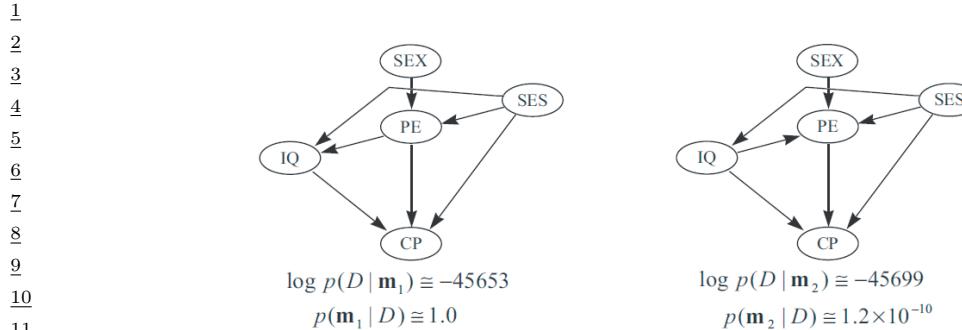
We say that the marginal likelihood **decomposes** or factorizes according to the graph structure.

30.3.3.3 Setting the prior

How should we set the hyper-parameters α_{tck} ? It is tempting to use a Jeffreys prior of the form $\alpha_{tck} = \frac{1}{2}$ (Main Equation (3.233)). However, it turns out that this violates a property called **likelihood equivalence**, which is sometimes considered desirable. This property says that if G_1 and G_2 are Markov equivalent (Section 30.3.2), they should have the same marginal likelihood, since they are essentially equivalent models. Geiger97 proved that, for complete graphs, the only prior that satisfies likelihood equivalence and parameter independence is the Dirichlet prior, where the pseudo counts have the form

$$\alpha_{tck} = \alpha p_0(x_t = k, \mathbf{x}_{\text{pa}(t)} = c) \quad (30.29)$$

where $\alpha > 0$ is called the **equivalent sample size**, and p_0 is some prior joint probability distribution. This is called the **BDe** prior, which stands for Bayesian Dirichlet likelihood equivalent.



13 *Figure 30.12: The two most probable DAGs learned from the Sewell-Shah data. From [Heckerman97]. Used
14 with kind permission of David Heckerman*

18 To derive the hyper-parameters for other graph structures, **Geiger97** invoked an additional
19 assumption called **parameter modularity**, which says that if node X_t has the same parents in G_1
20 and G_2 , then $p(\theta_t | G_1) = p(\theta_t | G_2)$. With this assumption, we can always derive α_t for a node t in
21 any other graph by marginalizing the pseudo counts in Equation (30.29).

22 Typically the prior distribution p_0 is assumed to be uniform over all possible joint configurations.
23 In this case, we have $\alpha_{tck} = \frac{\alpha}{K_t C_t}$, since $p_0(x_t = k, \mathbf{x}_{\text{pa}(t)} = c) = \frac{1}{K_t C_t}$. Thus if we sum the pseudo
24 counts over all $C_t \times K_t$ entries in the CPT, we get a total equivalent sample size of α . This is called
25 the **BDeu** prior, where the “u” stands for uniform. This is the most widely used prior for learning
26 Bayes net structures. For advice on setting the global tuning parameter α , see [Silander07].
27

28 30.3.3.4 Example: analysis of the college plans dataset

30 We now consider a larger example from [Heckerman97], who analyzed a dataset of 5 variables,
31 related to the decision of high school students about whether to attend college. Specifically, the
32 variables are as follows:

- 33 • Sex: Male or female
- 34 • SES: Socio economic status: low, lower middle, upper middle or high.
- 35 • IQ: Intelligence quotient: discretized into low, lower middle, upper middle or high.
- 36 • PE: Parental encouragment: low or high
- 37 • CP: College plans: yes or no.

38 These variables were measured for 10,318 Wisconsin high school seniors. There are $2 \times 4 \times 4 \times 2 \times 128$ possible joint configurations.
39

40 Heckerman et al. computed the exact posterior over all 29,281 possible 5 node DAGs, except for
41 ones in which SEX and/or SES have parents, and/or CP have children. (The prior probability of
42 these graphs was set to 0, based on domain knowledge.) They used the BDeu score with $\alpha = 5$,
43 although they said that the results were robust to any α in the range 3 to 40. The top two graphs are
44 shown in Figure 30.12. We see that the most probable one has approximately all of the probability
45 mass, so the posterior is extremely peaked.

46 It is tempting to interpret this graph in terms of causality (see Main Chapter 36 for a detailed
47

discussion of this topic). In particular, it seems that socio-economic status, IQ and parental encouragement all causally influence the decision about whether to go to college, which makes sense. Also, sex influences college plans only indirectly through parental encouragement, which also makes sense. However, the direct link from socio economic status to IQ seems surprising; this may be due to a hidden common cause. In Section 30.3.8.5 we will re-examine this dataset allowing for the presence of hidden variables.

30.3.3.5 Marginal likelihood for non-tabular CPDs

If all CPDs are linear Gaussian, we can replace the Dirichlet-multinomial model with the normal-gamma model, and thus derive a different exact expression for the marginal likelihood. See [Geiger94] for the details. In fact, we can easily combine discrete nodes and Gaussian nodes, as long as the discrete nodes always have discrete parents; this is called a **conditional Gaussian** DAG. Again, we can compute the marginal likelihood in closed form. See [Bottcher03] for the details.

In the general case (i.e., everything except Gaussians and CPTs), we need to approximate the marginal likelihood. The simplest approach is to use the BIC approximation, which has the form

$$\sum_t \log p(\mathcal{D}_t | \hat{\theta}_t) - \frac{K_t C_t}{2} \log N \quad (30.30)$$

30.3.4 Bayesian model selection: algorithms

In this section, we discuss some algorithms for approximately computing the mode of (or samples from) the posterior $p(G|D)$.

30.3.4.1 The K2 algorithm for known node orderings

Suppose we know a total ordering of the nodes. Then we can compute the distribution over parents for each node independently, without the risk of introducing any directed cycles: we simply enumerate over all possible subsets of ancestors and compute their marginal likelihoods. If we just return the best set of parents for each node, we get the the **K2 algorithm** [Cooper92]. In this case, we can find the best set of parents for each node using ℓ_1 -regularization, as shown in [Schmidt07aaai].

30.3.4.2 Dynamic programming algorithms

In general, the ordering of the nodes is not known, so the posterior does not decompose. Nevertheless, we can use dynamic programming to find the globally optimal MAP DAG (up to Markov equivalence), as shown in [Koivisto04; Silander06].

If our goal is knowledge discovery, the MAP DAG can be misleading, for reasons we discussed in Main Section 7.4.1. A better approach is to compute the marginal probability that each edge is present, $p(G_{st} = 1|\mathcal{D})$. We can also compute these quantities using dynamic programming, as shown in [Koivisto06; Parviainen11ancestor].

Unfortunately, all of these methods take $N_G 2^{N_G}$ time in the general case, making them intractable for graphs with more than about 16 nodes.

1 **30.3.4.3 Scaling up to larger graphs**

3 The main challenge in computing the posterior over DAGs is that there are so many possible graphs.
4 More precisely, [Robinson73] showed that the number of DAGs on D nodes satisfies the following
5 recurrence:

$$\underline{7} \quad f(D) = \sum_{i=1}^D (-1)^{i+1} \binom{D}{i} 2^{i(D-i)} f(D-i) \quad (30.31)$$

8

10 for $D > 2$. The base case is $f(1) = 1$. Solving this recurrence yields the following sequence: 1, 3, 25,
11 543, 29281, 3781503, etc.¹

12 Indeed, the general problem of finding the globally optimal MAP DAG is provably NP-complete
13 [Chickering96np]. In view of the enormous size of the hypothesis space, we are generally forced to
14 use approximate methods, some of which we review below.

16 **30.3.4.4 Hill climbing methods for approximating the mode**

17 A common way to find an approximate MAP graph structure is to use a greedy hill climbing method.
18 At each step, the algorithm proposes small changes to the current graph, such as adding, deleting or
19 reversing a single edge; it then moves to the neighboring graph which most increases the posterior.
20 The method stops when it reaches a local maximum. It is important that the method only proposes
21 local changes to the graph, since this enables the change in marginal likelihood (and hence the
22 posterior) to be computed in constant time (assuming we cache the sufficient statistics). This is
23 because all but one or two of the terms in Equation (30.25) will cancel out when computing the log
24 Bayes factor $\delta(G \rightarrow G') = \log p(G'|\mathcal{D}) - \log p(G|\mathcal{D})$.

26 We can initialize the search from the best tree, which can be found using exact methods discussed
27 in Section 30.2.1. For speed, we can restrict the search so it only adds edges which are part of the
28 Markov blankets estimated from a dependency network [SchmidtThesis]. Figure 30.13 gives an
29 example of a DAG learned in this way from the 20-newsgroup data. For binary data, it is possible to
30 use techniques from frequent itemset mining to find good Markov blanket candidates, as described in
31 [Goldenberg04].

32 We can use techniques such as multiple random restarts to increase the chance of finding a good
33 local maximum. We can also use more sophisticated local search methods, such as genetic algorithms
34 or simulated annealing, for structure learning. (See also Section 30.3.6 for gradient based techniques
35 based on continuous relaxations.)

36 It is also possible to perform the greedy search in the space of PDAGs instead of in the space of
37 DAGs; this is known as the **greedy equivalence search** method [Chickering02opt]. Although
38 each step is somewhat more complicated, the advantage is that the search space is smaller.

39 **30.3.4.5 Sampling methods**

41 If our goal is knowledge discovery, the MAP DAG can be misleading, for reasons we discussed in
42 Main Section 7.4.1. A better approach is to compute the probability that each edge is present, $p(G_{st} =$

44 1. A longer list of values can be found at <http://www.research.att.com/~njas/sequences/A003024>. Interestingly,
45 the number of DAGs is equal to the number of (0,1) matrices all of whose eigenvalues are positive real numbers
46 [McKay04].

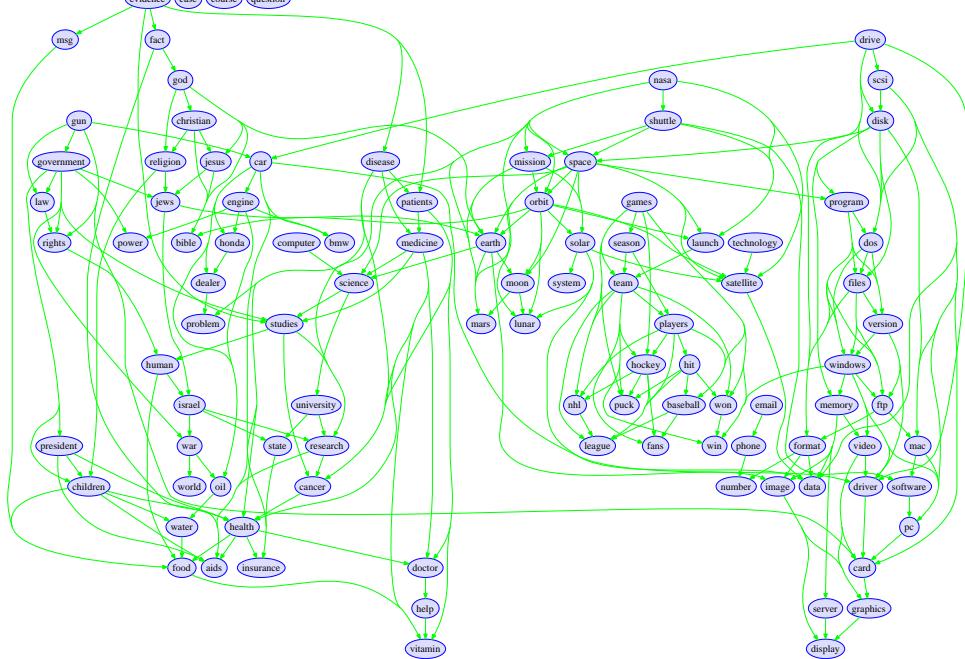


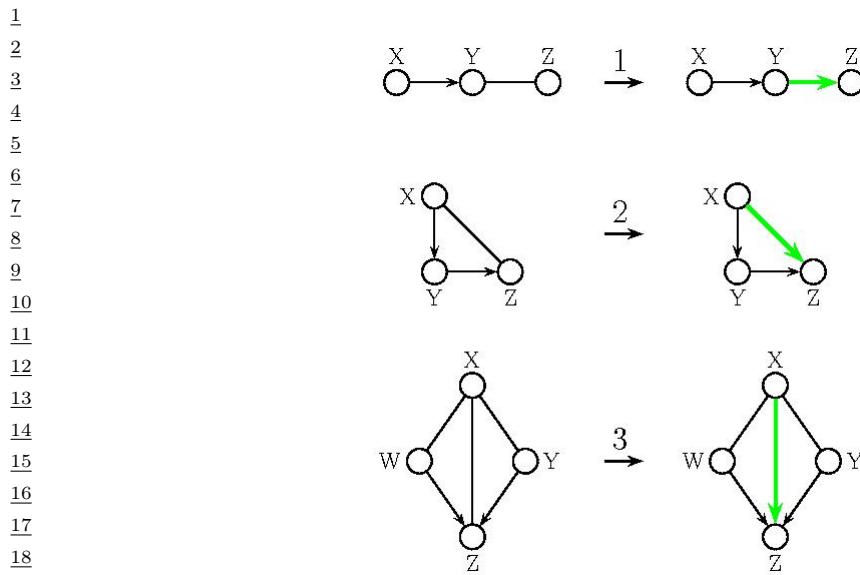
Figure 30.13: A locally optimal DAG learned from the 20-newsgroup data. From Figure 4.10 of [SchmidtThesis]. Used with kind permission of Mark Schmidt.

$1|\mathcal{D}$). We can do this exactly using dynamic programming [Koivisto06; Parviainen11ancestor], although this can be expensive. An approximate method is to sample DAGs from the posterior, and then to compute the fraction of times there is an $s \rightarrow t$ edge or path for each (s, t) pair. The standard way to draw samples is to use the Metropolis Hastings algorithm (Main Section 12.2), where we use the same local proposal as we did in greedy search [Madigan94].

A faster-mixing method is to use a collapsed MH sampler, as suggested in [Friedman03nir]. This exploits the fact that, if a total ordering of the nodes is known, we can select the parents for each node independently, without worrying about cycles, as discussed in Section 30.3.4.1. By summing over all possible choice of parents, we can marginalize out this part of the problem, and just sample total orders. [Ellis08] also use order-space (collapsed) MCMC, but this time with a parallel tempering MCMC algorithm.

30.3.5 Constraint-based approach

We now present an approach to learning a DAG structure — up to Markov equivalence (the output of the method is a PDAG) — that uses local conditional independence tests, rather than scoring models globally with a likelihood. The CI tests are combined together to infer the global graph structure, so this approach is called **constraint-based**. The advantage of CI testing is that it is more local and does not require specifying a complete model. (However, the form of the CI test



20 *Figure 30.14: The 3 rules for inferring compelled edges in PDAGs. Adapted from [Peer05].*

21
22
23 implicitly relies on assumptions, see e.g., [Shah2018].)

26 30.3.5.1 IC algorithm

27 The original algorithm, due to Verma and Pearl [Verma90], was called the **IC algorithm**, which
28 stands for “inductive causation”. The method is as follows [PearlBook]:
29

- 30 1. For each pair of variables a and b , search for a set S_{ab} such that $a \perp b | S_{ab}$. Construct an undirected
31 graph such that a and b are connected iff no such set S_{ab} can be found (i.e., they cannot be made
32 conditionally independent).
33
- 34 2. Orient the edges involved in v-structures as follows: for each pair of nonadjacent nodes a and b ,
35 with a common neighbor c , check if $c \in S_{ab}$; if it is, the corresponding DAG must be $a \rightarrow c \rightarrow b$,
36 $a \leftarrow c \rightarrow b$ or $a \leftarrow c \leftarrow b$, so we cannot determine the direction; if it is not, the DAG must be
37 $a \rightarrow c \leftarrow b$, so add these arrows to the graph.
38
- 39 3. In the partially directed graph that results, orient as many of the undirected edges as possible,
40 subject to two conditions: (1) the orientation should not create a new v-structure (since that
41 would have been detected already if it existed), and (2) the orientation should not create a directed
42 cycle. More precisely, follow the rules shown in Figure 30.14. In the first case, if $X \rightarrow Y$ has a
43 known orientation, but $Y - Z$ is unknown, then we must have $Y \rightarrow Z$, otherwise we would have
44 created a new v-structure $X \rightarrow Y \leftarrow Z$, which is not allowed. The other two cases follow similar
45 reasoning.
46

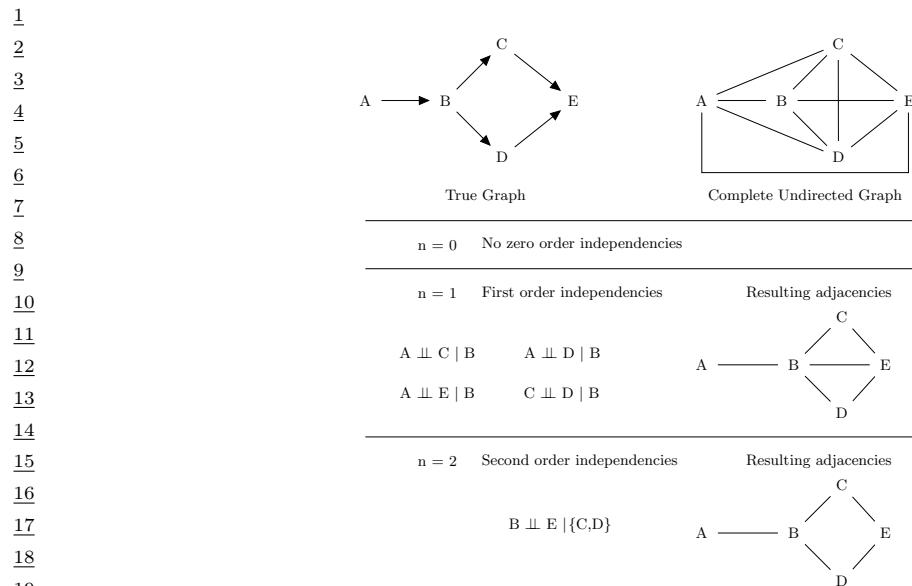


Figure 30.15: Example of step 1 of the PC algorithm. Adapted from Figure 5.1 of [SpirtesBook].

30.3.5.2 PC algorithm

A significant speedup of IC, known as the **PC algorithm** after its creators Peter Spirtes and Clark Glymour [Spirtes91], can be obtained by ordering the search for separating sets in step 1 in terms of sets of increasing cardinality. We start with a fully connected graph, and then look for sets S_{ab} of size 0, then of size 1, and so on; as soon we find a separating set, we remove the corresponding edge. See Figure 30.15 for an example.

Another variant on the PC algorithm is to learn the original undirected structure (i.e., the Markov blanket of each node) using generic variable selection techniques instead of CI tests. This tends to be more robust, since it avoids issues of statistical significance that can arise with independence tests. See [Pellet08] for details.

The running time of the PC algorithm is $O(D^{K+1})$ [SpirtesBook], where D is the number of nodes and K is the maximal degree (number of neighbors) of any node in the corresponding undirected graph.

30.3.5.3 Frequentist vs Bayesian methods

The IC/PC algorithm relies on an oracle that can test for conditional independence between any set of variables, $A \perp\!\!\!\perp B \mid C$. This can be approximated using hypothesis testing methods applied to a finite data set, such as chi-squared tests for discrete data. However, such methods work poorly with small sample sizes, and can run into problems with multiple testing (since so many hypotheses are being compared). In addition, errors made at any given step can lead to an incorrect final result, as erroneous constraints get propagated. In practice it is common to use a hybrid approach, where we use IC/PC to create an initial structure, and then use this to speed up Bayesian model selection,

1 which tends to be more robust, since it avoids any hard decisions about conditional independence or
2 lack thereof.
3

4 5 30.3.6 Methods based on sparse optimization

6 There is a 1:1 connection between sparse graphs and sparse adjacency matrices. This suggests that
7 we can perform structure learning by using continuous optimization methods that enforce sparsity,
8 similar to lasso and other ℓ_1 penalty methods (Main Section 15.2.6). In the cases of undirected graphs,
9 this is relatively straightforward, and results in a convex objective, as we discuss in Section 30.4.2.
10 However, in the case of DAGs, the problem is harder, because of the acyclicity constraint. Fortunately,
11 [Zheng2018dags] showed how to encode this constraint as a smooth penalty term. (They call their
12 method “DAGs with no tears”, since it is supposed to be painless to use.) In particular, they show
13 how to convert the combinatorial problem into a continuous problem:
14

$$\min_{\mathbf{W} \in \mathbb{R}^{D \times D}} f(\mathbf{W}) \text{ s.t. } G(\mathbf{W}) \in \text{DAGs} \iff \min_{\mathbf{W} \in \mathbb{R}^{D \times D}} f(\mathbf{W}) \text{ s.t. } h(\mathbf{W}) = 0 \quad (30.32)$$

15 Here \mathbf{W} is a weighted adjacency matrix on D nodes, $G(\mathbf{W})$ is the corresponding graph (obtained
16 by thresholding \mathbf{W} at 0), $f(\mathbf{W})$ is a scoring function (e.g., penalized log likelihood), and $h(\mathbf{W})$ is a
17 constraint function that measures how close \mathbf{W} is to defining a DAG. The constraint is given by
18

$$h(\mathbf{W}) = \text{tr}((\mathbf{I} + \alpha \mathbf{W})^d) - d \propto \text{tr}\left(\sum_{k=1}^d \alpha^k \mathbf{W}^k\right) \quad (30.33)$$

19 where $\mathbf{W}^k = \mathbf{W} \cdots \mathbf{W}$ with k terms, and $\alpha > 0$ is a regularizer. Element (i, j) of \mathbf{W}^k will be non-zero
20 iff there is a path from j to i made of K edges. Hence the diagonal elements count the number of
21 paths from an edge to itself in k steps. Thus $h(\mathbf{W})$ will be 0 if \mathbf{W} defines a valid DAG.

22 The scoring function considered in [Zheng2018dags] has the form

$$f(\mathbf{W}) = \frac{1}{2N} \|\mathbf{X} - \mathbf{X}\mathbf{W}\|_F^2 + \lambda \|\mathbf{W}\|_1 \quad (30.34)$$

23 where $\mathbf{X} \in \mathbb{R}^{ND}$ is the data matrix. The show how to find a local optimum of the equality constrained
24 objective using gradient-based methods. The cost per iteration is $O(D^3)$.

25 Several extensions of this have been proposed. For example, [Yu2019DAGGNN] replace the
26 Gaussian noise assumption with a VAE (variational autoencoder, Main Section 21.2), and use a graph
27 neural network as the encoder/decoder. And [Lachapelle2020] relax the linearity assumption, and
28 allow for the use of neural network dependencies between variables.

29 30 30.3.7 Consistent estimators

31 A natural question is whether any of the above algorithms can recover the “true” DAG structure G
32 (up to Markov equivalence), in the limit of infinite data. We assume that the data was generated by
33 a distribution p that is faithful to G (see Section 30.3.1).

34 The posterior mode (MAP) is known to converge to the MLE, which in turn will converge to the
35 true graph G (up to Markov equivalence), so any exact algorithm for Bayesian inference is a consistent
36 estimator. [Chickering02opt] showed that his greedy equivalence search method (which is a form of
37

hill climbing in the space of PDAGs) is a consistent estimator. Similarly, [SpirtesBook; Kalisch07] showed that the PC is a consistent estimator. However, the running time of these algorithms might be exponential in the number of nodes. Also, all of these methods assume that all the variables are fully observed.

30.3.8 Handling latent variables

In general, we will not get to observe the values of all the nodes (i.e., the complete data assumption does not hold), either because we have missing data, and/ or because we have hidden variables. This makes it intractable to compute the marginal likelihood of any given graph structure, as we discuss in Section 30.3.8.1. It also opens up new problems, such as knowing how many hidden variables to add to the model, and how to connect them, as we discuss in Section 30.3.8.7.

30.3.8.1 Approximating the marginal likelihood

If we have hidden or missing variables \mathbf{h} , the marginal likelihood is given by

$$p(\mathcal{D}|G) = \int \sum_{\mathbf{h}} p(\mathcal{D}, \mathbf{h}|\boldsymbol{\theta}, G) p(\boldsymbol{\theta}|G) d\boldsymbol{\theta} = \sum_{\mathbf{h}} \int p(\mathcal{D}, \mathbf{h}|\boldsymbol{\theta}, G) p(\boldsymbol{\theta}|G) d\boldsymbol{\theta} \quad (30.35)$$

In general this is intractable to compute. For example, consider a mixture model, where we don't observe the cluster label. In this case, there are K^N possible completions of the data (assuming we have K clusters); we can evaluate the inner integral for each one of these assignments to \mathbf{h} , but we cannot afford to evaluate all of the integrals. (Of course, most of these integrals will correspond to hypotheses with little posterior support, such as assigning single data points to isolated clusters, but we don't know ahead of time the relative weight of these assignments.) Below we mention some faster deterministic approximations for the marginal likelihood.

30.3.8.2 BIC approximation

A simple approximation to the marginal likelihood is to use the BIC score (Main Section 3.8.7.2), which is given by

$$\text{BIC}(G) \triangleq \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \frac{\log N}{2} \dim(G) \quad (30.36)$$

where $\dim(G)$ is the number of degrees of freedom in the model and $\hat{\boldsymbol{\theta}}$ is the MAP or ML estimate. However, the BIC score often severely underestimates the true marginal likelihood [Chickering97], resulting in it selecting overly simple models. We discuss some better approximations below.

30.3.8.3 Cheeseman-Stutz approximation

We now discuss the **Cheeseman-Stutz approximation** (CS) to the marginal likelihood [Cheeseman96].

We first compute a MAP estimate of the parameters $\hat{\boldsymbol{\theta}}$ (e.g., using EM). Denote the expected sufficient statistics of the data by $\bar{\mathcal{D}} = \bar{\mathcal{D}}(\hat{\boldsymbol{\theta}})$; in the case of discrete variables, we just “fill in” the hidden

1 variables with their expectation. We then use the exact marginal likelihood equation on this filled-in
2 data:

3

$$\text{p}(\mathcal{D}|G) \approx p(\bar{\mathcal{D}}|G) = \int p(\bar{\mathcal{D}}|\boldsymbol{\theta}, G)p(\boldsymbol{\theta}|G)d\boldsymbol{\theta} \quad (30.37)$$

4

5 However, comparing this to Equation (30.35), we can see that the value will be exponentially smaller,
6 since it does not sum over all values of \boldsymbol{h} . To correct for this, we first write

7

$$\log p(\mathcal{D}|G) = \log p(\bar{\mathcal{D}}|G) + \log p(\mathcal{D}|G) - \log p(\bar{\mathcal{D}}|G) \quad (30.38)$$

8

9 and then we apply a BIC approximation to the last two terms:

10

$$\begin{aligned} \log p(\mathcal{D}|G) - \log p(\bar{\mathcal{D}}|G) &\approx \left[\log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \frac{\log N}{2} \dim(G) \right] \\ &\quad - \left[\log p(\bar{\mathcal{D}}|\hat{\boldsymbol{\theta}}, G) - \frac{\log N}{2} \dim(G) \right] \end{aligned} \quad (30.39)$$

11

12

$$= \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \log p(\bar{\mathcal{D}}|\hat{\boldsymbol{\theta}}, G) \quad (30.40)$$

13

14 Putting it altogether we get

15

$$\log p(\mathcal{D}|G) \approx \log p(\bar{\mathcal{D}}|G) + \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \log p(\bar{\mathcal{D}}|\hat{\boldsymbol{\theta}}, G) \quad (30.41)$$

16

17 The first term $p(\bar{\mathcal{D}}|G)$ can be computed by plugging in the filled-in data into the exact marginal
18 likelihood. The second term $p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G)$, which involves an exponential sum (thus matching the
19 “dimensionality” of the left hand side) can be computed using an inference algorithm. The final term
20 $p(\bar{\mathcal{D}}|\hat{\boldsymbol{\theta}}, G)$ can be computed by plugging in the filled-in data into the regular likelihood.

21

22 30.3.8.4 Variational Bayes EM

23

24 An even more accurate approach is to use the variational Bayes EM algorithm. Recall from
25 Main Section 10.3.5 that the key idea is to make the following factorization assumption:

26

$$p(\boldsymbol{\theta}, \mathbf{z}_{1:N}|\mathcal{D}) \approx q(\boldsymbol{\theta})q(\mathbf{z}) = q(\boldsymbol{\theta}) \prod_i q(\mathbf{z}_i) \quad (30.42)$$

27

28 where \mathbf{z}_i are the hidden variables in case i . In the E step, we update the $q(\mathbf{z}_i)$, and in the M step, we
29 update $q(\boldsymbol{\theta})$. The corresponding variational free energy provides a lower bound on the log marginal
30 likelihood. In [Beal06], it is shown that this bound is a much better approximation to the true log
31 marginal likelihood (as estimated by a slow annealed importance sampling procedure) than either
32 BIC or CS. In fact, one can prove that the variational bound will always be more accurate than CS
33 (which in turn is always more accurate than BIC).

34

35 30.3.8.5 Example: college plans revisited

36

37 Let us revisit the college plans dataset from Section 30.3.3.4. Recall that if we ignore the possibility
38 of hidden variables there was a direct link from socio economic status to IQ in the MAP DAG.

39

40

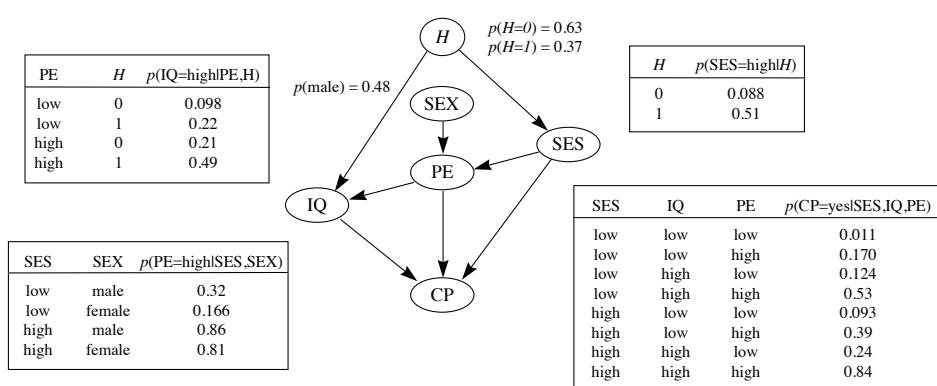


Figure 30.16: The most probable DAG with a single binary hidden variable learned from the Sewell-Shah data. MAP estimates of the CPT entries are shown for some of the nodes. From [Heckerman97]. Used with kind permission of David Heckerman.

Heckerman et al. decided to see what would happen if they introduced a hidden variable H , which they made a parent of both SES and IQ, representing a hidden common cause. They also considered a variant in which H points to SES, IQ and PE. For both such cases, they considered dropping none, one, or both of the SES-PE and PE-IQ edges. They varied the number of states for the hidden node from 2 to 6. Thus they computed the approximate posterior over $8 \times 5 = 40$ different models, using the CS approximation.

The most probable model which they found is shown in Figure 30.16. This is $2 \cdot 10^{10}$ times more likely than the best model containing no hidden variable. It is also $5 \cdot 10^9$ times more likely than the second most probable model with a hidden variable. So again the posterior is very peaked.

These results suggests that there is indeed a hidden common cause underlying both the socio-economic status of the parents and the IQ of the children. By examining the CPT entries, we see that both SES and IQ are more likely to be high when H takes on the value 1. They interpret this to mean that the hidden variable represents “parent quality” (possibly a genetic factor). Note, however, that the arc between H and SES can be reversed without changing the v-structures in the graph, and thus without affecting the likelihood; this underscores the difficulty in interpreting hidden variables.

Interestingly, the hidden variable model has the same conditional independence assumptions amongst the visible variables as the most probable visible variable model. So it is not possible to distinguish between these hypotheses by merely looking at the empirical conditional independencies in the data (which is the basis of the constraint-based approach to structure learning discussed in Section 30.3.5). Instead, by adopting a Bayesian approach, which takes parsimony into account (and not just conditional independence), we can discover the possible existence of hidden factors. This is the basis of much of scientific and everyday human reasoning (see e.g. [Griffiths09] for a discussion).

30.3.8.6 Structural EM

One way to perform structural inference in the presence of missing data is to use a standard search procedure (deterministic or stochastic), and to use the methods from Section 30.3.8.1 to estimate the

¹ marginal likelihood. However, this approach is not very efficient, because the marginal likelihood
² does not decompose when we have missing data, and nor do its approximations. For example, if
³ we use the CS approximation or the VBEM approximation, we have to perform inference in every
⁴ neighboring model, just to evaluate the quality of a single move!

⁵ [Friedman97nir; Thiesson98] presents a much more efficient approach called the **structural**
⁶ **EM** algorithm. The basic idea is this: instead of fitting each candidate neighboring graph and then
⁷ filling in its data, fill in the data once, and use this filled-in data to evaluate the score of all the
⁸ neighbors. Although this might be a bad approximation to the marginal likelihood, it can be a good
⁹ enough approximation of the difference in marginal likelihoods between different models, which is all
¹⁰ we need in order to pick the best neighbor.

¹¹ More precisely, define $\bar{\mathcal{D}}(G_0, \hat{\theta}_0)$ to be the data filled in using model G_0 with MAP parameters $\hat{\theta}_0$.
¹² Now define a modified BIC score as follows:

$$\text{BIC}(G, \mathcal{D}) \triangleq \log p(\mathcal{D}|\hat{\theta}, G) - \frac{\log N}{2} \dim(G) + \log p(G) + \log p(\hat{\theta}|G) \quad (30.43)$$

¹³ where we have included the log prior for the graph and parameters. One can show [Friedman97nir]
¹⁴ that if we pick a graph G which increases the BIC score relative to G_0 on the expected data, it will
¹⁵ also increase the score on the actual data, i.e.,

$$\text{BIC}(G, \bar{\mathcal{D}}) - \text{BIC}(G_0, \bar{\mathcal{D}}) \leq \text{BIC}(G, \mathcal{D}) - \text{BIC}(G_0, \mathcal{D}) \quad (30.44)$$

¹⁶ To convert this into an algorithm, we proceed as follows. First we initialize with some graph G_0
¹⁷ and some set of parameters θ_0 . Then we fill-in the data using the current parameters — in practice,
¹⁸ this means when we ask for the expected counts for any particular family, we perform inference using
¹⁹ our current model. (If we know which counts we will need, we can precompute all of them, which is
²⁰ much faster.) We then evaluate the BIC score of all of our neighbors using the filled-in data, and we
²¹ pick the best neighbor. We then refit the model parameters, fill-in the data again, and repeat. For
²² increased speed, we may choose to only refit the model every few steps, since small changes to the
²³ structure hopefully won't invalidate the parameter estimates and the filled-in data too much.

²⁴ One interesting application is to learn a phylogenetic tree structure. Here the observed leaves are
²⁵ the DNA or protein sequences of currently alive species, and the goal is to infer the topology of the
²⁶ tree and the values of the missing internal nodes. There are many classical algorithms for this task
²⁷ (see e.g., [Durbin98]), but one that uses structural EM is discussed in [FriedmanNirPhylo02].

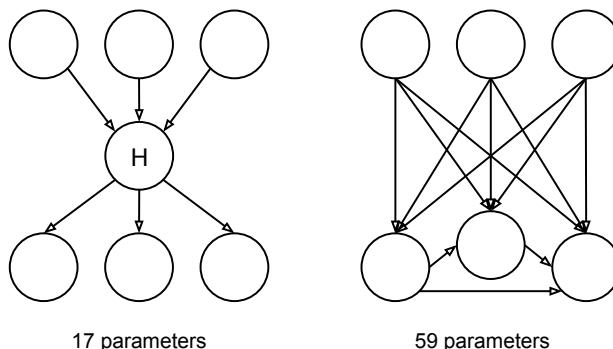
²⁸ Another interesting application of this method is to learn sparse mixture models [Barash02]. The
²⁹ idea is that we have one hidden variable C specifying the cluster, and we have to choose whether to
³⁰ add edges $C \rightarrow X_t$ for each possible feature X_t . Thus some features will be dependent on the cluster
³¹ id, and some will be independent. (See also [Law04] for a different way to perform this task, using
³² regular EM and a set of bits, one per feature, that are free to change across data cases.)

³³ 30.3.8.7 Discovering hidden variables

³⁴ In Section 30.3.8.5, we introduced a hidden variable “by hand”, and then figured out the local topology
³⁵ by fitting a series of different models and computing the one with the best marginal likelihood. How
³⁶ can we automate this process?

³⁷ Figure 30.17 provides one useful intuition: if there is a hidden variable in the “true model”, then its
³⁸ children are likely to be densely connected. This suggest the following heuristic [Elidan00]: perform
³⁹

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



14
15 Figure 30.17: A DGM with and without hidden variables. For example, the leaves might represent medical
16 symptoms, the root nodes primary causes (such as smoking, diet and exercise), and the hidden variable can
17 represent mediating factors, such as heart disease. Marginalizing out the hidden variable induces a clique.
18
19

20 structure learning in the visible domain, and then look for **structural signatures**, such as sets of
21 densely connected nodes (near-cliques); introduce a hidden variable and connect it to all nodes in
22 this near-clique; and then let structural EM sort out the details. Unfortunately, this technique does
23 not work too well, since structure learning algorithms are biased against fitting models with densely
24 connected cliques.

25 Another useful intuition comes from clustering. In a flat mixture model, also called a **latent class**
26 **model**, the discrete latent variable provides a compressed representation of its children. Thus we
27 want to create hidden variables with high mutual information with their children.

28 One way to do this is to create a tree-structured hierarchy of latent variables, each of which only
29 has to explain a small set of children. [Zhang04] calls this a **hierarchical latent class model**.
30 They propose a greedy local search algorithm to learn such structures, based on adding or deleting
31 hidden nodes, adding or deleting edges, etc. (Note that learning the optimal latent tree is NP-hard
32 [Roch06].)

33 Recently [Harmeling11] proposed a faster greedy algorithm for learning such models based on
34 agglomerative hierarchical clustering. Rather than go into details, we just give an example of what
35 this system can learn. Figure 30.18 shows part of a latent forest learned from the 20-newsgroup
36 data. The algorithm imposes the constraint that each latent node has exactly two children, for speed
37 reasons. Nevertheless, we see interpretable clusters arising. For example, Figure 30.18 shows separate
38 clusters concerning medicine, sports and religion. This provides an alternative to LDA and other
39 topic models (Main Section 28.5.1), with the added advantage that inference in latent trees is exact
40 and takes time linear in the number of nodes.

41 An alternative approach is proposed in [Choi11], in which the observed data is not constrained to
42 be at the leaves. This method starts with the Chow-Liu tree on the observed data, and then adds
43 hidden variables to capture higher-order dependencies between internal nodes. This results in much
44 more compact models, as shown in Figure 30.19. This model also has better predictive accuracy
45 than other approaches, such as mixture models, or trees where all the observed data is forced to
46 be at the leaves. Interestingly, one can show that this method can recover the exact latent tree
47

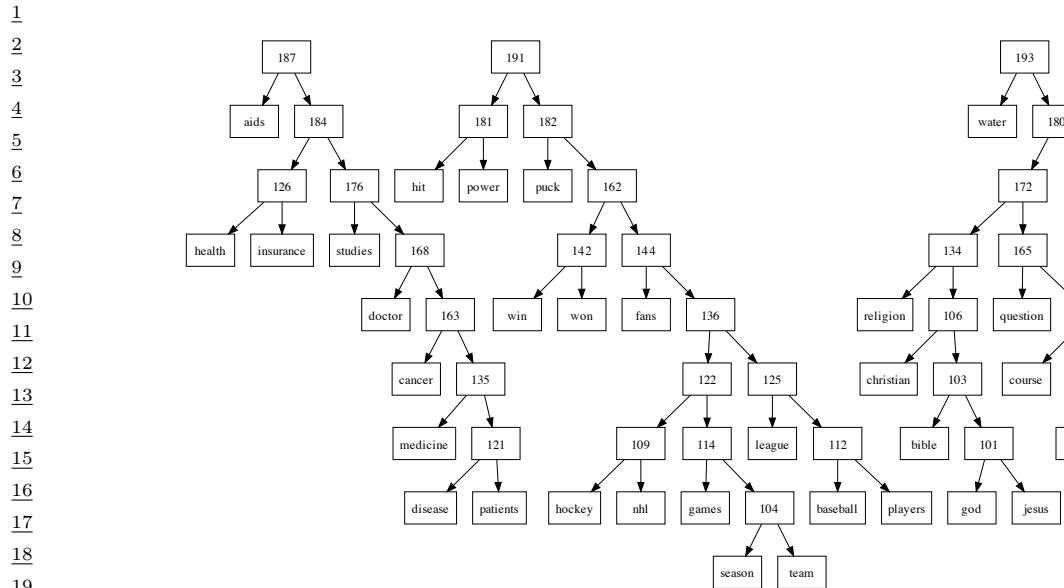


Figure 30.18: Part of a hierarchical latent tree learned from the 20-newsgroup data. From Figure 2 of [Harmeling11]. Used with kind permission of Stefan Harmeling.

structure, providing the data is generated from a tree. See [Choi11] for details. Note, however, that this approach, unlike [Zhang04; Harmeling11], requires that the cardinality of all the variables, hidden and observed, be the same. Furthermore, if the observed variables are Gaussian, the hidden variables must be Gaussian also.

30 30.3.8.8 Example: Google's Rephile

³¹ In this section, we describe a huge DGM called **Rephil**, which was automatically learned from data.²

³² The model is widely used inside Google for various purposes, including their famous AdSense system.³

The model structure is shown in Figure 30.20. The leaves are binary nodes, and represent the presence or absence of words or compounds (such as “New York City”) in a text document or query. The latent variables are also binary, and represent clusters of co-occurring words. All CPDs are noisy-OR, since some leaf nodes (representing words) can have many parents. This means each edge can be augmented with a hidden variable specifying if the link was activated or not; if the link is not active, then the parent cannot turn the child on. (A very similar model was proposed independently

⁴⁰ 2. The original system, called “Phil”, was developed by Georges Harik and Noam Shazeer,. It has been published as
⁴¹ US Patent #8024372, “Method and apparatus for learning a probabilistic generative model for text”, filed in 2004.
⁴² Rephil is a more probabilistically sound version of the method, developed by Uri Lerner et al. The summary below is
⁴³ based on notes by Brian Milch (who also works at Google).

40 3. AdSense is Google's system for matching web pages with content-appropriate ads in an automatic way, by extracting
41 semantic keywords from web pages. These keywords play a role analogous to the words that users type in when
42 searching; this latter form of information is used by Google's AdWords system. The details are secret, but [Levy11]
43 gives an overview.

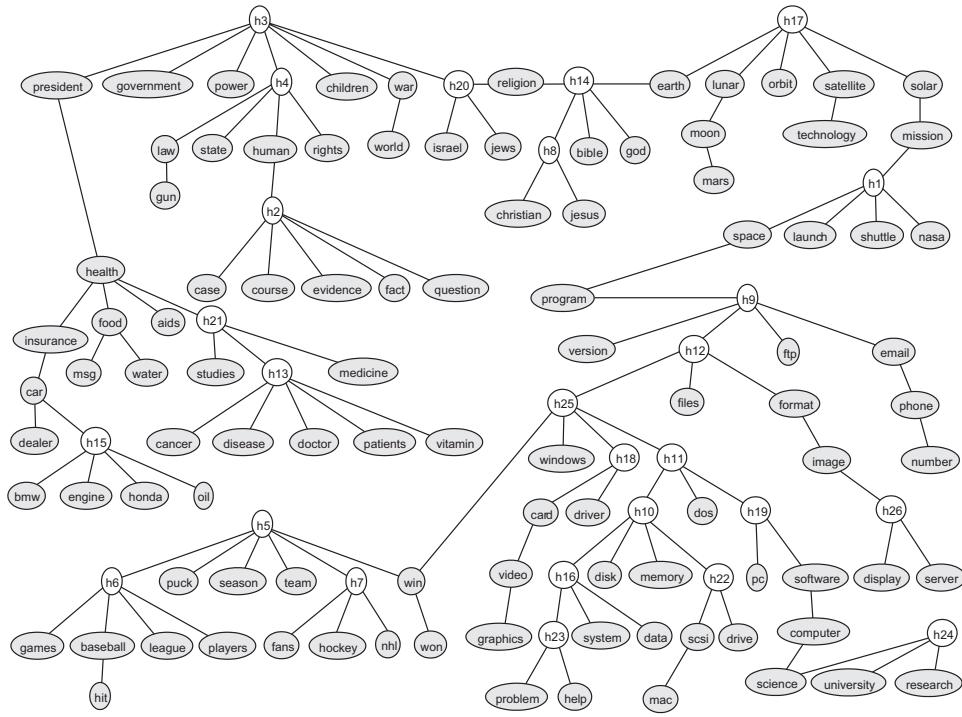


Figure 30.19: A partially latent tree learned from the 20-newsgroup data. Note that some words can have multiple meanings, and get connected to different latent variables, representing different “topics”. For example, the word “win” can refer to a sports context (represented by h_5) or the Microsoft Windows context (represented by h_{25}). From Figure 12 of [Choi11]. Used with kind permission of Jin Choi.

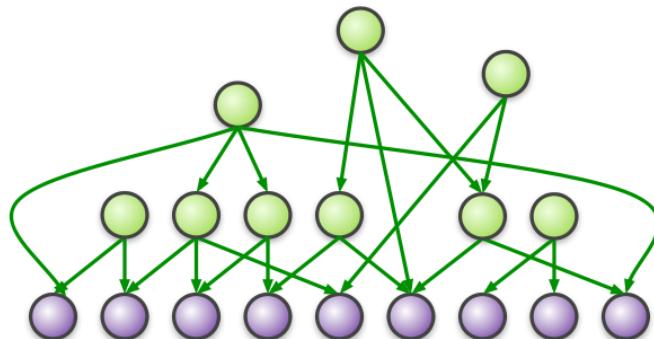


Figure 30.20: Google’s rephil model. Leaves represent presence or absence of words. Internal nodes represent clusters of co-occurring words, or “concepts”. All nodes are binary, and all CPDs are noisy-OR. The model contains 12 million word nodes, 1 million latent cluster nodes, and 350 million edges. Used with kind permission of Brian Milch.

¹ in [Singliar06].)

² Parameter learning is based on EM, where the hidden activation status of each edge needs to be
³ inferred [Meek97]. Structure learning is based on the old neuroscience idea that “**nodes that fire**
⁴ **together should wire together**”. To implement this, we run inference and check for cluster-word
⁵ and cluster-cluster pairs that frequently turn on together. We then add an edge from parent to
⁶ child if the link can significantly increase the probability of the child. Links that are not activated
⁷ very often are pruned out. We initialize with one cluster per “document” (corresponding to a set of
⁸ semantically related phrases). We then merge clusters A and B if A explains B ’s top words and vice
⁹ versa. We can also discard clusters that are used too rarely.

¹⁰ The model was trained on about 100 billion text snippets or search queries; this takes several
¹¹ weeks, even on a parallel distributed computing architecture. The resulting model contains 12 million
¹² word nodes and about 1 million latent cluster nodes. There are about 350 million links in the model,
¹³ including many cluster-cluster dependencies. The longest path in the graph has length 555, so the
¹⁴ model is quite deep.

¹⁵ Exact inference in this model is obviously infeasible. However note that most leaves will be off,
¹⁶ since most words do not occur in a given query; such leaves can be analytically removed. We can
¹⁷ also prune out unlikely hidden nodes by following the strongest links from the words that are up
¹⁸ to their parents to get a candidate set of concepts. We then perform iterative conditional modes
¹⁹ (ICM) to form approximate inference. (ICM is a deterministic version of Gibbs sampling that sets
²⁰ each node to its most probable state given the values of its neighbors in its Markov blanket.) This
²¹ continues until it reaches a local maximum. We can repeat this process a few times from random
²² starting configurations. At Google, this can be made to run in 15 milliseconds!

²³

²⁴ 30.3.8.9 Spectral methods

²⁵ Recently, various methods have been developed that can recover the exact structure of the DAG, even
²⁶ in the presence of (a known number of) latent variables, under certain assumptions. In particular,
²⁷ identifiability results have been obtained for the following cases:

- ²⁸
- ²⁹ • If \mathbf{x} contains 3 or more independent views of z [Goodman1974; Allman2009; Anandkumar12colt;
Hsu12], sometimes called the **triad constraint**.
 - ³⁰ • If z is categorical, and \mathbf{x} is a GMM with mixture components which depend on z [Anandkumar2014].
 - ³¹ • If z is composed of binary variables, and \mathbf{x} is a set of noisy-OR CPDs [Jernite13; Arora2016].

³² In terms of algorithms, most of these methods are not based on maximum likelihood, but instead use
³³ the method of moments and spectral methods. For details, see [Anandkumar2014].

³⁴ 30.3.8.10 Constraint-based methods for learning ADMGs

³⁵ An alternative to explicitly modeling latent variables is to marginalize them out, and work with acyclic
³⁶ directed mixed graphs (Main Section 4.5.4.2). It is possible to perform Bayesian model selection
³⁷ for ADMGs, although the method is somewhat slow and complicated [Silva09]. Alternatively, one
³⁸ can modify the PC/IC algorithm to learn an ADMG. This method is known as the **IC*** algorithm
³⁹ [PearlBook]; one can speed it up to get the **FCI** algorithm (FCI stands for “fast causal inference”)
⁴⁰ [SpirtesBook].

⁴¹

Since there will inevitably be some uncertainty about edge orientations, due to Markov equivalence, the output of IC*/ FCI is not actually an ADMG, but is a closely related structured called a **partially oriented inducing path graph** [SpirtesBook] or a **marked pattern** [PearlBook]. Such a graph has 4 kinds of edges:

- A marked arrow $a \xrightarrow{*} b$ signifying a directed path from a to b .
- An unmarked arrow $a \rightarrow b$ signifying a directed path from a to b or a latent common cause $a \leftarrow L \rightarrow b$.
- A bidirected arrow $a \leftrightarrow b$ signifying a latent common causes $a \leftarrow L \rightarrow b$.
- An undirected edge $a - b$ signifying $a \leftarrow b$ or $a \rightarrow b$ or a latent common causes $a \leftarrow L \rightarrow b$.

IC*/ FCI is faster than Bayesian inference, but suffers from the same problems as the original IC/PC algorithm (namely, the need for a CI testing oracle, problems due to multiple testing, no probabilistic representation of uncertainty, etc.) Furthermore, by not explicitly representing the latent variables, the resulting model cannot be used for inference and prediction.

30.4 Learning undirected graph structures

In this section, we discuss how to learn the structure of undirected graphical models. On the one hand, this is easier than learning DAG structure because we don't need to worry about acyclicity. On the other hand, it is harder than learning DAG structure since the likelihood does not decompose (see Main Section 4.3.9.1). This precludes the kind of local search methods (both greedy search and MCMC sampling) we used to learn DAG structures, because the cost of evaluating each neighboring graph is too high, since we have to refit each model from scratch (there is no way to incrementally update the score of a model). In this section, we discuss several solutions to this problem.

30.4.1 Dependency networks

A simple way to learn the structure of a UGM is to represent it as a product of full conditionals:

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{d=1}^D p(x_d | \mathbf{x}_{-d}) \quad (30.45)$$

This expression is called the **pseudolikelihood**.

Such a collection of local distributions defines a model called a **dependency network** [Heckerman00]. Unfortunately, a product of full conditionals which are independently estimated is not guaranteed to be consistent with any valid joint distribution. However, we can still use the model inside of a Gibbs sampler to approximate a joint distribution. This approach is sometimes used for data imputation [Gelman01].

However, the main use advantage of dependency networks is that we can use sparse regression techniques for each distribution $p(x_d | \mathbf{x}_{-d})$ to induce a sparse graph structure. For example, [Heckerman00] use classification/ regression trees, [Meinshausen06] use ℓ_1 -regularized linear regression, [Wainwright06; Wu2019nips] use ℓ_1 -regularized logistic regression, [Dobra09] uses Bayesian variable selection, etc.

Figure 30.21 shows a dependency network that was learned from the 20-newsgroup data using ℓ_1 regularized logistic regression, where the penalty parameter λ was chosen by BIC. Many of the words present in these estimated Markov blankets represent fairly natural associations (aids:disease,

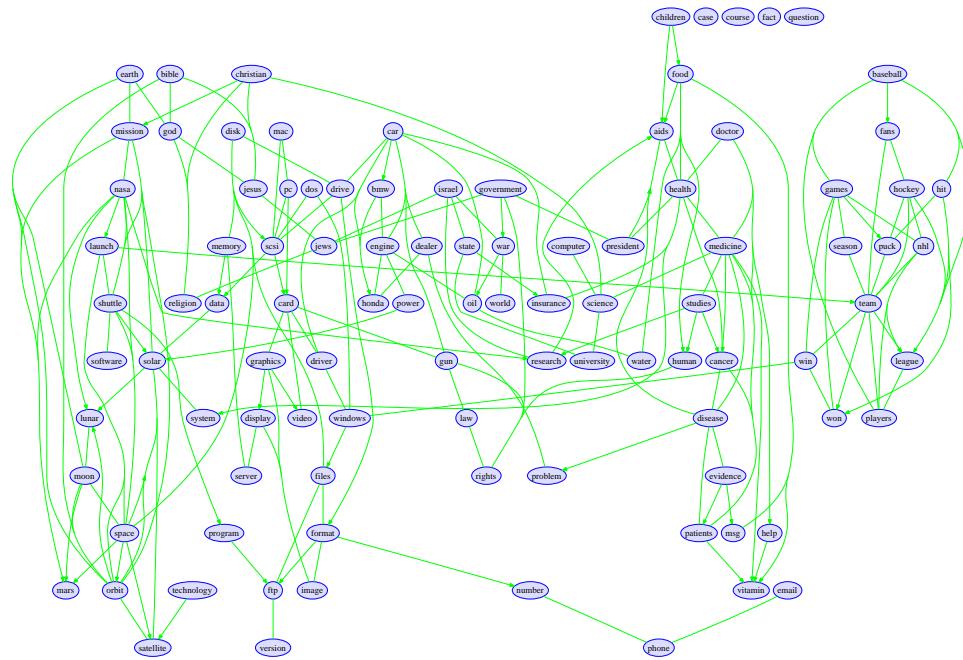


Figure 30.21: A dependency network constructed from the 20 newsgroup data. We show all edges with regression weight above 0.5 in the Markov blankets estimated by ℓ_1 penalized logistic regression. Undirected edges represent cases where a directed edge was found in both directions. From Figure 4.9 of [SchmidtThesis]. Used with kind permission of Mark Schmidt.

baseball:fans, bible:god, bmw:car, cancer:patients, etc.). However, some of the estimated statistical dependencies seem less intuitive, such as baseball:windows and bmw:christian. We can gain more insight if we look not only at the sparsity pattern, but also the values of the regression weights. For example, here are the incoming weights for the first 5 words:

- **aids**: children (0.53), disease (0.84), fact (0.47), health (0.77), president (0.50), research (0.53)
- **baseball**: *christian* (-0.98), *drive* (-0.49), games (0.81), *god* (-0.46), *government* (-0.69), hit (0.62), *memory* (-1.29), players (1.16), season (0.31), *software* (-0.68), *windows* (-1.45)
- **bible**: *car* (-0.72), *card* (-0.88), christian (0.49), fact (0.21), god (1.01), jesus (0.68), orbit (0.83), *program* (-0.56), religion (0.24), version (0.49)
- **bmw**: car (0.60), *christian* (-11.54), engine (0.69), *god* (-0.74), *government* (-1.01), *help* (-0.50), *windows* (-1.43)
- **cancer**: disease (0.62), medicine (0.58), patients (0.90), research (0.49), studies (0.70)

Words in italic red have negative weights, which represents a dissociative relationship. For example, the model reflects that baseball:windows is an unlikely combination. It turns out that most of the weights are negative (1173 negative, 286 positive, 8541 zero) in this model.

[Meinshausen06] discuss theoretical conditions under which dependency networks using ℓ_1 -regularized linear regression can recover the true graph structure, assuming the data was generated from a sparse Gaussian graphical model. We discuss a more general solution in Section 30.4.2.

30.4.2 Graphical lasso for GGMs

In this section, we consider the problem of learning the structure of undirected Gaussian graphical models (GGM)s. These models are useful, since there is a 1:1 mapping between sparse parameters and sparse graph structures. This allows us to extend the efficient techniques of ℓ_1 regularized estimation in Main Section 15.2.6 to the graph case; the resulting method is called the **graphical lasso** or **Glasso** [Friedman08glasso; Mazumder12].

30.4.2.1 MLE for a GGM

Before discussing structure learning, we need to discuss parameter estimation. The task of computing the MLE for a (non-decomposable) GGM is called **covariance selection** [Dempster72].

The log likelihood can be written as

$$\ell(\boldsymbol{\Omega}) = \log \det \boldsymbol{\Omega} - \text{tr}(\mathbf{S}\boldsymbol{\Omega}) \quad (30.46)$$

where $\boldsymbol{\Omega} = \boldsymbol{\Sigma}^{-1}$ is the precision matrix, and $\mathbf{S} = \frac{1}{N} \sum_{i=1}^N (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T$ is the empirical covariance matrix. (For notational simplicity, we assume we have already estimated $\hat{\mu} = \bar{\mathbf{x}}$.) One can show that the gradient of this is given by

$$\nabla \ell(\boldsymbol{\Omega}) = \boldsymbol{\Omega}^{-1} - \mathbf{S} \quad (30.47)$$

However, we have to enforce the constraints that $\Omega_{st} = 0$ if $G_{st} = 0$ (structural zeros), and that $\boldsymbol{\Omega}$ is positive definite. The former constraint is easy to enforce, but the latter is somewhat challenging (albeit still a convex constraint). One approach is to add a penalty term to the objective if $\boldsymbol{\Omega}$ leaves the positive definite cone; this is the approach used in [Dahl08]. Another approach is to use a coordinate descent method, described in [HastieBook].

Interestingly, one can show that the MLE must satisfy the following property: $\Sigma_{st} = S_{st}$ if $G_{st} = 1$ or $s = t$, i.e., the covariance of a pair that are connected by an edge must match the empirical covariance. In addition, we have $\Omega_{st} = 0$ if $G_{st} = 0$, by definition of a GGM, i.e., the precision of a pair that are not connected must be 0. We say that $\boldsymbol{\Sigma}$ is a positive definite **matrix completion** of \mathbf{S} , since it retains as many of the entries in \mathbf{S} as possible, corresponding to the edges in the graph, subject to the required sparsity pattern on $\boldsymbol{\Sigma}^{-1}$, corresponding to the absent edges; the remaining entries in $\boldsymbol{\Sigma}$ are filled in so as to maximize the likelihood.

Let us consider a worked example from [HastieBook]. We will use the following adjacency matrix, representing the cyclic structure, $X_1 - X_2 - X_3 - X_4 - X_1$, and the following empirical covariance matrix:

$$\mathbf{G} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}, \quad \mathbf{S} = \begin{pmatrix} 10 & 1 & 5 & 4 \\ 1 & 10 & 2 & 6 \\ 5 & 2 & 10 & 3 \\ 4 & 6 & 3 & 10 \end{pmatrix} \quad (30.48)$$

1 The MLE is given by
2

$$\Sigma = \begin{pmatrix} 10.00 & 1.00 & \mathbf{1.31} & 4.00 \\ 1.00 & 10.00 & 2.00 & \mathbf{0.87} \\ \mathbf{1.31} & 2.00 & 10.00 & 3.00 \\ 4.00 & \mathbf{0.87} & 3.00 & 10.00 \end{pmatrix}, \quad \Omega = \begin{pmatrix} 0.12 & -0.01 & \mathbf{0} & -0.05 \\ -0.01 & 0.11 & -0.02 & \mathbf{0} \\ \mathbf{0} & -0.02 & 0.11 & -0.03 \\ -0.05 & \mathbf{0} & -0.03 & 0.13 \end{pmatrix} \quad (30.49)$$

8
9 (See [gmm_fit_demo.ipynb](#) for the code to reproduce these numbers, using the coordinate descent
10 algorithm from [\[Friedman08glasso\]](#).) The constrained elements in Ω , and the free elements in Σ ,
11 both of which correspond to absent edges, have been highlighted.

12

13 30.4.2.2 Promoting sparsity

14
15 We now discuss one way to learn a sparse Gaussian MRF structure, which exploits the fact that
16 there is a 1:1 correspondence between zeros in the precision matrix and absent edges in the graph.
17 This suggests that we can learn a sparse graph structure by using an objective that encourages zeros
18 in the precision matrix. By analogy to lasso (see Main Section 15.2.6), one can define the following
19 ℓ_1 penalized NLL:

$$\begin{aligned} \text{J}(\Omega) &= -\log \det \Omega + \text{tr}(\mathbf{S}\Omega) + \lambda \|\Omega\|_1 \end{aligned} \quad (30.50)$$

20
21 where $\|\Omega\|_1 = \sum_{j,k} |\omega_{jk}|$ is the 1-norm of the matrix. This is called the **graphical lasso** or **Glasso**.
22
23 Although the objective is convex, it is non-smooth (because of the non-differentiable ℓ_1 penalty)
24 and is constrained (because Ω must be a positive definite matrix). Several algorithms have been
25 proposed for optimizing this objective [\[Yuan07gmm; Banerjee08; Duchi08\]](#), although arguably
26 the simplest is the one in [\[Friedman08glasso\]](#), which uses a coordinate descent algorithm similar
27 to the shooting algorithm for lasso. An even faster method, based on soft thresholding, is described
28 in [\[Fattahii2018; Fattahii2018ieee\]](#).

29
30 As an example, let us apply the method to the flow cytometry dataset from [\[Sachs05\]](#). A discretized
31 version of the data is shown in Main Figure 20.7(a). Here we use the original continuous data.
32 However, we are ignoring the fact that the data was sampled under intervention. In Main Figure 30.1,
33 we illustrate the graph structures that are learned as we sweep λ from 0 to a large value. These
34 represent a range of plausible hypotheses about the connectivity of these proteins.

35
36 It is worth comparing this with the DAG that was learned in Main Figure 20.7(b). The DAG has
37 the advantage that it can easily model the interventional nature of the data, but the disadvantage
38 that it cannot model the feedback loops that are known to exist in this biological pathway (see the
39 discussion in [\[Schmidt09dcg\]](#)). Note that the fact that we show many UGMs and only one DAG is
40 incidental: we could easily use BIC to pick the “best” UGM, and conversely, we could easily display
several DAG structures, sampled from the posterior.

41

42 30.4.3 Graphical lasso for discrete MRFs/CRFs

43
44 It is possible to extend the graphical lasso idea to the discrete MRF and CRF case. However, now
45 there is a set of parameters associated with each edge in the graph, so we have to use the graph
46 analog of group lasso (see Main Section 15.2.6). For example, consider a pairwise CRF with ternary
47

1 nodes, and node and edge potentials given by
2

$$\begin{aligned} \psi_t(y_t, \mathbf{x}) &= \begin{pmatrix} \mathbf{v}_{t1}^\top \mathbf{x} \\ \mathbf{v}_{t2}^\top \mathbf{x} \\ \mathbf{v}_{t3}^\top \mathbf{x} \end{pmatrix}, \quad \psi_{st}(y_s, y_t, \mathbf{x}) = \begin{pmatrix} \mathbf{w}_{t11}^\top \mathbf{x} & \mathbf{w}_{st12}^\top \mathbf{x} & \mathbf{w}_{st13}^\top \mathbf{x} \\ \mathbf{w}_{st21}^\top \mathbf{x} & \mathbf{w}_{st22}^\top \mathbf{x} & \mathbf{w}_{st23}^\top \mathbf{x} \\ \mathbf{w}_{st31}^\top \mathbf{x} & \mathbf{w}_{st32}^\top \mathbf{x} & \mathbf{w}_{st33}^\top \mathbf{x} \end{pmatrix} \end{aligned} \quad (30.51)$$

7 where we assume \mathbf{x} begins with a constant 1 term, to account for the offset. (If \mathbf{x} only contains 1,
8 the CRF reduces to an MRF.) Note that we may choose to set some of the \mathbf{v}_{tk} and \mathbf{w}_{stjk} weights to
9 0, to ensure identifiability, although this can also be taken care of by the prior.

10 To learn sparse structure, we can minimize the following objective:

$$\begin{aligned} J &= -\sum_{i=1}^N \left[\sum_t \log \psi_t(y_{it}, \mathbf{x}_i, \mathbf{v}_t) + \sum_{s=1}^{N_G} \sum_{t=s+1}^{N_G} \log \psi_{st}(y_{is}, y_{it}, \mathbf{x}_i, \mathbf{w}_{st}) \right] \\ &\quad + \lambda_1 \sum_{s=1}^{N_G} \sum_{t=s+1}^{N_G} \|\mathbf{w}_{st}\|_p + \lambda_2 \sum_{t=1}^{N_G} \|\mathbf{v}_t\|_2 \end{aligned} \quad (30.52)$$

18 where $\|\mathbf{w}_{st}\|_p$ is the p -norm; common choices are $p = 2$ or $p = \infty$, as explained in Main Section
19 15.2.6. This method of CRF structure learning was first suggested in [Schmidt08]. (The use of
20 ℓ_1 regularization for learning the structure of binary MRFs was proposed in [Lee06].)

21 Although this objective is convex, it can be costly to evaluate, since we need to perform inference
22 to compute its gradient, as explained in Main Section 4.4.3 (this is true also for MRFs), due to the
23 global partition function. We should therefore use an optimizer that does not make too many calls to
24 the objective function or its gradient, such as the projected quasi-Newton method in [Schmidt09].
25 In addition, we can use approximate inference, such as loopy belief propagation (Main Section 9.4),
26 to compute an approximate objective and gradient more quickly, although this is not necessarily
27 theoretically sound.

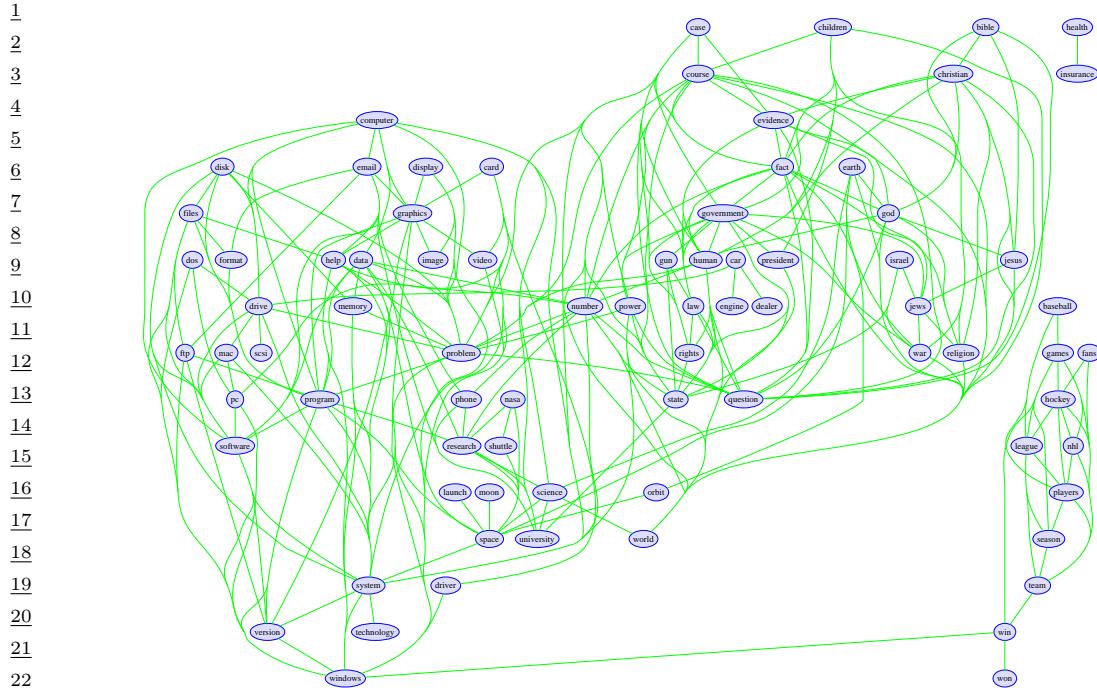
28 Another approach is to apply the group lasso penalty to the pseudo-likelihood discussed in
29 Main Section 4.3.9.3. This is much faster, since inference is no longer required [Hoeffling09].
30 Figure 30.22 shows the result of applying this procedure to the 20-newsgroup data, where y_{it} indicates
31 the presence of word t in document i , and $\mathbf{x}_i = 1$ (so the model is an MRF).

32 For a more recent approach to learning sparse discrete UPGM structures, based on sparse full
33 conditionals, see the GRISE (Generalized Regularized Interaction Screening Estimator) method of
34 [Vuffray2019], which takes polynomial time, yet its sample complexity is close to the information-
35 theoretic lower bounds [Lokhov2018].

37 30.4.4 Bayesian inference for undirected graph structures

39 Although the graphical lasso is reasonably fast, it only gives a point estimate of the structure.
40 Furthermore, it is not model-selection consistent [Meinshausen05], meaning it cannot recover the
41 true graph even as $N \rightarrow \infty$. It would be preferable to integrate out the parameters, and perform
42 posterior inference in the space of graphs, i.e., to compute $p(G|\mathcal{D})$. We can then extract summaries
43 of the posterior, such as posterior edge marginals, $p(G_{ij} = 1|\mathcal{D})$, just as we did for DAGs. In this
44 section, we discuss how to do this.

45 If the graph is decomposable, and if we use conjugate priors, we can compute the marginal likelihood
46 in closed form [Dawid93]. Furthermore, we can efficiently identify the decomposable neighbors of



23 *Figure 30.22: An MRF estimated from the 20-newsgroup data using group ℓ_1 regularization with $\lambda = 256$.
24 Isolated nodes are not plotted. From Figure 5.9 of [SchmidtThesis]. Used with kind permission of Mark
25 Schmidt.*

26
27
28
29 a graph [Thomas09], i.e., the set of legal edge additions and removals. This means that we can
30 perform relatively efficient stochastic local search to approximate the posterior (see e.g. [Giudici99;
31 Armstrong08; Scott08]).

32 However, the restriction to decomposable graphs is rather limiting if one's goal is knowledge
33 discovery, since the number of decomposable graphs is much less than the number of general
34 undirected graphs.⁴

35 A few authors have looked at Bayesian inference for GGM structure in the non-decomposable case
36 (e.g., [Dellaportas03; Wong03; Jones05]), but such methods cannot scale to large models because
37 they use an expensive Monte Carlo approximation to the marginal likelihood [Atay-Kayis05].
38 [Lenkoski08] suggested using a Laplace approximation. This requires computing the MAP estimate
39 of the parameters for Ω under a G-Wishart prior [Roverato02]. In [Lenkoski08], they used the
40 iterative proportional scaling algorithm [Speed86; Hara08] to find the mode. However, this is very
41 slow, since it requires knowing the maximal cliques of the graph, which is NP-hard in general.

42 In [Moghaddam09], a much faster method is proposed. In particular, they modify the gradient-

43
44 4. The number of decomposable graphs on N_G nodes, for $N_G = 2, \dots, 8$, is as follows ([Armstrong05]): 2; 8; 61; 822;
45 18,154; 61,7675; 30,888,596. If we divide these numbers by the number of undirected graphs, which is $2^{N_G(N_G-1)/2}$,
46 we find the ratios are: 1, 1, 0.95, 0.8, 0.55, 0.29, 0.12. So we see that decomposable graphs form a vanishing fraction of
the total hypothesis space.

based methods from Section 30.4.2.1 to find the MAP estimate; these algorithms do not need to know the cliques of the graph. A further speedup is obtained by just using a diagonal Laplace approximation, which is more accurate than BIC, but has essentially the same cost. This, plus the lack of restriction to decomposable graphs, enables fairly fast stochastic search methods to be used to approximate $p(G|\mathcal{D})$ and its mode. This approach significantly outperformed graphical lasso, both in terms of predictive accuracy and structural recovery, for a comparable computational cost.

30.5 Learning causal DAGs

Causal reasoning (which we discuss in more detail in Main Chapter 36) relies on knowing the underlying structure of the DAG (although [Jaber2019] shows how to answer some queries if we just know the graph up to Markov equivalence). Learning this structure is called **causal discovery** (see e.g., [Glymour2019]).

If we just have two variables, we need to know if the causal model should be written as $X \rightarrow Y$ or $X \leftarrow Y$. Both of these models are Markov equivalent (Section 30.3.2), meaning they cannot be distinguished from observational data, yet they make very different causal predictions. We discuss how to learn cause-effect pairs in Section 30.5.1.

When we have more than 2 variables, we need to consider more general techniques. In Section 30.3, we discuss how to learn a DAG structure from observational data using likelihood based methods, and hypothesis testing methods. However, these approaches cannot distinguish between models that are Markov equivalent, so we need to perform interventions to reduce the size of the equivalence class [Solus2019]. We discuss some suitable methods in Section 30.5.2.

The above techniques assume that the causal variables of interest (e.g., cancer rates, smoking rates) can be measured directly. However, in many ML problems, the data is much more “low level”. For example, consider trying to learn a causal model of the world from raw pixels. We briefly discuss this topic in Section 30.5.3.

For more details on causal discovery methods, see e.g., [Eberhardt2017; Peters2017; HeinzeDeml2018; Guo2021].

30.5.1 Learning cause-effect pairs

If we only observe a pair of variables, we cannot use methods discussed in Section 30.3 to learn graph structure, since such methods are based on conditional independence tests, which need at least 3 variables. However, intuitively, we should still be able to learn causal relationships in this case. For example, we know that altitude X causes temperature Y and not vice versa. For example, suppose we measure X and Y in two different countries, say the Netherlands (low altitude) and Switzerland (high altitude). If we represent the joint distribution as $p(X, Y) = p(X)p(Y|X)$, we find that the $p(Y|X)$ distribution is stable across the two populations, while $p(X)$ will change. However, if we represent the joint distribution as $p(X, Y) = p(Y)p(X|Y)$, we find that both $p(Y)$ and $p(X|Y)$ need to change across populations, so both of the corresponding distributions will be more “complicated” to capture this non-stationarity in the data. In this section, we discuss some approaches that exploit this idea. Our presentation is based on [Peters2017]. (See [Mooij2016] for more details.)

1 2 **30.5.1.1 Algorithmic information theory**

3 Suppose $X \in \{0, 1\}$ and $Y \in \mathbb{R}$ and we represent the joint $p(x, y)$ using

4
$$p(x, y) = p(x)p(y|x) = \text{Ber}(x|\theta)\mathcal{N}(y|\mu_x, 1) \quad (30.53)$$

5 We can equally well write this in the following form [Dawid02]:

6
$$p(x, y) = p(y)p(x|y) = [\theta\mathcal{N}(y|\mu_1, 1) + (1 - \theta)\mathcal{N}(y|\mu_2, 1)]\text{Ber}(x|\sigma(\alpha + \beta y)) \quad (30.54)$$

7 where $\alpha = \text{logit}(\theta) + \mu_2^2 - \mu_1^2$ and $\beta = \mu_1 - \mu_2$. We can plausibly argue that the first model, which
8 corresponds to $X \rightarrow Y$, is more likely to be correct, since it consists of two simple distributions
9 that seem to be rather generic. By contrast, in Equation (30.54), the distribution of $p(Y)$ is more
10 complex, and seems to be dependent on the specific form of $p(X|Y)$.

11 [Janzing10] show how to formalize this intuition using **algorithmic information theory**. In
12 particular, they say that X causes Y if the *distributions* P_X and $P_{Y|X}$ (not the random variables X
13 and Y) are **algorithmically independent**. To define this, let $P_X(X)$ be the distribution induced
14 by $f_X(X, U_X)$, where U_X is a bit string, and f_X is represented by a **Turing machine**. Define $P_{Y|X}$
15 analogously. Finally, let $K(s)$ be the **Kolmogorov complexity** of bit string s , i.e., the length of
16 the shortest program that would generate s using a universal Turing machine. We say that P_X and
17 $P_{Y|X}$ are algorithmically independent if

18
$$K(P_{X,Y}) = K(P_X) + K(P_{Y|X}) \quad (30.55)$$

19 Unfortunately, there is no algorithm to compute the Kolmogorov complexity, so this approach is
20 purely conceptual. In the sections below, we discuss some more practical metrics.

21 22 **30.5.1.2 Additive noise models**

23 A generic two-variable SCM of the form $X \rightarrow Y$ requires specifying the function $X = f_X(U_X)$, the
24 distribution of U_X , the function $Y = f_Y(X, U_Y)$, and the distribution of U_Y . We can simplify our
25 notation by letting $X = U_x$ and defining $p(X)$ directly, and defining $Y = f_Y(X, U_Y) = f(X, U)$,
26 where U is a noise term.

27 In general, such a model is not identifiable from a finite dataset. For example, we can imagine that
28 the value of U can be used to select between different functional mappings, $Y = f(X, U = u) = f_u(X)$.
29 Since U is not observed, the induced distribution will be a mixture of different mappings, and it
30 will generally be impossible to disentangle. For example, consider the case where X and U are
31 Bernoulli random variables, and U selects between the functions $Y = f_{\text{id}}(X) = \mathbb{I}(Y = X)$ and
32 $Y = f_{\text{neg}}(X) = \mathbb{I}(Y \neq X)$. In this case, the induced distribution $p(Y)$ is uniform, independent of X ,
33 even though we have the structure $X \rightarrow Y$.

34 The above concerns motivate the desire to restrict the flexibility of the functions at each node. One
35 natural family is **additive noise models (ANM)**, where we assume each variable has the following
36 dependence on its parents [Hoyer2009]:

37
$$X_i = f_i(X_{\text{pa}_i}) + U_i \quad (30.56)$$

38 In the case of two variables, we have $Y = f(X) + U$. If X and U are both Gaussian, and f is linear,
39 the system defines a jointly Gaussian distribution $p(X, Y)$, as we discussed in Main Section 2.3.2.2.

40

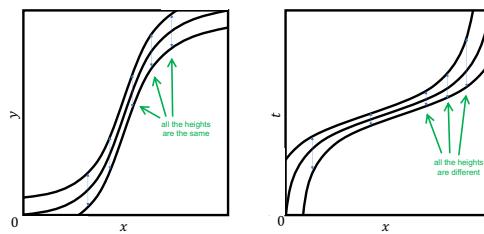


Figure 30.23: Signature of X causing Y . Left: If we try to predict Y from X , the residual error (noise term, shown by vertical arrows) is independent of X . Right: If we try to predict X from Y , the residual error is not constant. From Figure 8.8 of [Varshney2021]. Used with kind permission of Kush Varshney.

This is symmetric, and prevents us distinguishing $X \rightarrow Y$ from $Y \rightarrow X$. However, if we let f be nonlinear, and/or let X or U be non-Gaussian, we can distinguish $X \rightarrow Y$ from $Y \rightarrow X$, as we discuss below.

30.5.1.3 Nonlinear additive noise models

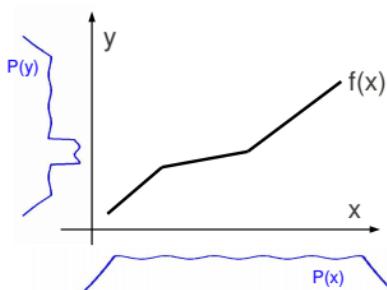
Suppose $p_{Y|X}$ is an additive noise model (possibly Gaussian noise) where f is a nonlinear function. In this case, we will not, in general, be able to create an ANM for $p_{X|Y}$. Thus we can determine whether $X \rightarrow Y$ or vice versa as follows: we fit a (nonlinear) regression model for $X \rightarrow Y$, and then check if the residual error $Y - \hat{f}_Y(X)$ is independent of X ; we then repeat the procedure swapping the roles of X and Y . The theory [Peters2017] says that the independence test will only pass for the causal direction. See Figure 30.23 for an illustration.

30.5.1.4 Linear models with non-Gaussian noise

If the function mapping from X to Y is linear, we cannot tell if $X \rightarrow Y$ or $Y \rightarrow X$ if we assume Gaussian noise. This is apparent from the symmetry of Figure 30.23 in the linear case. However, by combining linear models with non-Gaussian noise, we can recover identifiability.

For example, consider the ICA model from Main Section 28.6. This is a simple linear model of the form $\mathbf{y} = \mathbf{Ax}$, where $p(\mathbf{x})$ has a non-Gaussian distribution, and $p(\mathbf{y}|\mathbf{x})$ is a degenerate distribution, since we assume the observation model is deterministic (noise-free). In the ICA case, we can uniquely identify the parameters \mathbf{A} and the corresponding latent source \mathbf{x} . This lets us distinguish the $X \rightarrow Y$ model from the $Y \rightarrow X$ model. (The intuition behind this method is that linear combinations of random variables tend towards a Gaussian distribution (by the central limit theorem), so if $X \rightarrow Y$, then $p(Y)$ will “look more Gaussian” than $p(X)$.)

Another setting in which we can distinguish the direction of the arrow is when we have non-Gaussian observation noise, i.e., $\mathbf{y} = \mathbf{Ax} + U_Y$, where U_Y is non-Gaussian. This is an example of a “linear non-Gaussian acyclic model” (LiNGAM) [Shimizu2006]. The non-Gaussian additive noise results in the induced distributions $p(X, Y)$ being different depending on whether $X \rightarrow Y$ or $Y \rightarrow X$.



13 *Figure 30.24: Illustration of information-geometric causal inference for $Y = f(X)$. The density of the effect*
14 *$p(Y)$ tends to be high in regions where f is flat (and hence f^{-1} is steep). From Figure 4 of [Janzing2012].*

17 30.5.1.5 Information-geometric causal inference

19 An alternative approach, known as **information-geometric causal inference**, or **IGCI**, was
20 proposed in [Danisius2010; Janzing2012]. In this method, we assume f is a deterministic strictly
21 monotonic function on $[0, 1]$, with $f(0) = 0$ and $f(1) = 1$, and there is no observation noise, so
22 $Y = f(X)$. If X has the distribution $p(X)$, then the shape of the induced distribution $p(Y)$ will
23 depend on the form of the function f , as illustrated in Figure 30.24. Intuitively, the peaks of $p(Y)$
24 will occur in regions where f has small slope, and thus f^{-1} has large slope. Thus $p_Y(Y)$ and $f^{-1}(Y)$
25 will depend on each other, whereas $p_X(X)$ and $f(X)$ do not (since we assume the distribution of
26 causes is independent of the causal mechanism).

27 More precisely, let the functions $\log f'$ (the log of the derivative function) and p_X be viewed
28 as random variables on the probability space $[0, 1]$ with a uniform distribution. We say $p_{X,Y}$
29 satisfies an IGCI model if f is a mapping as above, and the following independence criterion holds:
30 $\text{Cov} [\log f', p_X] = 0$, where

$$\text{Cov} [\log f', p_X] = \int_0^1 \log f'(x)p_X(x)dx - \int_0^1 \log f'(x)dx \int_0^1 p_X(x)dx \quad (30.57)$$

34 where $\int_0^1 p_X(x)dx = 1$. One can show that the inverse function f^{-1} satisfies $\text{Cov} [\log f'^{-1}, p_Y] \geq 0$,
35 with equality iff f is linear.

37 This can be turned into an empirical test as follows. Define

$$C_{X \rightarrow Y} = \int_0^1 \log f'(x)p(x)dx \approx \frac{1}{N-1} \sum_{j=1}^{N-1} \log \frac{|y_{j+1} - y_j|}{|x_{j+1} - x_j|} \quad (30.58)$$

42 where $x_1 < x_2 < \dots < x_N$ are the observed x -values in increasing order. The quantity $C_{Y \rightarrow X}$ is defined
43 analogously. We then choose $X \rightarrow Y$ as the model whenever $\hat{C}_{X \rightarrow Y} < \hat{C}_{Y \rightarrow X}$. This is called the
44 **slope based approach** to IGCI.

45 One can also show that an IGCI model satisfies the property that $H(X) \leq H(Y)$, where $H()$
46 is the differential entropy. Intuitively, the reason is that applying a nonlinear function f to p_X
47

can introduce additional irregularities, thus making p_Y less uniform than p_X . This is illustrated in Figure 30.24. We can then choose between $X \rightarrow Y$ and $X \leftarrow Y$ based on the difference in estimated entropies.

An empirical comparison of the slope-based and entropy-based approaches to IGCI can be found in [Mooij2016].

30.5.2 Learning causal DAGs from interventional data

In Section 30.3, we discuss how to learn a DAG structure from observational data, using either likelihood-based (Bayesian) methods of model selection, or constraint-based (frequentist) methods. (See [Tu2019nips] for a recent empirical comparison of such methods applied to a medical simulator.) However, such approaches cannot distinguish between models that are Markov equivalent, and thus the output may not be sufficient to answer all causal queries of interest.

To distinguish DAGs within the same Markov equivalence class, we can use **interventional data**, where certain variables have been set, and the consequences have been measured. In particular, we can modify the standard likelihood-based DAG learning method discussed in Section 30.3 to take into account the fact that the data generating mechanism has been changed. For example, if $\theta_{ijk} = p(X_i = j | X_{\text{pa}(i)} = k)$ is a CPT for node i , then when we compute the sufficient statistics $N_{ijk} = \sum_n \mathbb{I}(X_{ni} = j, X_{n,\text{pa}(i)} = k)$, we exclude cases n where X_i was set externally by intervention, rather than sampled from θ_{ijk} . This technique was first proposed in [Cooper99], and corresponds to Bayesian parameter inference from a set of mutilated models with shared parameters.

The preceding method assumes that we use **perfect interventions**, where we deterministically set a variable to a chosen value. In reality, experimenters can rarely control the state of individual variables. Instead, they can perform actions which may affect many variables at the same time. (This is sometimes called a “**fat hand intervention**”, by analogy to an experiment where someone tries to change a single component of some system (e.g., electronic circuit), but accidentally touching multiple components and thereby causing various side effects.) We can model this by adding the intervention nodes to the DAG (Main Section 4.7.3), and then learning a larger augmented DAG structure, with the constraint that there are no edges between the intervention nodes, and no edges from the “regular” nodes back to the intervention nodes.

For example, suppose we perturb various proteins in a cellular signalling pathway, and measure the resulting phosphorylation status using a technique such as flow cytometry, as in [Sachs05]. An example of such a dataset is shown in Main Figure 20.7(a). Main Figure 20.7(b) shows the augmented DAG that was learned from the interventional flow cytometry data depicted in Main Figure 20.7(a). In particular, we plot the median graph, which includes all edges for which $p(G_{ij} = 1 | \mathcal{D}) > 0.5$. These were computed using the exact algorithm of [Koivisto06]. See [Eaton07aistats] for details.

Since interventional data can help to uniquely identify the DAG, it is natural to try to choose the optimal set of interventions so as to discover the graph structure with as little data as possible. This is a form of active learning or experiment design, and is similar to what scientists do. See e.g., [Murphy01causal; He09; Kalisch2014; Hauser2014; Mueller2017] for some approaches to this problem.

1 **30.5.3 Learning from low-level inputs**

2 In many problems, the available data is quite “low level”, such as pixels in an image, and is believed
3 to be generated by some “higher level” latent causal factors, such as objects interacting in a scene.
4 Learning causal models of this type is known as **causal representation learning**, and combines
5 the causal discovery methods discussed in this section with techniques from latent variable modeling
6 (e.g., VAEs, Main Chapter 21) and representation learning (Main Chapter 32). For more details, see
7 e.g., [Chalupka2017; Scholkopf2021].
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

31 Non-parametric Bayesian models

This chapter is written by Vinayak Rao.

31.1 Dirichlet processes

A **Dirichlet process** (DP) is a nonparametric probability distribution over probability distributions, and is useful as a flexible prior for unsupervised learning tasks like clustering and density modeling [ferguson1973bayesian]. We give more details in the sections below.

31.1.1 Definition of a DP

Let G be a probability distribution or a probability measure (we will use the latter terminology in this chapter) on some space Θ . Recall that a probability measure is a function that assigns values to subsets $T \subseteq \Theta$ satisfying the usual axioms of probability: $0 \leq G(T) \leq 1$, $G(\Theta) = \int_{\Theta} G(\theta) d\theta = 1$, and for disjoint subsets T_1, \dots, T_K of Θ , $G(T_1 \cup \dots \cup T_K) = \sum_{k=1}^K G(T_k)$. Bayesian unsupervised learning now seeks to place a prior on the probability measure G .

We have already seen examples of *parametric* priors over probability measures. As a simple example, consider a Gaussian distribution $\mathcal{N}(\theta|\mu, \sigma^2)$: this is a probability measure on Θ , and by placing priors on the parameters μ and σ^2 , we have a **parametric prior** on probability measures. Mixture models form more flexible priors, allowing multimodality and asymmetry, and are parameterized by the probabilities of the mixture components, as well as their parameters. DPs directly define a probability on probability measures G .

A Dirichlet process is specified by a positive real number α , called the **concentration parameter**, and a probability measure H , called the **base measure**. We write a random measure drawn from a DP as $G \sim \text{DP}(\alpha, H)$. H is typically a standard probability measure on Θ , and forms the mean of the Dirichlet process. That is, if $G \sim \text{DP}(\alpha, H)$, then for any subset T of Θ , $\mathbb{E}[G(T)] = H(T)$. The parameter α measures how concentrated the Dirichlet process is around H , with $\mathbb{V}[G(T)] = \frac{H(T)(1-H(T))}{1+\alpha}$. If Θ is \mathbb{R}^2 , then setting H to the bivariate normal $\mathcal{N}(0, I_2)$ and α to a large value implies a prior belief that G sampled from $\text{DP}(\alpha, H)$ is close to the normal, whereas a small α represents a relatively uninformative prior.

We now define the Dirichlet process more precisely. Let (T_1, \dots, T_K) be a finite partition of Θ , that is, (T_1, \dots, T_K) are disjoint sets whose union is Θ . For a probability measure G , let $(G(T_1), \dots, G(T_K))$ be the vector of probabilities of the elements of this partition. Then $\text{DP}(\alpha, H)$ is a prior over probability measures G satisfying the following requirement: for any finite partition, the

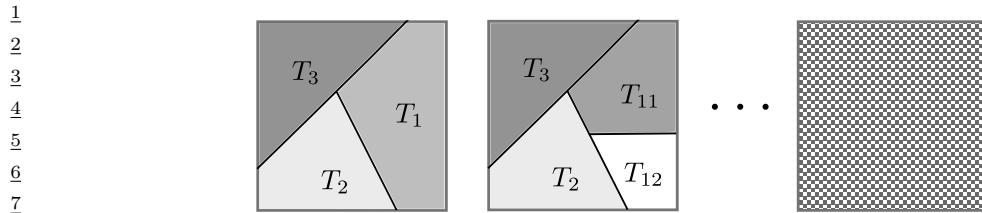


Figure 31.1: Partitions of the unit square. (left) One possible partition into $K = 3$ regions, and (center) A refined partition into $K = 4$ regions. In both figures, the shading of cell T_k is proportional $G(T_k)$, resulting from the same realization of a Dirichlet process. (right) An ‘infinite partition’ of the unit square. The Dirichlet process can informally be viewed as an infinite-dimensional Dirichlet distribution defined on this.

associated vector of probabilities has the following joint Dirichlet distribution:

$$(G(T_1), \dots, G(T_K)) \sim \text{Dir}(\alpha H(T_1), \dots, \alpha H(T_K)). \quad (31.1)$$

Just like the Gaussian process, the DP is defined implicitly through a set of finite-dimensional distributions, in this case through the distribution of G projected onto any finite partition. The finite-dimensional distributions are **consistent** in the following sense: if T_{11} and T_{12} form a partition of T_1 , then one can sample $G(T_1)$ in two ways: directly, by sampling

$$(G(T_1), \dots, G(T_K)) \sim \text{Dir}(\alpha H(T_1), \dots, \alpha H(T_K)) \quad (31.2)$$

or, indirectly, by sampling

$$(G(T_{11}), G(T_{12}), \dots, G(T_K)) \sim \text{Dir}(\alpha H(T_{11}), \alpha H(T_{12}), \dots, \alpha H(T_K)) \quad (31.3)$$

and then setting $G(T_1) = G(T_{11}) + G(T_{12})$. From the properties of the Dirichlet distribution, $G(T_1)$ sampled either way follows the same distribution. This consistency property implies, via Kolmogorov’s extension theorem [**kallenberg2006foundations**], that underlying all finite-dimensional probability vectors for different partitions is a single infinite-dimensional vector that we could informally write as

$$G(d\theta_1), \dots, G(\theta_\infty) \sim \text{Dir}(\alpha H(d\theta_1), \dots, \alpha H(d\theta_\infty)). \quad (31.4)$$

Very roughly, this ‘infinite-dimensional Dirichlet distribution’ is the Dirichlet process. Figure 31.1 sketches this out.

Why is the Dirichlet process, defined in this indirect fashion, useful to practitioners? The answer has to do with conjugacy properties that it inherits from the Dirichlet distribution. One of the simplest unsupervised learning problems seeks to learn an unknown probability distribution G from iid samples $\{\bar{\theta}_1, \dots, \bar{\theta}_N\}$ drawn from it. Consider placing a DP prior on the unknown G . Then given the data, one is interested in the posterior distribution over G , representing the updated probability distribution over G . For a partition (T_1, \dots, T_K) of Θ , an observation falls into the cell z following a multinoulli distribution:

$$z \sim \text{Cat}(G(T_1), \dots, G(T_K)). \quad (31.5)$$

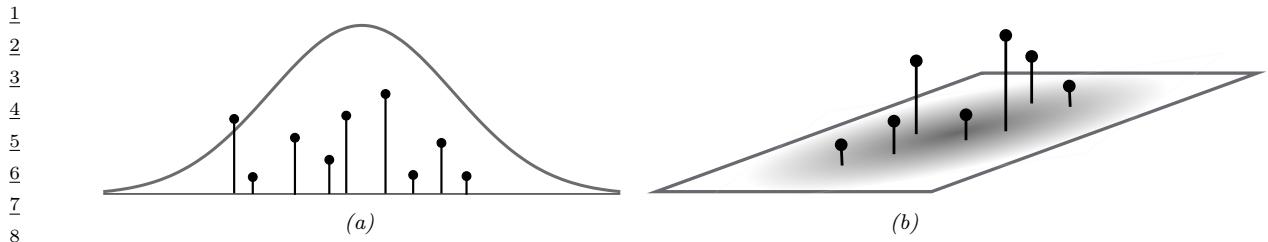


Figure 31.2: Realizations from a Dirichlet process when Θ is (a) the real line, and (b) the unit square. Also shown are the base measures H . In reality, the number of atoms is infinite for both cases.

Under a DP prior on G , $(G(T_1), \dots, G(T_K))$ follows a Dirichlet distribution (equation (31.1)). From the Dirichlet-multinomial conjugacy, the posterior for $(G(T_1), \dots, G(T_K))$ given the observations is

$$(G(T_1), \dots, G(T_K)) | \{\bar{\theta}_1, \dots, \bar{\theta}_N\} \sim \text{Dir}(G(T_1) + \sum_{i=1}^N \mathbb{I}(\bar{\theta}_i \in T_1), \dots, G(T_K) + \sum_{i=1}^N \mathbb{I}(\bar{\theta}_i \in T_K)) \quad (31.6)$$

This is true for any finite partition, so that following our earlier definition, the posterior over G itself is a Dirichlet process, and it is easy to see that:

$$G | \bar{\theta}_1, \dots, \bar{\theta}_N, \alpha, H \sim \text{DP} \left(\alpha + N, \frac{1}{\alpha + N} \left(\alpha H + \sum_{i=1}^N \delta_{\theta_i} \right) \right). \quad (31.7)$$

Thus we see that the DP prior on G is a conjugate prior for iid observations from G , with the posterior distribution over G also a Dirichlet process with concentration parameter $\alpha + N$, and base measure a convex combination of the original base measure H and the empirical distribution of the observations. Note that as N increases, the influence of the original base measure H starts to wane, and the posterior base measure becomes closer and closer to the empirical distribution of the observations. At the same time, the concentration parameter increases, suggesting that the posterior distribution concentrates around the empirical distribution.

31.1.2 Stick breaking construction of the DP

Our discussion so far has been very abstract, with no indication of how to sample either the random measure G or observations from G . We address the first question, giving a constructive definition for the DP known as the **stick-breaking construction** [sethuraman1994constructive].

We first mention that probability measures G sampled from a DP are **discrete with probability one** (see Figure 31.2), taking the form

$$G(\theta) = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k}(\theta). \quad (31.8)$$

Thus G consists of an infinite number of **atoms**, the k th atom located at θ_k , and having weight π_k . Informally, this follows from Equation (31.4), which represents the DP as an infinite-dimensional

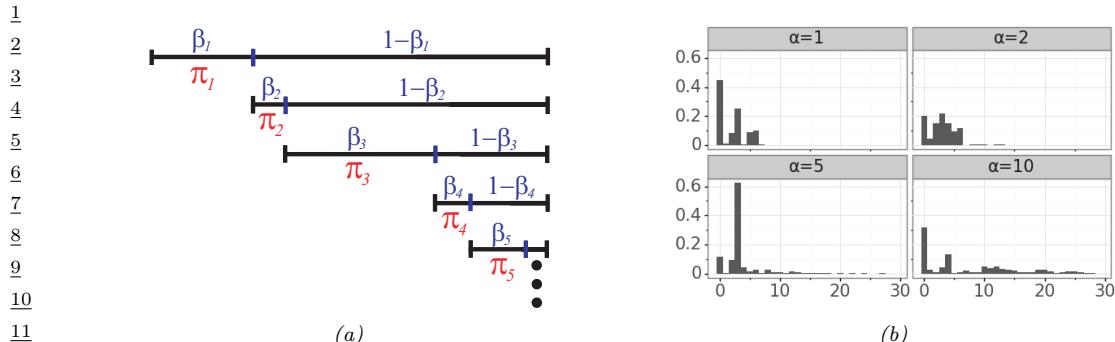


Figure 31.3: Illustration of the stick breaking construction. (a) We have a unit length stick, which we break at a random point β_1 ; the length of the piece we keep is called π_1 ; we then recursively break off pieces of the remaining stick, to generate π_2, π_3, \dots . From Figure 2.22 of [SudderthThesis]. Used with kind permission of Erik Sudderth. (b) Samples of π_k from this process for different values of α . Generated by [stick_breaking_demo.ipynb](#).

but infinitely-sparse Dirichlet distribution (recall that as its parameters become smaller, a Dirichlet distribution concentrates on sparse distributions that are dominated by a few components).

For a DP, the locations θ_k of the atoms are drawn independently from the base measure H , whereas the concentration parameter α controls the distribution of the weights π_k . Observe that the infinite sequence of weights (π_1, π_2, \dots) must add up to one, since G is a probability measure. The weights can be simulated by the following process sketched in Figure 31.3, and known as the **stick-breaking process**. Start with a stick of length 1 representing the total probability mass, and sequentially break off a random Beta($1, \alpha$) distributed fraction of the remaining stick. The k th break forms π_k . In equations, for $k = 1, 2, \dots$,

$$\beta_k \sim \text{Beta}(1, \alpha), \quad \theta_k \sim H, \quad (31.9)$$

$$\pi_k = \beta_k \prod_{l=1}^{k-1} (1 - \beta_l) = \beta_k \left(1 - \sum_{l=1}^{k-1} \pi_l\right) \quad (31.10)$$

Then, setting $G(\theta) = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k}(\theta)$, one can show that $G \sim \text{DP}(\alpha, H)$. The distribution over the weights is often denoted by

$$\pi \sim \text{GEM}(\alpha), \quad (31.11)$$

where **GEM** stands for Griffiths, Engen, and McCloskey (this term is due to [Ewens90]). Some samples from this process are shown in Figure 31.3.

We note that since the number of atoms is infinite, one cannot exactly simulate from a DP in finite time. However, the sequence of weights from the GEM distribution are **stochastically ordered**, having decreasing averages, and the truncation error resulting from terminating after a finite number of steps quickly becoming negligible [ishwaran2001gibbs]. Nevertheless, we will see in the next section that it is possible to simulate samples and make predictions from a DP-distributed probability measure G without any truncation error. This exploits the conjugacy property of the DP.

31.1.3 The Chinese restaurant process (CRP)

Consider a single observation $\bar{\theta}_1$ from a DP-distributed probability measure G . The probability that $\bar{\theta}_1$ lies within a set $T \subseteq \Theta$, marginalizing out the random G , is $\mathbb{E}[G(T)] = H(T)$, the equality following from the definition of the DP. This holds for arbitrary T , which implies that the first observation $\bar{\theta}_1$ is distributed as the base measure of the DP:

$$p(\bar{\theta}_1 = \theta | \alpha, H) = H(\theta). \quad (31.12)$$

Given N observations $\bar{\theta}_1, \dots, \bar{\theta}_N$, the updated distribution over G is still a DP, but now modified as in Equation (31.7). Repeating the same argument, it follows that the $(N + 1)$ st observation is distributed as the base measure of the posterior DP, given by

$$p(\bar{\theta}_{N+1} = \theta | \bar{\theta}_{1:N}, \alpha, H) = \frac{1}{\alpha + N} \left(\alpha H(\theta) + \sum_{k=1}^K N_k \delta_{\bar{\theta}_k}(\theta) \right) \quad (31.13)$$

where N_k is the number of observations equal to $\bar{\theta}_k$. The previous two equations form the basis of what is called the **Pólya urn** or **Blackwell-MacQueen** sampling scheme [blackwell1973ferguson]. This provides a way to exactly produce samples from a DP-distributed random probability measure.

It is often more convenient to work with discrete variables (z_1, \dots, z_N) , with z_i specifying which value of θ_k the i th sample takes. In particular, for the i th observation, $\bar{\theta}_i = \theta_{z_i}$. This allows us to decouple the cluster or partition structure of the dataset (controlled by α) and the cluster parameters (controlled by H). Let us assign the first observation to cluster 1, i.e., $z_1 = 1$. The second observation can either belong to the same cluster as observation 1, or belong to a new cluster, which we call cluster 2. In the former event, $z_2 = 1$, after which z_3 can equal 1 or 2. In the latter event, $z_2 = 2$, and z_3 can equal 1, 2, or 3. Based on the Equation (31.13), we have

$$p(z_{N+1} = z | z_{1:N}, \alpha) = \frac{1}{\alpha + N} \left(\alpha \mathbb{I}(z = K + 1) + \sum_{k=1}^K N_k \mathbb{I}(z = k) \right), \quad (31.14)$$

assuming the first N observations have been assigned to K clusters. This is called the **Chinese restaurant process** or **CRP**, based on the following analogy: observations are customers in a restaurant with an infinite number of tables, each corresponding to a different cluster. Each table has a dish, corresponding to the parameter θ of that cluster. When a customer enters the restaurant, they may choose to join an existing table with probability proportional to the number of people already sitting at this table (i.e., they join table k with probability proportional to N_k); otherwise, with probability proportional to α , they choose to sit at a new table, ordering a new dish by sampling from the base measure H .

The sequence $Z = (z_1, \dots, z_N)$ of cluster assignments is **partition of the integers** 1 to N , and the CRP is a distribution over such partitions. The probability of creating a new table diminishes as the number of observations increases, but is always non-zero, and one can show that the number of occupied tables K approaches $\alpha \log(N)$ as $N \rightarrow \infty$ almost surely. The fact that currently occupied tables are more likely to get new customers is sometimes called a **rich get richer** phenomenon. It is important to recognize that despite being defined as a sequential process, the CRP is an **exchangeable process**, with partition probabilities that are independent of the observation indices.

Indeed, it is easy to show that the probability of a partition of N integers into K clusters with sizes N_1, \dots, N_K is

$$p(N_1, \dots, N_k) = \frac{\alpha^{K-1}}{[\alpha+1]_1^N} \prod_{k=1}^K N_k! \quad (31.15)$$

Here, $[\alpha+1]_1^N = \prod_{i=0}^{N-1} (\alpha+1+i)$ is the rising factorial. Equation (31.15) depends only on the cluster sizes, and is called the Ewens sampling formula [ewens1972sampling]. Exchangeability implies that the probability that the first two customers sit at the same table is the same as the probability that the first and last sit at the same table. Similarly all customers have the same probability of ending up in a cluster of size S . The fact that the first customer can only belong to cluster 1 (i.e., that $z_1 = 1$) does not contradict exchangeability and reflects the fact that the cluster indices are chosen arbitrarily. This disappears if we index clusters by their associated parameter θ_k .

31.2 Dirichlet process mixture models

Real-world datasets are often best modeled by continuous probability densities. By contrast, a sample G from a DP is discrete with probability one, and sampling observations from G will result in repeated values, making it inappropriate for many applications. However, the discrete structure of G is useful in clustering applications, as a prior for the latent variables underlying the observed datapoints. In particular, z_i and $\bar{\theta}_i$ can represent the cluster assignment and cluster parameter of the i 'th datapoint, whose value \mathbf{x}_i is a draw from some parametric distribution $F(\mathbf{x}|\boldsymbol{\theta})$ indexed by $\boldsymbol{\theta}$, with base measure H . The resulting model follows along the lines of a standard mixture model, but now is an **infinite mixture model**, consisting of an infinite number of components or clusters, one for each atom in G .

A very common setting when $\mathbf{x}_i \in \mathbb{R}^d$ is to set F to be the multivariate normal distribution, $\boldsymbol{\theta} = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$, and H to be the normal-inverse-Wishart distribution. Then, each of the infinite clusters has an associated mean and covariance matrix, and to generate a new observation, one picks cluster k with probability π_k , and simulates from a normal with mean $\boldsymbol{\mu}_k$ and covariance $\boldsymbol{\Sigma}_k$. See Figure 31.4 for some samples from this model.

We discuss DP mixture models (DPMM) in more detail below.

31.2.1 Model definition

We define the DPMM model as follows:

$$\boldsymbol{\pi} \sim GEM(\alpha), \quad \boldsymbol{\theta}_k \sim H, \quad k = 1, 2, \dots \quad (31.16)$$

$$z_i \sim \boldsymbol{\pi}, \quad \mathbf{x}_i \sim F(\boldsymbol{\theta}_{z_i}), \quad i = 1, \dots, N. \quad (31.17)$$

Equivalently, we can write this as

$$G \sim DP(\alpha, H) \quad (31.18)$$

$$\bar{\boldsymbol{\theta}}_i \sim G, \quad \mathbf{x}_i \sim F(\bar{\boldsymbol{\theta}}_i), \quad i = 1, \dots, N. \quad (31.19)$$

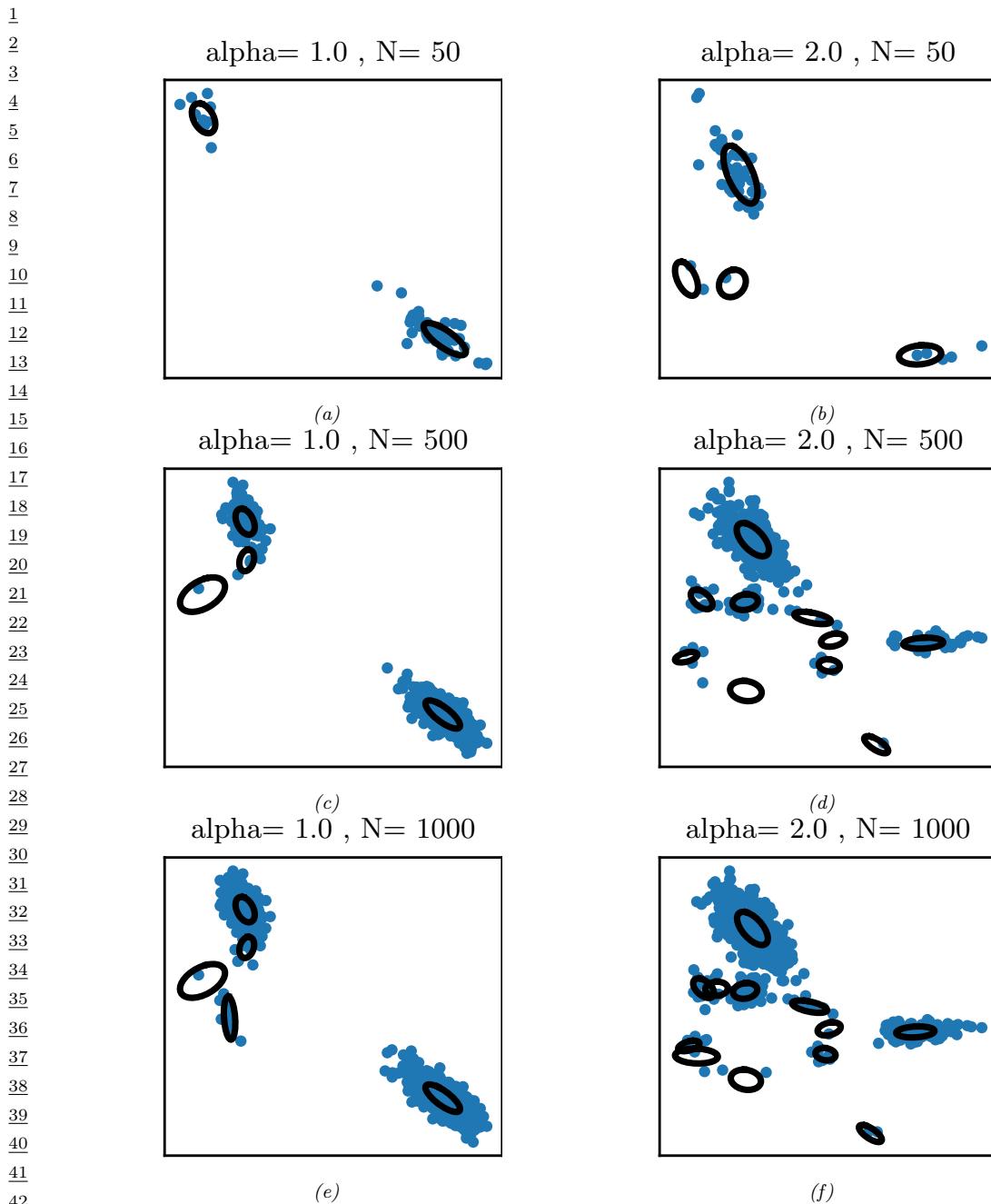


Figure 31.4: Some samples from a Dirichlet process mixture model of 2d Gaussians, with concentration parameter $\alpha = 1$ (left column) and $\alpha = 2$ (right column). From top to bottom, we show $N = 50$, $N = 500$ and $N = 1000$ samples. Generated by [dp_mixgauss_sample.ipynb](#).

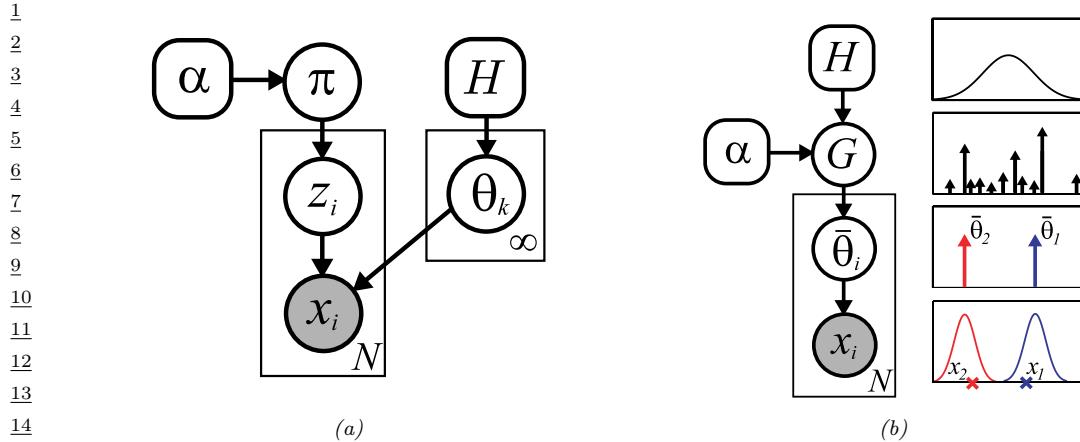


Figure 31.5: Two views of N observations sampled from a DP mixture model. Left: representation where cluster indicators are sampled from the GEM-distributed distribution π . Right: representation where parameters are samples from the DP-distributed random measure G . The rightmost picture illustrates the case where $N = 2$, θ is real-valued with a Gaussian prior $H(\cdot)$, and $F(x|\theta)$ is a Gaussian with mean θ and variance σ^2 . We generate two parameters, $\bar{\theta}_1$ and $\bar{\theta}_2$, from G , one per data point. Finally, we generate two data points, x_1 and x_2 , from $\mathcal{N}(\bar{\theta}_1, \sigma^2)$ and $\mathcal{N}(\bar{\theta}_2, \sigma^2)$. From Figure 2.24 of [SudderthThesis]. Used with kind permission of Erik Sudderth.

G and F together define the infinite mixture model: $G_F(\mathbf{x}) \sim \sum_{k=1}^{\infty} \pi_k F(\mathbf{x}|\theta_k)$. If $F(\mathbf{x}|\theta)$ is continuous, then so is $G_F(\mathbf{x})$, and the Dirichlet process mixture model serves as a nonparametric prior over continuous distributions or probability densities.

Figure 31.5 illustrates two graphical models that summarize this, corresponding to the two sets of equations above. The first generates the set of weights (π_1, π_2, \dots) from the GEM distribution, along with an infinite collection of cluster parameters $(\theta_1, \theta_2, \dots)$. It then generates observations by first sampling a cluster indicator z_i from $F(\theta_{z_i})$. The second graphical model simulates a random measure G from the DP. It generates observations by directly simulating a parameter $\bar{\theta}_i$ from G , and then simulating x_i from $F(\bar{\theta}_i)$. The infinite mixture model can be viewed as the limit of a K -component finite mixture model with a $\text{Dir}(\alpha/K, \dots, \alpha/K)$ prior on the mixture weights (π_1, \dots, π_K) and with mixture parameters $\theta_1, \dots, \theta_K$, as $K \rightarrow \infty$ [Rasmussen00; Neal00]. Producing exact samples (x_1, \dots, x_N) from this model involves one additional step to the Chinese restaurant process: after selecting a table (cluster) z_i with its associate dish (parameter) θ_{z_i} , the i 'th customer now samples an observation from the distribution $F(\theta_{z_i})$.

40

31.2.2 Fitting using collapsed Gibbs sampling

Given a dataset of observations, one is interested in the posterior distribution $p(G, z_1, \dots, z_N | \mathbf{x}_1, \dots, \mathbf{x}_N, \alpha, H)$, or equivalently, $p(\pi, \theta_1, \theta_2, \dots, z_1, \dots, z_N | \mathbf{x}_1, \dots, \mathbf{x}_N, \alpha, H)$. The most common way to fit a DPMM is via Markov chain Monte Carlo (MCMC), producing samples by constructing a Markov chain that targets this posterior distribution. We describe a **collapsed Gibbs sampler** based on the

47

Chinese restaurant process that marginalizes out the infinite-dimensional random measure G , and that targets the distribution $p(z_1, \dots, z_N | \mathbf{x}_1, \dots, \mathbf{x}_N, \alpha, H)$ summarizing all clustering information. It produces samples from this distribution by cycling through each observation \mathbf{x}_i , and updating its assigned cluster z_i , conditioned on all other variables. Write \mathbf{x}_{-i} for all observations other than the i th observation, and \mathbf{z}_{-i} for their cluster assignments. Then we have

$$p(z_i = k | \mathbf{z}_{-i}, \mathbf{x}, \alpha, H) \propto p(z_i = k | \mathbf{z}_{-i}, \alpha) p(\mathbf{x}_i | \mathbf{x}_{-i}, z_i = k, \mathbf{z}_{-i}, H) \quad (31.20)$$

By exchangeability, each observation can be treated as the last customer to enter the restaurant. Hence the first term is given by

$$p(z_i | \mathbf{z}_{-i}, \alpha) = \frac{1}{\alpha + N - 1} \left(\alpha \mathbb{I}(z_i = K_{-i} + 1) + \sum_{k=1}^{K_{-i}} N_{k,-i} \mathbb{I}(z_i = k) \right) \quad (31.21)$$

where $N_{k,-i}$ is the number of observations in cluster k , and K_{-i} is the number of clusters used by \mathbf{x}_{-i} , both obtained after removing observation i , eliminating empty clusters, and renumbering clusters.

To compute the second term, $p(\mathbf{x}_i | \mathbf{x}_{-i}, z_i = k, \mathbf{z}_{-i}, H)$, let us partition the data \mathbf{x}_{-i} into clusters based on \mathbf{z}_{-i} . Let $\mathbf{x}_{-i,c} = \{\mathbf{x}_j : z_j = c, j \neq i\}$ be the datapoints assigned to cluster c . If $z_i = k$, then x_i is conditionally independent of all the datapoints except those assigned to cluster k . Hence,

$$p(\mathbf{x}_i | \mathbf{x}_{-i}, \mathbf{z}_{-i}, z_i = k) = p(\mathbf{x}_i | \mathbf{x}_{-i,k}) = \frac{p(\mathbf{x}_i, \mathbf{x}_{-i,k})}{p(\mathbf{x}_{-i,k})}, \quad (31.22)$$

where

$$p(\mathbf{x}_i, \mathbf{x}_{-i,k}) = \int p(\mathbf{x}_i | \boldsymbol{\theta}_k) \left[\prod_{j \neq i: z_j=k} p(\mathbf{x}_j | \boldsymbol{\theta}_k) \right] H(\boldsymbol{\theta}_k) d\boldsymbol{\theta}_k \quad (31.23)$$

is the marginal likelihood of all the data assigned to cluster k , including i , and $p(\mathbf{x}_{-i,k})$ is an analogous expression excluding i . Thus we see that the term $p(\mathbf{x}_i | \mathbf{x}_{-i}, \mathbf{z}_{-i}, z_i = k)$ is the posterior predictive distribution for cluster k evaluated at \mathbf{x}_i .

If $z_i = k^*$, corresponding to a new cluster, we have

$$p(\mathbf{x}_i | \mathbf{x}_{-i}, \mathbf{z}_{-i}, z_i = k^*) = p(\mathbf{x}_i) = \int p(\mathbf{x}_i | \boldsymbol{\theta}) H(\boldsymbol{\theta}) d\boldsymbol{\theta} \quad (31.24)$$

which is just the prior predictive distribution for a new cluster evaluated at \mathbf{x}_i .

The overall sampler is sometimes called ‘‘Algorithm 3’’ (from [Neal00]). Algorithm 31.1 provides the pseudocode. The algorithm is very similar to collapsed Gibbs for finite mixtures except that we have to consider the case $z_i = k^*$. Note that in order to evaluate the integrals in Equation (31.23) and Equation (31.24), we require the base measure H to be conjugate to the likelihood F . For example, if we use an NIW prior for the Gaussian likelihood, we can use the results from ?? to compute the predictive distributions. Extensions to the case of non-conjugate priors are discussed in [Neal00].

2

Algorithm 31.1: Collapsed Gibbs sampler for DP mixture model

```

1   foreach  $i = 1 : N$  in random order do
2     Remove  $\mathbf{x}_i$ 's sufficient statistics from old cluster  $z_i$ 
3     foreach  $k = 1 : K$  do
4       Compute  $p_k(\mathbf{x}_i) = p(\mathbf{x}_i | \mathbf{x}_{-i}(k))$ 
5       Set  $N_{k,-i} = \dim(\mathbf{x}_{-i}(k))$ 
6       Compute  $p(z_i = k | \mathbf{z}_{-i}, \mathcal{D}) = \frac{N_{k,-i}}{\alpha + N - 1} p_k(\mathbf{x}_i)$ 
7     Compute  $p(z_i = * | \mathbf{z}_{-i}, \mathcal{D}) = \frac{\alpha}{\alpha + N - 1} p(\mathbf{x}_i)$ 
8     Normalize  $p(z_i | \cdot)$ 
9     Sample  $z_i \sim p(z_i | \cdot)$ 
10    Add  $\mathbf{x}_i$ 's sufficient statistics to new cluster  $z_i$ 
11    If any cluster is empty, remove it and decrease  $K$ 
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
```

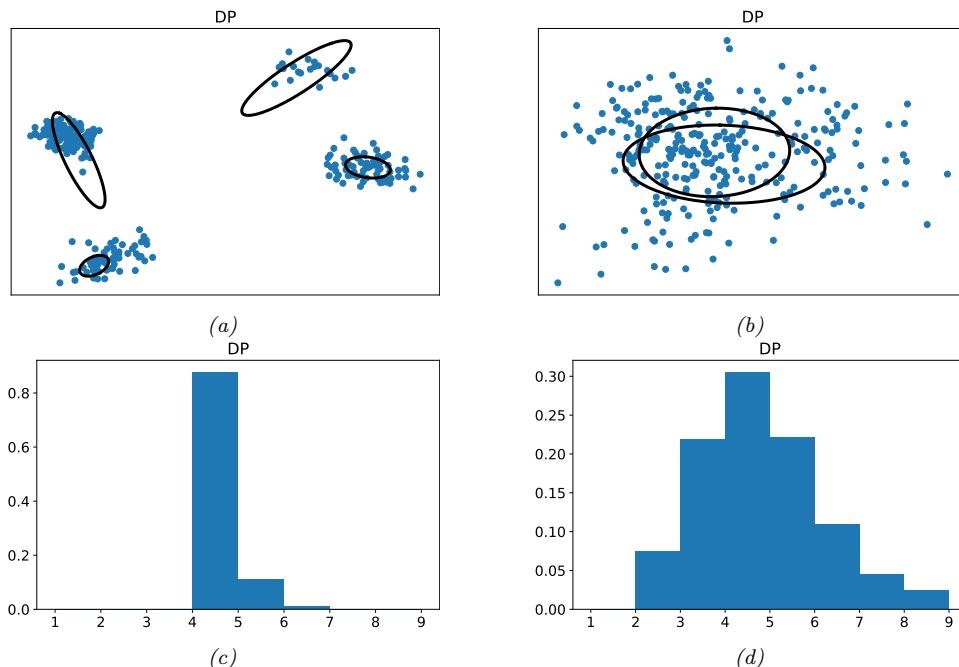


Figure 31.6: Output of the DP mixture model fit using Gibbs sampling to two different datasets. Left column: dataset with 4 clear clusters. Right column: dataset with an unclear number of clusters. Top row: single sample from the Markov chain. Bottom row: empirical fraction of times a given number of cluster is used, computed across all samples from the chain. Generated by [dp_mixgauss_sample.ipynb](#).

31.2.3 Fitting using variational Bayes

This section is written by Xinglong Li.

In this section, we discuss how to fit a DP mixture model using mean field variational Bayes (??), as described in [blei2006variational].

Given samples $\mathbf{x}_1, \dots, \mathbf{x}_N$ from DP mixture, the mean field variational inference (MFVI) algorithm is based on the stick-breaking representation of the DP mixture. The target of the inference is the joint posterior distribution of the beta random variables $\boldsymbol{\beta} = \{\beta_1, \beta_2, \dots\}$ in the stick-breaking construction of the DP, the locations $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots\}$ of atoms, and the cluster assignments $\mathbf{z} = \{z_1, \dots, z_N\}$:

$$\underline{w} = \{\boldsymbol{\beta}, \boldsymbol{\theta}, \mathbf{z}\} \quad (31.25)$$

The hyperparameters are the concentration parameter of the DP and the parameter of the conjugate base distribution of $\boldsymbol{\theta}$:

$$\underline{\lambda} = \{\alpha, \boldsymbol{\eta}\} \quad (31.26)$$

The variational inference algorithm minimizes the KL-divergence between $q_{\psi}(\underline{w})$ and $p(\underline{w}|\mathbf{x}, \underline{\lambda})$:

$$D_{\text{KL}}(q_{\psi}(\underline{w}) \parallel p(\underline{w}|\mathbf{x}, \underline{\lambda})) = \mathbb{E}_q[\log q_{\psi}(\underline{w})] - \mathbb{E}_q[\log p(\underline{w}, \mathbf{x}|\underline{\lambda})] + \log p(\mathbf{x}|\underline{\lambda}) \quad (31.27)$$

Minimizing the KL divergence is equivalent to maximizing the evidence lower bound (ELBO):

$$\underline{L} = \mathbb{E}_q[\log p(\underline{w}, \mathbf{x}|\underline{\lambda})] - \mathbb{E}_q[\log q_{\psi}(\underline{w})] \quad (31.28)$$

$$= \mathbb{E}_q[\log p(\boldsymbol{\beta}|\alpha)] + \mathbb{E}_q[\log p(\boldsymbol{\theta}|\boldsymbol{\eta})] + \sum_{n=1}^N (\mathbb{E}_q[\log p(z_n|\boldsymbol{\beta})] + \mathbb{E}_q[\log p(\mathbf{x}_n|z_n)]) \quad (31.29)$$

$$- \mathbb{E}_q[\log q_{\psi}(\boldsymbol{\beta}, \boldsymbol{\theta}, \mathbf{z})] \quad (31.30)$$

To deal with the infinite parameters in $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$, the variational inference algorithm truncates the DP by fixing a value T and setting $q(\beta_T = 1) = 1$, which implies that $\pi_t = 0$ for $t > T$. Therefore, $q_{\psi}(\underline{w})$ in the MFVI for DP mixture models factorizes into

$$q_{\psi}(\boldsymbol{\beta}, \boldsymbol{\theta}, \mathbf{z}) = \prod_{t=1}^{T-1} q_{\gamma_t}(\beta_t) \prod_{t=1}^T q_{\tau_t}(\boldsymbol{\theta}_t) \prod_{n=1}^N q_{\phi_n}(z_n), \quad (31.31)$$

where $q_{\gamma_t}(\beta_t)$ is the beta distribution with parameters $\{\gamma_{t,1}, \gamma_{t,2}\}$, $q_{\tau_t}(\boldsymbol{\theta}_t)$ is the exponential family distribution with natural parameters $\boldsymbol{\tau}_t$, and $q_{\phi_n}(z_n)$ is the categorical distribution of the cluster assignment of observation \mathbf{x}_n , with $q(z_n = t) = \phi_{n,t}$. The free variational parameters are

$$\psi = \{\gamma_1, \dots, \gamma_{T-1}, \tau_1, \dots, \tau_T, \phi_1, \dots, \phi_N\}. \quad (31.32)$$

Notice that only $q_{\psi}(\underline{w})$ is truncated, the true posterior $p(\underline{w}, \mathbf{x}|\underline{\lambda})$ from the model need not be truncated when minimizing the KL.

The MFVI can be optimized via the coordinate ascent algorithm, and the closed form update in each step exists when the base measure of the DP is conjugate to the likelihood of observations. In

1 particular, suppose that conditional distribution of \mathbf{x}_n conditioned on z_n and $\boldsymbol{\theta}$ is an exponential
2 family distribution (??):

3

$$p(\mathbf{x}_n|z_n, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots) = h(\mathbf{x}_n) \exp\{\boldsymbol{\theta}_{z_n}^\top \mathbf{x}_n - a(\boldsymbol{\theta}_{z_n})\} \quad (31.33)$$

4 where \mathbf{x}_n is the sufficient statistic for the natural parameter $\boldsymbol{\theta}_{z_n}$. Therefore, the conjugate base
5 distribution is

6

$$p(\boldsymbol{\theta}|\boldsymbol{\eta}) = h(\boldsymbol{\theta}) \exp(\boldsymbol{\eta}_1^\top \boldsymbol{\theta} + \eta_2(-a(\boldsymbol{\theta})) - a(\boldsymbol{\eta})), \quad (31.34)$$

7 where $\boldsymbol{\eta}_1$ contains the first $\dim(\boldsymbol{\theta})$ components and η_2 is a scalar. See Algorithm 31.2 for the resulting
8 pseudocode.

9 Extensions of this method to infer the hyperparameters, $\boldsymbol{\lambda} = \{\alpha, \boldsymbol{\eta}\}$, can be found in the appendix
10 of [blei2006variational].

11

12 **Algorithm 31.2:** Variational inference for DP mixture model

13 1 Initialize the variational parameters:
14 2 ϕ_{nt} is membership probability of \mathbf{x}_n in cluster t ;
15 3 $\boldsymbol{\tau}_t$ are the natural parameters for cluster t ;
16 4 $\boldsymbol{\gamma}_t$ are the parameters for the stick breaking distribution.
17 5 **while** not converged **do**
18 **foreach** $\boldsymbol{\gamma}_t$ **do**
19 Update the beta distribution $q_{\boldsymbol{\gamma}_t}(\beta_t)$;
20 $\gamma_{t,1} = 1 + \sum_n \phi_{n,t}$
21 $\gamma_{t,2} = \alpha + \sum_n \sum_{j=t+1}^T \phi_{n,j}$
22 **foreach** $\boldsymbol{\tau}_t$ **do**
23 Update the exponential family distribution $q_{\boldsymbol{\tau}_t}(\boldsymbol{\theta}_t)$ given sufficient statistics $\{\mathbf{x}_n\}$;
24 $\boldsymbol{\tau}_{t,1} = \boldsymbol{\eta}_1 + \sum_n \phi_{n,t} \mathbf{x}_n$
25 $\tau_{t,2} = \eta_2 + \sum_n \phi_{n,t}$
26 **foreach** $\phi_{n,t}$ **do**
27 Update the categorical distribution $q_{\phi_n}(z_n)$ for each observation;
28 $\phi_{n,t} \propto \exp(S_t)$
29 $S_t = \mathbb{E}_q[\log \beta_t] + \sum_{i=1}^{t-1} \mathbb{E}_q[\log(1 - \beta_i)] + \mathbb{E}_q[\boldsymbol{\theta}_t]^\top \mathbf{x}_n - \mathbb{E}_q[a(\boldsymbol{\theta}_t)]$

30

31

32 **31.2.4 Other fitting algorithms**

33 While collapsed Gibbs sampling is the simplest approach to posterior inference for DPMMs, a variety
34 of other methods have been proposed as well. One popular class of MCMC samplers works with
35 the stick-breaking representation of the DP instead of CRP, instantiating the random measure
36 G [ishwaran2001gibbs]. The sampler then proceeds by sampling the cluster assignments \mathbf{z} given
37 G , and then G given \mathbf{z} . An advantage of this is that the cluster assignments can be updated in
38 parallel, unlike the CRP, where they are updated sequentially. To be feasible however, these methods
39

40

require truncating G to a finite number of atoms, though the resulting approximation error can be quite small. The posterior approximation error can be eliminated altogether by slice-sampling methods [**kalli2011slice**], that work with random truncation levels.

Alternatives to MCMC also exist. [**Daume07**] shows how one can use A* search and beam search to quickly find an approximate MAP estimate. [**Mansinghka07**] discusses how to fit a DPMM online using particle filtering, which is a like a stochastic version of beam search. This can be more efficient than Gibbs sampling, particularly for large datasets.

In Section 31.2.3 we discussed an approach based on mean field variational inference. A variety of other variational approximation methods have been proposed as well, for example [**Kurihara06; teh2008collapsed; Zobay09; wang2012truncation**].

31.2.5 Choosing the hyper-parameters

An important issue is how to set the model hyper-parameters. These include the DP concentration parameter α , as well as any parameters λ of the base measure H . For the DP, the value of α does not have much impact on predictive accuracy, but has quite a strong affect the number of clusters. One approach is to put a gamma prior for α , and then form its posterior, $p(\alpha|K, N, a, b)$ [**Escobar95**]. Simulating α given the cluster assignments z is quite straightforward, and can be incorporated into the earlier Gibbs sampler. The same is the case with the hyper-parameters λ [**Rasmussen00**]. Alternatively, one can use empirical Bayes [**McAuliffe06**] to fit rather than sample these parameters.

31.3 Generalizations of the Dirichlet process

Dirichlet process mixture models are flexible nonparametric models of continuous probability densities, and if set up with a little care, can possess important frequentist properties like **asymptotic consistency**: with more and more observations, the posterior distribution concentrates around the ‘true’ data generating distribution, with very little assumed about the this distribution. Nevertheless, DPs still represent very strong prior information, especially in clustering applications. We saw that the number of clusters in a dataset of size N a priori is around $\alpha \log N$. As indicated by Equation (31.15), not just the number of clusters, but also the distribution of their sizes is controlled by a single parameter α . The resulting clustering model is thus quite inflexible, and in many cases, inappropriate. Two examples from machine learning that highlight its limitations are applications involving text and image data. Here, it has been observed empirically that the number of unique words in a document, the frequency of word usage, the number of objects in an image, or the number of pixels in an object follow power-law distributions. Clusterings sampled from the CRP do not produce this property, and the resulting model mismatch can result in poor predictive performance.

31.3.1 Pitman-Yor process

A popular generalization of the Dirichlet process is the Pitman-Yor process [**pitman1997two**] (also called the two-parameter Poisson-Dirichlet process). Written at $\text{PYP}(\alpha, d, H)$, the Pitman-Yor process includes an additional **discount parameter** d which must be greater than 0. The concentration parameter can now be negative, but must satisfy $\alpha \geq -d$. As with the DP, a sample G from a PYP is a random probability measure that is discrete almost surely. It has a stick-breaking representation that generalizes that of the DP: again, we start with a stick of length 1, but now at

1 step k , a random $\text{Beta}(1 - d, \alpha + kd)$ fraction of the remaining probability mass is broken off, so that
2 G is written as
3

$$\frac{4}{5} G = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k}, \tag{31.35}$$

$$\frac{7}{8} \beta_k \sim \text{Beta}(1 - d, \alpha + kd), \quad \theta_k \sim H, \tag{31.36}$$

$$\frac{9}{10} \pi_k = \beta_k \prod_{l=1}^{k-1} (1 - \beta_l) = \beta_k (1 - \sum_{l=1}^{k-1} \pi_l). \tag{31.37}$$

11 Because G is discrete, once again observations $\bar{\theta}_1, \dots, \bar{\theta}_d$ sampled iid from G will possess a clustering
12 structure. These can directly be sampled following a sequential process that generalizes the CRP.
13 Now, when a new customer enters the restaurant, they join an existing table with N_k customers
14 with probability proportional to $(N_k - d)$, and create a new table with probability proportional to
15 $\alpha + Kd$, where K is the number of clusters.
16

17 Observe that the Dirichlet process is a special instance of the Pitman-Yor process, corresponding
18 to d equal to 0. Non-zero settings of d counteract the rich-get-richer dynamics of the DP to some
19 extent, increasing the probability of creating new clusters. The more clusters there are present in a
20 dataset, the higher the probability of creating a new cluster (relative to the Dirichlet process). This
21 behavior means that a large number of clusters, as well as a few very large clusters are more likely
22 under the PYP than the DP.

23 An even more general class of probability measures are the so-called Gibbs-type priors [**de2013gibbs**].
24 Under such a prior, given N observations, the probability these are clustered into K clusters, the k th
25 having n_k observations, is

$$\frac{26}{27} p(n_1, \dots, n_k) = V_{N,K} \prod_{k=1}^K (\sigma - 1)_{N_k - 1}, \tag{31.38}$$

29 where $(\sigma)_n = \sigma(\sigma + 1)\dots(\sigma + n - 1)$, and for non-negative weights $V_{N,K}$ satisfying $V_{N,K} =$
30 $(N - \sigma K)V_{N+1,K} + V_{N+1,K+1}$ and $V_{1,1} = 1$. This definition ensures that the probability over
31 partitions is consistent, exchangeable and tractable, and includes the DP and PYP as special cases.
32 Besides these two, Gibbs-type priors include settings where the number of components (or the number
33 of atoms in the random measures) are random but bounded. Recall that DP and PYP mixture models
34 are infinite mixture models, with the number of components growing with the number of observations.
35 A sometimes undesirable feature of these models is that if a dataset is generated from a finite number
36 of clusters, these models will not recover the true number of clusters [**miller2014inconsistency**].
37 Instead, the estimated number of clusters will increase with the size of the dataset, resulting in
38 redundant clusters that are located very close to each other. Gibbs-type priors with almost surely
39 bounded number of components can learn the true number of clusters while still remaining reasonably
40 tractable: so long as one can calculate the $V_{N,K}$ terms, MCMC for all these models can be carried
41 out by modifications of the CRP-based sampler described earlier.
42

43 31.3.2 Dependent random probability measures

44 Dirichlet processes, and more generally, random probability measures, have also been generalized
45 from settings with a single set of observations to those involving grouped observations, or observations
46
47

indexed by covariates. Consider T sets of observations $\{\mathbf{X}^1, \dots, \mathbf{X}^T\}$, each perhaps corresponding to a different country, a different year, or more abstractly, a different set of observed covariates. There are two immediate (though inadequate) ways to model such data: 1) by pooling all datasets into a single dataset, modeled as drawn from a DP or DPMM-distributed random probability measure G , or 2) by treating each group as independent, having its own random probability measure G^t . The first approach fails to capture differences between the groups, while the second ignores similarities between the groups (e.g., shared clusters). Dependent random probability measures seek a compromise between these extremes, allowing statistical information to be shared across different groups.

A seminal paper in the machine learning literature that addresses this problem is the **hierarchical Dirichlet process** (HDP) [teh06_hdp]. The basic idea here is simple: each group has its own random measure drawn from a Dirichlet process $\text{DP}(\alpha, H)$. The twist now is that the base measure H itself is random, in fact it is itself drawn from a Dirichlet process. Thus, the overall generative process is

$$H \sim \text{DP}(\alpha_0, H_0), \quad (31.39)$$

$$G^t \sim \text{DP}(\alpha, H), \quad t \in 1, \dots, T \quad (31.40)$$

$$\bar{\theta}_i^t \sim G^t, \quad i \in 1, \dots, N_t, \quad t \in 1, \dots, T. \quad (31.41)$$

The $\bar{\theta}_i^t$'s might be the observations themselves, or the latent parameter underlying each observation, with $\mathbf{x}_i^t \sim F(\bar{\theta}_i^t)$.

Recall that if a probability measure G^1 is drawn from $\text{DP}(\alpha, H)$, its atoms are drawn independently from the base measure H . In the HDP, H , which is a draw from a DP, is discrete, so that some atoms of G^1 will sit on top of each other, becoming a single atom. More importantly, all measures G^t will share the same infinite set of locations: each atom of H will eventually be sampled to form a location of an atom of each G^t . This will allow the same clusters to appear in all groups, though they will have different weights. Moreover, a big cluster in one group is a priori likely to be a big cluster in another group, as underlying both is a large atom in H . Since the common probability measure H itself is random, its components (both weights as well as locations) will be learned from data. Despite its apparent complexity, it is fairly easy to develop an analogue of the Chinese restaurant process for the HDP, allowing us to sample observations directly without having to instantiate any of the infinite-dimensional measures. This is called the Chinese restaurant franchise, and essentially amounts to each group having its own Chinese restaurant with the following modification: whenever a customer sits at a new table and orders a dish, that dish itself is sampled from an upper CRP common to all restaurants. It is also possible to develop stick-breaking representations of the HDP.

The HDP has found wide application in the machine learning literature. A common application is document modeling, where the location of each atom is a **topic**, corresponding to some distribution over words. Rather than bounding the number of topics, there are an infinite number of topics, with document d having its own distribution over topics (represented by a measure G^d). By tying all the G^d 's together through an HDP, different documents can share the same topics while emphasizing different topics.

Another application involves **infinite-state hidden Markov models**, also called **HDP-HMM**. Recall from ?? that a Markov chain is parameterized by a transition matrix, with row r giving the distribution over the next state if the current state is r . For an infinite-state HMM, this is an infinite-by-infinite matrix, with row r corresponding to a distribution G^r with an infinite number of

1 atoms. The different G^r 's can be tied together by modeling them with an HDP, so that atoms from
 2 each correspond to the same states [Fox08].
 3

4 Hierarchical nonparametric models of this kind can be constructed with other measures, such as the
 5 Pitman-Yor process. For certain parameter settings, the PYP possesses convenient marginalization
 6 properties that the DP does not [Wood2009]. In particular, simulating a random probability
 7 measure (RPM) in the following two steps

$$8 \quad G_0|H_0 \sim \text{PYP}(0, d_0, H_0), \quad (31.42)$$

$$9 \quad G_1|G_0 \sim \text{PYP}(0, d_1, G_0) \quad (31.43)$$

11 is equivalent to directly simulating G_1 without instantiating G_0 as below:
 12

$$13 \quad G_1|H_0 \sim \text{PYP}(0, d_0 d_1, H_0). \quad (31.44)$$

15 This marginalization property (also called **coagulation**) allows deep hierarchies of dependent RPMs,
 16 with only a smaller, dataset-dependent subset of G 's having to be instantiated. In the **sequence**
 17 **memoizer** of [wood2011sequence], the authors model sequential data (e.g., text) with hierarchies
 18 that are *infinitely* deep, but with only a finite number of levels ever having to be instantiated. If
 19 needed, intermediate random measures can be instantiated by a dual **fragmentation** operator.

20 Deeper hierarchies like those described above allow more refined modeling of similarity between
 21 different groups. Under the original HDP, the groups themselves are exchangeable, with no subset of
 22 groups a priori more similar to each other than to others. For instance, suppose each of the measures
 23 G_1, \dots, G_T correspond to distributions over topics in different scientific journals. Modeling the G_t 's
 24 with an HDP allows statistical sharing across journals (through shared clusters and similar cluster
 25 probabilities), but does not regard some journals as a priori more similar than others. If one had
 26 further information, e.g., that some are physics journals and the rest are biology journals, then one
 27 might add another level to the hierarchy. Now rather than each journal having a probability measure
 28 G^t drawn from a $\text{DP}(\alpha, H)$, physics and biology journals have their own base measures H_p and H_b
 29 that allow statistical sharing among physics and biology journals respectively. Like the HDP, these
 30 are draws from a DP with base measure H . To allow sharing *across* disciplines, H is again random,
 31 drawn from a DP with base measure H_0 . Overall, if there are D disciplines, $1, 2, \dots, D$, the overall
 32 model is

$$33 \quad H \sim \text{DP}(\alpha_0, H_0), \quad (31.45)$$

$$34 \quad H^d \sim \text{DP}(\alpha_1, H), \quad d \in 1, \dots, D \quad (31.46)$$

$$35 \quad G^{t,d} \sim \text{DP}(\alpha_2, H^d), \quad t \in 1, \dots, T_d \quad (31.47)$$

$$36 \quad \theta_i^{t,d} \sim G^{t,d} \quad d \in 1, \dots, D, \quad t \in 1, \dots, T_d, \quad i \in 1, \dots, N_t \quad (31.48)$$

39 One might add further levels to the hierarchy given more information (e.g., if disciplines are grouped
 40 into physical sciences, social sciences and humanities).

42 Dependent random probability measures can also be indexed by covariates in some continuous
 43 space, whether time, space, or some Euclidean space or manifold. This space is typically endowed
 44 with some distance or similarity function, and one expects that measures with similar covariates are
 45 a priori more similar to each other. Thus, G^t might represent a distribution over topics in year t ,
 46 and one might wish to model G^t evolving gradually over time. There is rich history of dependent
 47

random probability measures in statistics literature, starting from [maceachern1999dependent]. A common requirement in such models is that at any fixed time t , the marginal distribution of G^t follows some well specified distribution, e.g., a Dirichlet process, or a PYP. Approaches exploit the stick-breaking representation, the CRP representation or something else like the Poisson structure underlying such models (see Section 31.6).

As a simple and early example, recall the stick-breaking construction from Section 31.1.2, where a random probability measure is represented by an infinite collection of pairs (β_k, θ_k) . To construct a family of RPMs indexed by some covariate t , we need a family of such sets (β_k^t, θ_k^t) for each of the possibly infinite values of t . To ensure each G_t is marginally DP distributed with concentration parameter α and base measure H , we need that for each t , the β_k^t 's are marginally iid draws from a Beta(1, α), and the θ_k^t 's iid draws from H . Further, we do not want independence across t , rather, for two times t_1 and t_2 , we want similarity to increase as $|t_1 - t_2|$ decreases. To achieve this, define an infinite sequence of independent Gaussian processes on T , $f_k(t)$, $k = 1, 2, \dots$, with mean 0 and some covariance function. At any time t , $f_k(t)$ for all k are iid draws from a normal distribution. By transforming these Gaussian processes through the cdf of a Gaussian density (to produce an iid uniform random variables at each t), and then through the inverse cdf of a Beta(1, α), one has an infinite collection of iid Beta(1, α) random variables at any time t . The GP construction means that each β_k^t varies smoothly with t . A similar approach can be used to construct smoothly varying θ_k^t 's that are marginally H -distributed, and together, these form a family of gradually varying RPMs G^t , with

$$G^t = \sum_{i=1}^{\infty} \pi_k^t \delta_{\theta_k^t}, \quad \text{where } \pi_k^t = \beta_k^t \prod_{j=1}^{k-1} (1 - \beta_j^t) \quad (31.49)$$

Of course, such a model comes with formidable computational challenges, however the marginal properties allows standard MCMC methods from the DP and other RPMs to be adapted to such settings.

31.4 The Indian buffet process and the beta process

The Dirichlet process is a Bayesian nonparametric model useful for clustering applications, where the number of clusters is allowed to grow with the size of the dataset. Under this generative model, each observation is assigned to a cluster through the variable z_i . Equivalently, z_i can be written as a one-hot binary vector, and the entire clustering structure can be written as a binary matrix Z consisting of N rows, each with exactly one non-zero element (Figure 31.7(a)).

Clustering models that limit each observation to a single cluster can be overly restrictive, failing to capture the complexity of the real datasets. For instance, in computer vision applications, rather than assign an image to a single cluster, it might be more appropriate to assign it a binary vector of features, indicating whether different object types are or are not present in the image. Similarly, in movie recommendation systems, rather than assign a movie to a single genre ('comedy' or 'romance' etc.), it is more realistic to assign to multiple genres ('comedy' AND 'romance'). Now, in contrast to all-or-nothing clustering (which would require a new genre 'romantic comedy'), different movies can have different but overlapping sets of features, allowing a partial sharing of statistical information.

Latent feature models generalize clustering models, allowing each observation to have multiple features. Nonparametric latent feature models allow the number of available features to be infinite

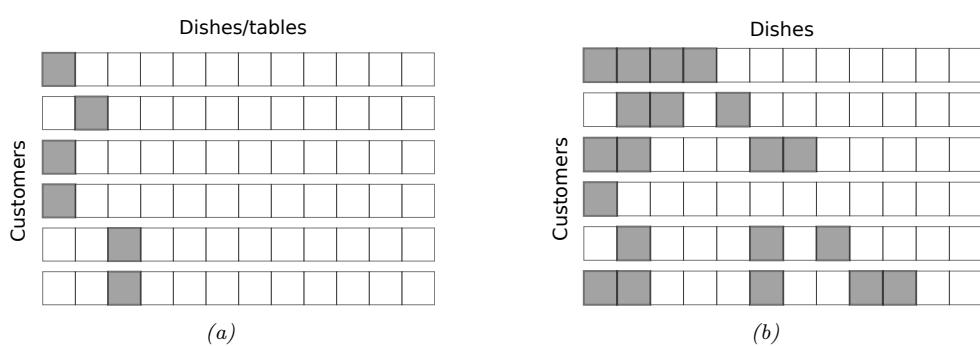


Figure 31.7: (a) A realization of the cluster matrix Z from the Chinese restaurant process (CRP) (b) A realization of the feature matrix Z from the Indian buffet process (IBP). Rows are customers and columns are dishes (for the CRP each table has its own dish). Both produce binary matrices, with the CRP assigning a customer to a single table (cluster), and the IBP assigning a set of dishes (features) to each customer.

(rather than fixed a priori to some finite number), with the number of active features in a dataset growing with a dataset size. Such models associate the dataset with an infinite binary matrix consisting of N rows, but now where each row can have multiple elements set to 1, corresponding to the active features. This is shown in Figure 31.7(b). As with the clustering models, each column is associated with a parameter drawn iid from some base measure H .

The Indian buffet process (IBP) is a Bayesian nonparametric analogue of the CRP for latent feature models. As with the CRP, the IBP is specified by a concentration parameter α , and a base measure H on some space Θ . The former controls the distribution over the binary feature matrix, whereas feature parameters θ_k are drawn iid from the latter. Under the IBP, individuals enter sequentially into a restaurant, now picking a set of dishes (instead of a single table). The first customer samples a Poisson(α)-distributed random number of dishes, and assigns each of them values drawn from H . When the i th customer enters the restaurant, they first make a pass through all dishes already chosen. Suppose N_d of the earlier customers have chosen dish d : then customer i selects this with probability N_d/i . This results in a rich-get-richer phenomenon, where popular dishes (common features) are more likely to be selected in the future. Additionally, the i customer samples a Poisson(α/i) number of new dishes. This results in a non-zero probability of new dishes, that nonetheless decreases with i .

A key property of the Indian buffet process is that, like the CRP, it is exchangeable. In other words, its statistical properties do not change if its rows are permuted. For instance, one can show that the number of dishes picked by any customer is marginally Poisson(α) distributed. Similarly, the distribution over the number of features shared by the first two customers is the same as that for the first and last customer. We mention that like the CRP, the ordering of the dishes (or columns) is arbitrary and might appear to violate exchangeability. For instance, the first customer cannot pick the first and third dishes and not the second dish, while this is possible for the second customer. These artifacts disappear if we index columns by their associated parameters. Equivalently, after reordering rows, we can transform the feature matrix to be left-ordered (essentially, all new dishes selected by a customer must be adjacent, see [griffiths2011indian]), and we can view the IBP as a prior on such left-ordered matrices.

47

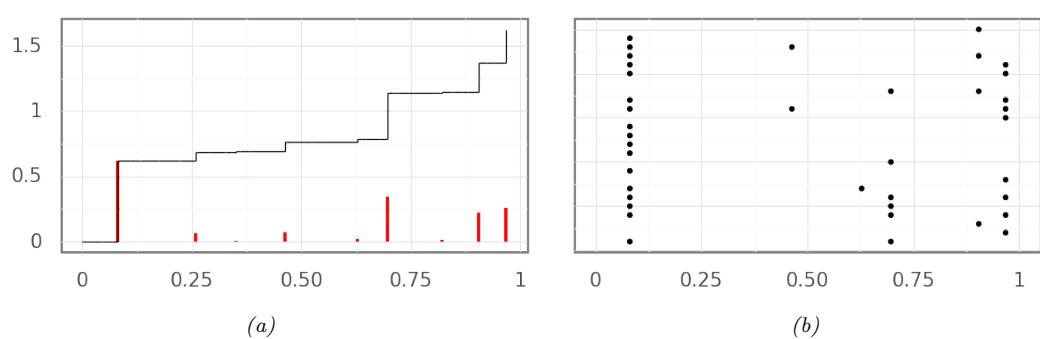


Figure 31.8: (a) A realization of a beta process on the real line. The atoms of the random measure are shown in red, while the beta subordinator (whose value at time t sums all atoms up to t) is shown in black. (b) Samples from the beta process

The exchangeability of the rows of the IBP implies via de Finetti's theorem that there exists an underlying random measure G , conditioned on which the rows are iid draws. Just as the Chinese restaurant process represents observations drawn from a Dirichlet process-distributed random probability measure, the dishes of each customer in the IBP represent observations drawn from a **Beta process**. Like the DP, the beta process is an atomic measure taking the form

$$G(\boldsymbol{\theta}) = \sum_{k=1}^{\infty} \pi_k \delta_{\boldsymbol{\theta}_k}(\boldsymbol{\theta}). \quad (31.50)$$

Each of the π_k 's lie between 0 and 1, but unlike the DP where they add up to 1, the π_k 's sum up to a finite random number. The beta process is thus a **random measure**, rather than a random probability measure. One can imagine the k 'th atom as a coin located at $\boldsymbol{\theta}_k$, with probability of success equal to π_k . To simulate a row of the IBP, one flips each of the infinite coins, selecting the feature at $\boldsymbol{\theta}_k$ if the k coin comes up heads. One can show that if the π_k 's sum up to a finite number, the number of active features will be finite. Of the infinite atoms in G , a few will dominate the rest, these will be revealed through the rich-gets-richer dynamics of the IBP as features common to a large proportion of the observations.

The beta process has a construction similar to the stick-breaking representation of the Dirichlet process. As with the DP, the locations of the atoms are independent draws from the base measure, while the sequence of weights π_1, π_2, \dots are constructed from an infinite sequence of beta variables: now these are $\text{Beta}(\alpha, 1)$ distributed, rather than $\text{Beta}(1, \alpha)$ distributed. The overall representation of the Beta process is then

$$\beta_k \sim \text{Beta}(\alpha, 1), \quad \boldsymbol{\theta}_k \sim H, \quad (31.51)$$

$$\pi_k = \beta_k \pi_{k1} = \prod_{l=1}^k \beta_l \quad (31.52)$$

It is not hard to see that under the IBP, the total number of dishes underlying a dataset of size N follows a $\text{Poisson}(\alpha H_N)$ distribution, where $H_N = \sum_{i=1}^N 1/i$ is the N th harmonic number.

1 The beta process and the IBP have been generalized to three-parameter versions allowing power-law
 2 behavior in the total number of dishes (features) in a dataset of size N , as well as in the number
 3 of customers trying each dish [[teh2009indian](#)]. It has found application in tasks from genetics,
 4 collaborative filtering, and in models for graphs. Just as with the DP, posterior inference can proceed
 5 via MCMC (exploiting either the IBP or the stick-breaking representation), particle filtering or using
 6 variational methods.
 7

8 31.5 Small-variance asymptotics

9 Nonparametric Bayesian methods can serve as a basis to develop new and efficient discrete optimization
 10 algorithms that have the flavor of Lloyd's algorithm for the k -means objective function. The starting
 11 point for this line of work is the view of the k -means algorithm as the *small-variance asymptotic*
 12 limit of the EM algorithm for a mixture of Gaussians. Specifically, consider the EM algorithm to
 13 estimate the unknown cluster means $\mu \equiv (\mu_1, \dots, \mu_k)$ of a mixture of k Gaussians, all of which have
 14 the same known covariance $\sigma^2 I$. In the limit as $\sigma^2 \rightarrow 0$, the E-step, which computes the cluster
 15 assignment probabilities of each observation given the current parameters $\mu^{(t)}$, now just assigns each
 16 observation to the nearest cluster. The M-step recomputes a new set of parameters $\mu^{(t+1)}$ given the
 17 cluster assignment probabilities, and when each observation is hard-assigned to a single cluster, the
 18 cluster means are just the means of the assigned observations. The process of repeatedly assigning
 19 observations to the nearest cluster, and recomputing cluster locations by averaging the assigned
 20 observations is exactly Lloyd's algorithm for k -means clustering.

21 To avoid having to specify the number of clusters k , [[kulis2011revisiting](#)] considered a Dirichlet
 22 process mixture of Gaussians, with all infinite components again having the same known variance σ^2 .
 23 The base measure H from which the component means are drawn was set to a zero-mean Gaussian
 24 with variance ρ^2 , with α the concentration parameter. The authors then considered a Gibbs sampler
 25 for this model, very closely related to the sampler in Algorithm 31.1, except that instead of collapsing
 26 or marginalizing out the cluster parameters, these are instantiated. Thus, an observation x_i as assigned
 27 to a cluster c with mean μ_c with probability proportional to $N_{c,-i} \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{1}{2\sigma^2} \|x_i - \mu_c\|^2)$, while
 28 it is assigned to a new cluster with probability proportional to $\alpha \frac{1}{\sqrt{2\pi(\sigma^2 + \rho^2)}} \exp(-\frac{1}{2(\sigma^2 + \rho^2)} \|x_i\|^2)$.
 29

30 After cycling through all observations, one then resamples the parameters of each cluster. Following
 31 the Gaussian base measure and Gaussian likelihood, this too follows a Gaussian distribution, whose
 32 mean is a convex combination of the prior mean 0 and data-driven term, specifically the average of
 33 the assigned observations. The weight of the prior term is proportional to the inverse of the prior
 34 variance ρ^2 , while the weight of the likelihood term is proportional to the inverse of the likelihood
 35 variance σ^2 .

36 To derive a small-variance limit of the sampler above, we cannot just let σ^2 go to 0, as that
 37 would result in each observation being assigned to its own cluster. To prevent the likelihood from
 38 dominating the prior in this way, one must also send the concentration parameter α to 0, so that a
 39 the DP prior enforces an increasingly strong penalty on a large number of clusters (recall that a
 40 priori, the average number of clusters underlying N observations is $\alpha \log N$). [[kulis2011revisiting](#)]
 41 showed that with α scaled as $\alpha = (1 + \rho^2/\sigma^2)^{d/2} \exp(-\frac{\lambda}{2\sigma^2})$ for some parameter λ , and taking the
 42 limit $\sigma^2 \rightarrow 0$ with ρ fixed, we get the following modification of the k -means algorithm:
 43

- 44 1. Assign each observation to the nearest cluster, *unless the distance to the nearest cluster exceeds λ ,*
 45 *in which case assign the observation to its own cluster.*

- 1
2 2. Set the cluster means equal to the average of the assigned observations.
3
4

5 **Algorithm 31.3:** DP-means hard clustering algorithm for data $\mathcal{D} = \{x_1, \dots, x_N\}$

6 1 Initialize the number of clusters $K = 1$, with cluster parameter μ_1 equal to the global mean:
7 $\mu_1 = \frac{1}{N} \sum_{i=1}^N x_i$
8 2 Initialize all cluster assignment indicators $z_i = 1$
9 3 **while** all z_i s have not converged **do**
10 4 **for** each $i = 1 : N$ **do**
11 5 Compute distance $d_{ik} = \|x_i - \mu_k\|^2$ to cluster k for $k = 1, \dots, K$
12 6 **if** $\min_k d_{ik} > \lambda$ **then**
13 7 Increase the number of clusters K by 1: $K = K + 1$
14 8 Assign observation i to this cluster: $z_i = K$ and $\mu_K = x_i$
15 9 **else**
16 10 Set $z_i = \arg \min_c d_{ik}$
17 11 **for** each $k = 1 : K$ **do**
18 12 Set \mathcal{D}_k be the observations assigned to cluster k . Compute cluster mean
19 20 $\mu_k = \frac{1}{|\mathcal{D}_k|} \sum_{x \in \mathcal{D}_k} x$
21

22
23 Like k -means, this is a hard-clustering algorithm, except that instead of having to specify the
24 number of clusters k , one just specifies a penalty λ for introducing a new cluster. The actual number
25 of clusters is determined by the data, [kulis2011revisiting] refer to this algorithm as DP-means.
26 One can show that the iterates above converge monotonically to a local maximum of the following
27 objective function:
28

29
30
$$\sum_{k=1}^K \sum_{x \in \mathcal{D}_k} \|x - \mu_k\|^2 + \lambda K, \quad \text{where } \mu_k = \frac{1}{|\mathcal{D}_k|} \sum_{x \in \mathcal{D}_k} x. \quad (31.53)$$

31
32

33 The first term in the expression above is exactly the objective function of the k -means algorithm with
34 K clusters. The second term is an penalty term that introduces a cost λ for each additional cluster.
35 Interestingly, the penalty term above corresponds to the so called Akaike information criterion (AIC),
36 a well studied approach to penalizing model complexity.
37

38 It is also possible to derive hard-clustering algorithms for simultaneously clustering multiple
39 datasets, while allowing these to share clusters. This is possible through the small-variance limit of a
40 Gibbs sampler for the hierarchical Dirichlet process (HDP) and results in a clustering algorithm that
41 now has two thresholding parameters, a local one λ_l and a global one λ_g . The algorithm proceeds by
42 maintaining a set of global clusters, with the local clusters of each dataset assigned to a subset of the
43 global clusters. It then repeats the following steps until convergence:
44

45 **Assign observations to local clusters** For x_{ij} , the i 'th datapoint in dataset j , compute the
46 distance to all global clusters. For those global clusters not currently present in dataset j , add λ_l
47 to their distance; this reflects the cost of introducing a new cluster into dataset j . Now, assign x_{ij}

1 to the cluster with the smallest distance, unless the smallest distance exceeds $\lambda_g + \lambda_l$, in which
2 case, create a new global cluster and assign x_{ij} to it. Observe that in the latter case, the distance
3 of x_{ij} to the new cluster is 0, with $\lambda_g + \lambda_l$ reflecting the cost of introducing a new global and
4 then local cluster.
5

6 **Assign local clusters to global clusters** For each local cluster l , compute the sum of the dis-
7 tances of all its assigned observations to the cluster mean. Call this d_l . Also compute the sum of
8 the distances of the assigned observations to each global cluster. For global cluster p , call this $d_{l,p}$.
9 Then assign local cluster l to the global cluster with the smallest $d_{l,p}$, unless $\min d_{l,p} > d_l + \lambda_g$ in
10 which case we create a new global cluster.
11

12 **Recompute global cluster means** Set the global cluster means equal to the average of the as-
13 signed observations across all datasets.
14

15 In [**jiang2012small**], these ideas were extended from DP mixtures of Gaussians to DP mixtures of
16 more general exponential family distributions. Briefly, the hard-clustering algorithms maintained the
17 same structure, the only difference being that distance from clusters was measured using a **Bregman**
18 divergence specific to that exponential family distribution. For Gaussians, the Bregman divergence
19 reduces to the usual Euclidean distance.
20

21 In [**broderick2013mad**], the authors showed how such small-variance algorithms could be derived
22 directly from a probabilistic model, and independent of any specific computational algorithm such as
23 EM or Gibbs sampling. Their approach involves computing the MAP solution of the parameters of
24 the model, and then taking the small-variance limit to obtain an objective function. This approach,
25 called MAP-based asymptotic derivations from Bayes or **MAD-Bayes**, allowed them to derive,
26 among other things, an analog of the DP-means algorithm from feature-based models. They called
27 this the **BP-means** algorithm, after the beta process underlying the Indian buffet process.
28

29 Write X for an $N \times D$ matrix of N D -dimensional observations, Z for an $N \times K$ matrix of binary
30 feature assignments, and A for a $K \times D$ matrix of K D -dimensional features, with one seeking a
31 pair (A, Z) such that ZA approximates X as well as possible. [**broderick2013mad**] showed in the
32 small-variance limit, finding the MAP solution for the IBP is equivalent to the following problem:
33

$$\operatorname{argmin}_{K, Z, A} \text{tr}[(X - ZA)^t(X - ZA)] + \lambda K, \quad (31.54)$$

34 where again λ is a parameter of the algorithm, governing how the concentration parameter scales
35 with the variance, and specifying a penalty on introducing new features. This objective function
36 is very intuitive: the first term corresponds to the approximation error from K features, while the
37 second term penalizes model complexity resulting from a large number of features K . This objective
38 function can be optimized greedily by repeating three steps for each observation i until convergence:
39

- 40 1. Given A and K , compute the optimal value of the binary feature assignment vector z_i of observation
41 i and update Z .
- 42 2. Given Z and A , introduce an additional feature vector equal to the residual for the i 'th observation,
43 namely, $x_i - z_i A$. Call A' the updated feature matrix, and Z' the updated feature assignment
44 matrix, where only observation i has been assigned this feature. If the configuration $(K+1, Z', A')$
45 results in a lower value of the objective function than (K, Z, A) , set the former as the new
46 configuration. In other words, if the benefit of introducing a new feature outweighs the penalty
47 λ , then do so.

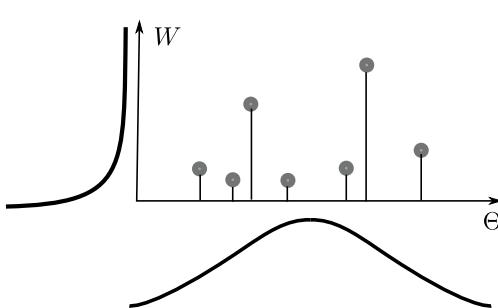


Figure 31.9: Poisson process construction of a completely random measure $G = \sum_i w_i \delta_{\theta_i}$. The set of pairs $\{(w_1, \theta_1), (w_2, \theta_2), \dots\}$ is a realization from a Poisson process with intensity $\lambda(\theta, w) = H(\theta)\gamma(w)$.

3. Update the feature vectors A given the feature assignment vectors Z .

[broderick2013mad] showed that this algorithm monotonically decreases the objective in Equation (31.54), converging eventually to a local optimum. Subsequent works have considered the small-variance asymptotics of more structured models, such as topic models, hidden Markov models and even continuous-time stochastic process models.

31.6 Completely random measures

The Dirichlet process is a example of a random probability measure, a class of random measures which always integrate to 1. The beta process, while not an RPM, belongs to another class of random measures: **completely random measures** (CRM). A completely random measure G satisfies the following property: for any two disjoint subsets T_1 and T_2 of Θ , the values $G(T_1)$ and $G(T_2)$ are independent random variables:

$$G(T_1) \perp\!\!\!\perp G(T_2) \quad \forall T_1, T_2 \subset \Theta \text{ s.t. } T_1 \cap T_2 = \emptyset. \quad (31.55)$$

Note that the Dirichlet process is not a CRM, since for disjoint sets T and its complement $T^c = \Theta \setminus T$, we have $G(T) = 1 - G(T^c)$, which is as far from independent as can be. More generally, under the DP, for disjoint sets T_1 and T_2 , the measures $G(T_1)$ and $G(T_2)$ are negatively correlated. We will see later though that the Dirichlet process is closely related to another CRM, the gamma process. As a side point, beyond the sum-to-one constraint, $G(T_1)$ does not tell us anything about the distribution of probability within $G(T_1^c)$, making the DP what is known as a **neutral process**.

The simplest example of a CRM is the **Poisson process** (see Section 31.8). A Poisson process with intensity $\lambda(\theta)$ is a **point process** producing points or events in Θ , with the number of points in any set T following a $\text{Poisson}(\int_T \lambda(\theta)d\theta)$ -distribution, and with the counts in any two disjoint sets independent of each other. While it is common to think of this as a point process, one can also think of a realization from a Poisson process as an **integer-valued random measure**, where the measure of any set is the number of points falling within that set. It is clear then that the Poisson process is an example of a CRM.

It turns out that the Poisson process underlies all completely random measures in a fundamental way. For some space Θ , and \mathbb{W} the positive real line, simulate a Poisson process on the product space

$W \times \Theta$ with intensity $\lambda(\theta, w)$. Figure 31.9 shows a realization from this Poisson process, write it as $M = \{(\theta_1, w_1), \dots, (\theta_{|M|}, w_{|M|})\}$ where $|M|$ is the number of events. This can be used to construct an atomic measure $G = \sum_{i=1}^{|M|} w_i \delta_{\theta_i}$ as illustrated in Figure 31.9. From its Poisson construction, this is a completely random measure on Θ , with set $T \in \Theta$ having measure $\sum_i w_i \mathbb{I}(\theta_i \in T)$. Different settings of $\lambda(\theta, w)$ give rise to different CRMs, and in fact, other than CRMs with atoms at some fixed locations in Θ , this construction characterizes all CRMs.

For a CRM, the Poisson intensity is typically chosen to factor as $\lambda(\theta, w) = H(\theta)\gamma(w)$, with $\int_{\Theta} H(\theta)d\theta = 1$. Then, $H(\theta)$ is the base measure controlling the locations of the atoms in the CRM, while the measure $\gamma(w)$ controls the number of atoms, and the distribution of their weights. Setting $\gamma(w) = w^{-1}(1-w)^{\alpha-1}$ gives the beta process with base measure $H(\theta)$. Other choices include the gamma process ($\gamma(w) = \alpha w^{-1} \exp(-w)$), the stable process ($\gamma(w) = \frac{1}{\Gamma(1+\sigma)} \alpha w^{-1-\sigma}$), and the generalized gamma process ($\gamma(w) = \frac{1}{\Gamma(1+\sigma)} \alpha w^{-1-\sigma} \exp(-\zeta w)$).

For all three processes described earlier, the $\gamma(w)$ integrates to infinity, so that $\int_{\Theta} \int_W \lambda(\theta, w)d\theta dw = \infty$. Consequently, the number of Poisson events, and thus the number of atoms in the CRMs are infinite with probability one. At the same time, mass of the $\gamma(w)$ function is mostly concentrated around 0 (see Figure 31.9), and for any $\epsilon > 0$, $\int_{\epsilon}^{\infty} \gamma(w)dw$ is finite. It is easy to show that the sum of the w 's is finite almost surely. Call this sum W , then the first condition ensures W greater than 0, while the second ensures it is finite. These two conditions make it sensible to divide a realization of a CRM by its sum, resulting in a random measure that integrates to 1: a random probability measure. Such RPMs are called **normalized completely random measures**, or sometimes just **normalized random measures (NRMs)**. The Dirichlet process we saw earlier is an example of an NRM: it is a normalized gamma process. This result mirrors the situation with the finite Dirichlet distribution: one can simulate from a d -dimensional $\text{Dir}(\alpha_1, \dots, \alpha_d)$ distribution by first simulating d independent gamma variables $g_i \sim \text{Ga}(\alpha_i, 1), i = 1, \dots, d$, and then defining the probability vector $\frac{1}{\sum_{i=1}^d g_i} (g_1, \dots, g_d)$. The Pitman-Yor process is not an NRM except for special settings of its parameters: like we saw, $d = 0$ is a Dirichlet process or normalized gamma process. $\alpha = 0$ is a **normalized stable process**. The normalized generalized gamma process is an NRM that includes the DP and the normalized stable process as special cases.

31

31.7 Lévy processes

Completely random measures are also closely related to **Lévy processes** and **Lévy subordinators** [bertoin1996levy]. A Lévy process is a continuous-time stochastic process $\{L_t\}_{t \geq 0}$ taking values in some space (e.g. \mathbb{R}^d) that satisfies two properties¹:

stationary increments: for $t, \Delta > 0$, the random variable $L_{t+\Delta} - L_t$ does not depend on t

independent increments: for $t, \Delta > 0$, $L_{t+\Delta} - L_t$ is independent of values before t .

A Lévy subordinator is a real-valued, **nondecreasing** Lévy process. If we drop the stationarity condition, it should be clear that the increments of a Lévy subordinator are exactly the atoms of a completely random measure (see Figure 31.8). Common examples of Lévy subordinators are beta subordinators (or beta processes), gamma subordinators, stable subordinators, and generalized

¹ There is also a technical continuity condition that we do not discuss here.

47

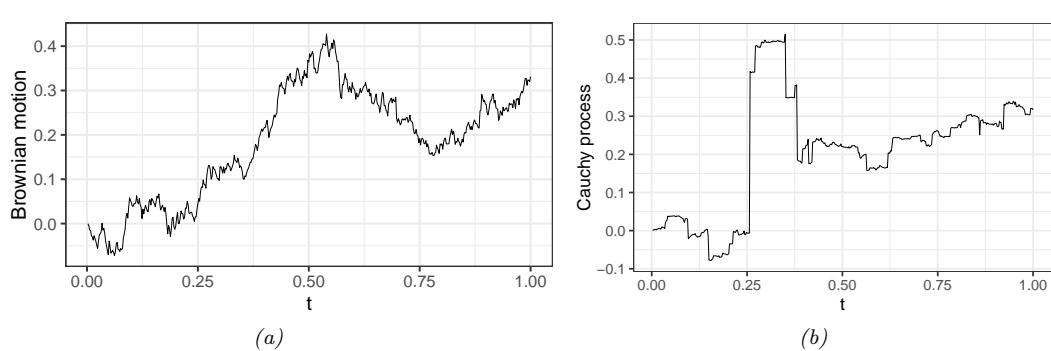


Figure 31.10: (a) A realization of a Brownian motion. (b) A realization of a Cauchy process (an α -stable process with $\alpha = 1$).

gamma subordinators. CRMs generalize Lévy subordinators, by allowing them to be indexed by some general space Θ (rather than by nonnegative reals), and by relaxing the stationarity condition to allow the atoms to follow some general base measure H .

Unlike Lévy subordinators, a general Lévy process can have negative changes as well, with the change $L_{t+\Delta} - L_t$ belonging to an **infinitely divisible distribution**, scaled by Δ . A random variable X follows an infinitely divisible distribution if, for any positive integer N , there exists some probability distribution such that the sum of N iid samples from that distribution has the same distribution as X . Examples of infinitely divisible distributions include the Poisson distribution, the gamma distribution, the Cauchy distribution, the α -stable distribution, and the inverse-Gaussian distribution (among many others), though by far the most well-known example is the Gaussian distribution. It is easy to see why the change in a Lévy process $L_{t+\Delta} - L_t$ over some time interval Δ follows an infinitely divisible distribution: just divide the interval into N equal-length segments. From the properties of the Lévy process, the changes over these segments are independent and identically distributed, and their sum equals $L_{t+\Delta} - L_t$. The **Lévy-Khintchine** formula shows that the converse is also true: any infinitely divisible distribution represents the change of an associated Lévy process. The Lévy process corresponding to the Gaussian distribution is the celebrated **Brownian motion** (or **Wiener process**). Brownian motion is a fundamental and widely applied stochastic process, whose mathematics was first studied by Louis Bachelier to model stock markets, and later, famously, by Albert Einstein to argue about the existence of atoms. For a Brownian motion, the increment $L_{t_1+\Delta} - L_{t_1}$ follows a normal $N(\mu\Delta, \sigma\Delta)$ distribution, where μ and σ are the *drift* and *diffusion* coefficients. Setting μ and σ to 0 and 1 gives **standard Brownian motion**. Paths sampled from the Weiner process are continuous with probability one, all other Lévy processes are jump processes. Figure 31.10 shows realizations from some processes. The jump processes have a Poisson construction related to the Lévy subordinators and completely random measures, and the **Lévy-Itô decomposition** shows that every Lévy process can be decomposed into Brownian and Poisson components. Levy processes have been widely applied in mathematical finance to model asset prices, insurance claims, stock prices, and other financial assets.

1 **31.8 Point processes with repulsion and reinforcement**

2
3 In this section, we look more closely at the Poisson process, as well as other, more general point
4 processes that allow inter-event interactions.
5

6
7 **31.8.1 Poisson process**

8 We have already briefly seen the Poisson process: this is a point process on some space Θ that is
9 parameterized by an intensity function $\lambda(\theta) \geq 0$, and that produces a $\text{Poisson}(\int_T \lambda(\theta)d\theta)$ -distributed
10 number of points of events in any set $T \in \Theta$, with the counts in any two disjoint sets independent
11 of each other. Recall that if $N_i \sim \text{Poisson}(\lambda_i)$ are independent, then a well-known property of the
12 Poisson distribution is that $N_1 + N_2 \sim \text{Poisson}(\lambda_1 + \lambda_2)$. This relates to the infinite divisibility
13 of the Poisson distribution. It is clear that the average number of points is large in areas where
14 $\lambda(\theta)$ is high, and small where $\lambda(\theta)$ is small. When the intensity $\lambda(\theta)$ is some constant λ , we have a
15 **homogeneous Poisson process**, otherwise we have an **inhomogeneous Poisson process**.
16

17 Depending on whether $\int_\Theta \lambda(\theta)d\theta$ is infinite or finite, a Poisson process will either produce an
18 infinite number of points (recall the Poisson process underlying the CRMs of Section 31.6) or a
19 finite number of points. The latter is more common in applications, such as modeling phone calls
20 or financial shocks (Θ is some finite time-interval), the locations of trees or forest fires (Θ is some
21 subset of the Euclidean plane), the locations of cells or galaxies (Θ is a 3-dimensional space) or
22 events in higher-dimensional spaces (for example, spatio-temporal activity). One way to simulate a
23 finite Poisson process is to first simulate the number of total number of points N , which follows a
24 $\text{Poisson}(\int_\Theta \lambda(\theta)d\theta)$ distribution. One can then simulate the locations of these points by sampling N
25 times from the probability density $\frac{\lambda(\theta)}{\int_\Theta \lambda(\theta)d\theta}$. For a homogeneous Poisson process, the locations are
26 uniformly distributed over Θ . One can also easily simulate a rate- λ homogeneous Poisson on the real
27 line by exploiting the fact that inter-event times follow an exponential distribution with mean $1/\lambda$.
28

29 If the integral $\int_\Theta \lambda(\theta)d\theta$ is difficult to evaluate, one can also simulate a Poisson process using
30 the **thinning theorem** [lewis1979simulation]. Here, one needs to find a function $\gamma(\theta)$ such that
31 $\gamma(\theta) \geq \lambda(\theta), \forall \theta$, and such that it is easy to simulate from a rate- $\gamma(\theta)$ Poisson process. Suppose the
32 result is $\Psi = \{\psi_1, \dots, \psi_N\}$. Since $\gamma(\theta) \geq \lambda(\theta)$, Ψ is going to contain more events, and one *thins* Ψ
33 by keeping each element ψ_i in Ψ with probability $\lambda(\psi_i)/\gamma(\psi_i)$ (otherwise one discards it). Once can
34 show that the set of surviving points is then a realization of a Poisson process with rate $\lambda(\theta)$.

35 The defining feature of the Poisson process is the assumption of independence among events. In
36 many settings, this is inappropriate and unrealistic, and the knowledge of an event at some location
37 θ might suggest a reduced or elevated probability of events in neighboring areas. Point processes
38 satisfying the former property are called underdispersed (Figure 31.11(b)), while point process
39 satisfying the latter property are called overdispersed (Figure 31.11(c)). Examples of underdispersed
40 point processes include the locations of trees or train stations, which tend to be more spread out than
41 Poisson because of limited resources. An example of an overdispersed point process is earthquake
42 locations, where aftershocks tend to occur in the vicinity of the main shock.

43 A simple approach to modeling overdispersed point processes is through hierarchical extensions of
44 the Poisson process that allow the Poisson intensity function $\lambda(\cdot)$ to be a random variable. Such
45 models are called doubly-stochastic Poisson processes or Cox processes [cox1980point], and a
46 common approach is to model the intensity $\lambda(\cdot)$ via a Gaussian process. Note though that the Poisson
47

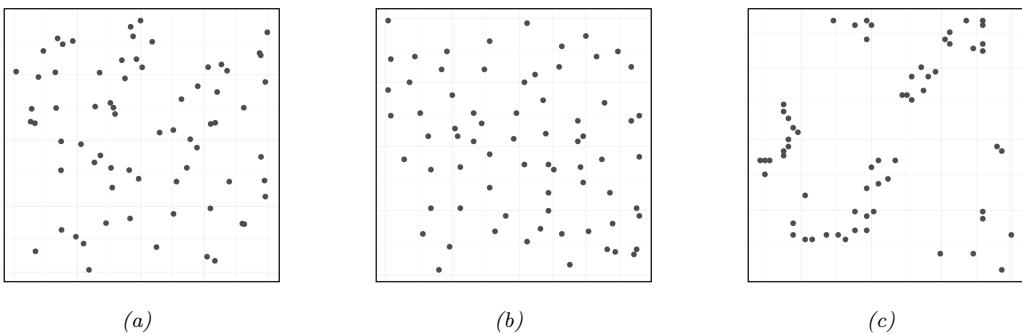


Figure 31.11: Realizations of (a) a homogeneous Poisson process; (b) an underdispersed point process (Swedish pine sapling locations [`ripley2005spatial`]); (c) an overdispersed point process (California redwood tree locations [`ripley1977modelling`]).

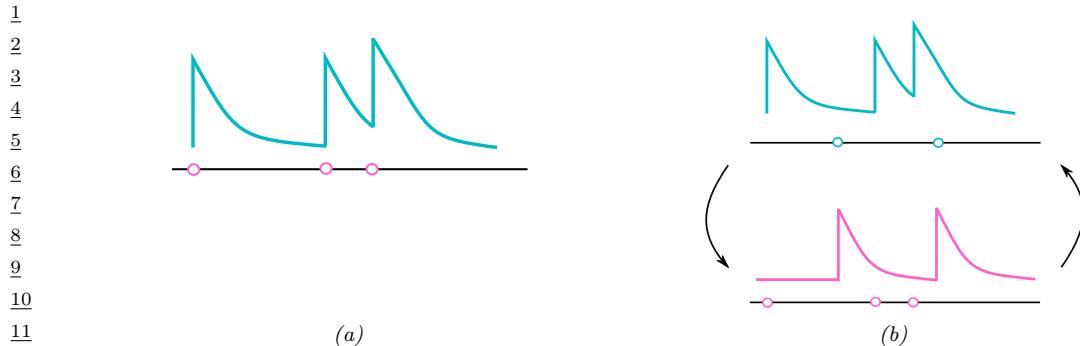
process intensity function must be nonnegative, so that λ is often a transformed Gaussian process:

$$\lambda(\theta) = g(\ell(\theta)), \quad \ell(\theta) \sim \text{GP}. \quad (31.56)$$

Common examples for g include exponentiation, sigmoid transformation or just thresholding. Because of the smoothness of the unknown $\lambda(\cdot)$, observing an event at some location suggests events are likely in the neighborhood. Such models still do not capture direct interactions between point process events, rather, these are mediated through the unknown intensity functions, making them inappropriate for many applications. For instance, in neuroscience a neuron's spiking can be driven directly by past activity, and activity of other neurons in the network, rather than just through some shared stimulus $\lambda(t)$. Similarly, social media activity has a strong reciprocal component, where, for instance, emails sent out by a user might be in response to past activity, or activity of other users. The next few subsections show how one might explicitly model such interactions.

31.8.2 Renewal process

Renewal processes are one class of models of repulsion and reinforcement for point processes defined on the real line (typically regarded as time). Recall that for a homogeneous Poisson process, inter-event times follow an exponential distribution. The exponential distribution has the property of **memorylessness**, where the time until the next event is independent of how far in the past the last event occurred. That is, if τ follows the exponential distribution, then for any δ and $\Delta > 0$, we have $p(\tau > \Delta + \delta | \tau \geq \delta) = p(\tau > \Delta)$. Renewal processes incorporate memory by allowing the interevent times to follow some general distribution on the positive reals. Examples include gamma renewal processes and Weibull renewal processes, where interevent times follow the gamma and Weibull distribution respectively. Both these processes include parameter settings that recover the Poisson process, but also allow **burstiness** and **refractoriness**. Burstiness refers to the phenomenon where, after an event has just occurred, one is more likely to see more events than a longer time afterwards. This is useful for modeling email activity for instance. Refractoriness refers to the opposite situation, where an event occurring implies new events temporarily less likely to occur. This is useful to model



13 *Figure 31.12: (a) A self-exciting Hawkes process. Each event causes a rise in the event rate, which can lead*
14 *to burstiness. (b) A pair of mutually exciting Hawkes processes, events from each cause a rise in the event*
15 *rate of the other.*

19 neural spiking activity, for instance, where after spiking, a neuron is depleted of resources, and
20 requires a recovery period before it can fire again.

23 31.8.3 Hawkes process

24 **Hawkes processes** [hawkes1971point] are another class of reinforcing point processes that have
25 attracted much recent attention. Hawkes processes provide an intuitive framework for modeling
26 reinforcement in point processes, through self-excitation when a single point process is involved, and
27 through mutual excitation (sometimes called reciprocity) when collections of point processes are
28 under study. The former, shown in Figure 31.12(a), is relevant when modeling bursty phenomena
29 like visits to a hospital by an individual, or purchases and sales of a particular stock. The latter,
30 displayed in Figure 31.12(b) is useful to characterize activity on social or biological networks, such as
31 email communications or neuronal spiking activity. In both examples, each event serves as a trigger
32 for subsequent bursts of activity. This is achieved by letting $\lambda(t)$, the event rate at time t , be a
33 function of past activity or *event history*. Write $\mathcal{H}(t) = \{t_k : t_k \leq t\}$ for the set of event times up to
34 time t . Then the rate at time t , called the *conditional intensity function* at that time, is given by

$$\lambda(t|\mathcal{H}(t)) = \gamma + \sum_{t_k \in \mathcal{H}(t)} \phi(t - t_k), \quad (31.57)$$

39 where γ is called the **base-rate** and the function $\phi(\cdot)$ is called the **triggering kernel**. The latter
40 characterizes the excitatory effect of a past event on the current event rate. Figure 31.12 shows
41 the common situation where $\phi(\cdot)$ is an exponential kernel $\phi(\Delta) = \beta e^{-\Delta/\tau}$, $\Delta \geq 0$. Here, a new
42 event causes a jump of magnitude β in the intensity $\lambda(t)$, with its excitatory influence decaying
43 exponentially back to γ , with τ the time-scale of the decay. For a multivariate Hawkes process, there
44 are m point processes $(N_1(t), N_2(t), \dots, N_m(t))$ associated with m users or nodes. Write $\mathcal{H}_i(t)$ for
45 the event history of the i th process at time t . Its conditional intensity can depend on all m event
46

histories, taking the form

$$\lambda_i(t|\{\mathcal{H}_j(t)\}_{j=1}^m) = \gamma_i + \sum_{j=1}^m \sum_{t_k \in H_j(t)} \phi_{ij}(t - t_k). \quad (31.58)$$

More typically, the conditional intensity of user i can depend only on the event histories of those nodes ‘connected’ to it, with connectivity specified by a graph structure, and with $\phi_{ij}(\cdot) = 0$ if there is no edge linking i and j . Alternately, events can have marks indicating whom they are sent to (e.g., recipients of an email), with each event only updating the conditional intensities of its recipients.

Simulating from a Hawkes process is a fairly straightforward extension of simulating from an inhomogeneous Poisson process. Consider the univariate Hawkes process, and suppose the last event occurred at time t_i . Then, until the next event occurs, at any time $t > t_i$, we have that $\mathcal{H}(t) = \mathcal{H}(t_i)$, so that the conditional intensity $\lambda(t|\mathcal{H}(t)) = \lambda(t|\mathcal{H}(t_i))$. The next event time, t_{i+1} , is then just the time of the first event of a Poisson process with intensity $\lambda(t|\mathcal{H}(t_i))$ on the interval $[t_i, \infty)$. With most choices of the kernel ϕ , t_{i+1} can easily be simulated, either by integrating $\lambda(t|\mathcal{H}(t_i))$, or by Poisson thinning. At time t_{i+1} , the history is updated to incorporate the new event, and the conditional intensity experiences a jump. The updated intensity is used to simulate the next event at some time $t_{i+2} > t_{i+1}$ from a rate- $\lambda(t|\mathcal{H}(t_{i+1}))$ Poisson process, and the process is repeated until the end of the observation interval. For the case of multivariate Hawkes processes, one has a collection of competing intensities $\lambda_i(t|\{\mathcal{H}_j(t)\}_{j=1}^m)$. The next event is the first event among all events produced by these intensities, after which the intensities are updated and the process is repeated. A realization Ψ from a Hawkes process has log-likelihood

$$\ell(\Psi) = \sum_{t^* \in \Psi} \log \lambda(t^*|\mathcal{H}(t^*)) - \int \lambda(t|\mathcal{H}(t)) dt. \quad (31.59)$$

This can typically be evaluated quite easily, so that maximum likelihood estimates of parameters like the base-rate as well as parameters of the excitation kernel can be obtained straightforwardly.

The Hawkes process as described is a fairly simple model, and there have been a number of extensions enriching its structure. An early example is [[blundell2012modelling](#)], where the authors considered the multivariate Hawkes process, now with an underlying clustering structure. Instead of each individual point process having its own conditional intensity function, each cluster has an intensity function shared by all point process assigned to it. The interaction kernels are also defined at the cluster level, with an event in process i causing a jump $\phi_{c_ic}(\tau)$ in the intensity function of cluster c (where c_i is the cluster point process i belongs to). The cluster structure was modeled through a Dirichlet process, allowing the authors to learn the underlying clustering, as well as the inter-cluster interaction kernels from interaction data. In [[tan2016content](#)], the authors considered **marked** point processes, where event i at time t_i has some associated content y_i (for example, each event is a social media post, and y_i is the text associated with the post at time t_i). The authors allowed the jump in the conditional intensity to depend on the associated mark, with the Hawkes kernel taking the form $\phi(\Delta) = f(y_i) \exp(-\Delta/\tau)$. In that work, the authors modeled the function f with a Gaussian process, though other approaches, such as ones based on neural networks, are possible.

The neural Hawkes process [[mei2017neural](#)] is a more fleshed out approach to modeling point processes using neural networks. This models event intensities through the state of a continuous-time

1 LSTM, a modification the more standard discrete-time LSTMs from ???. Central to an LSTM is a
2 memory cell \mathbf{c}_i to store long-term memory, summarizing the past until time step i . Continuous-time
3 LSTMs include two long-term memory cells, \mathbf{c}_i and $\tilde{\mathbf{c}}_i$, summarizing the history until the i th Hawkes
4 event. The first cell represents the starting value to which the intensity jumps after the i 'th event, and
5 the second represents a baseline rate after the i 'th event. These are both updated after each event,
6 with $\mathbf{c}(t)$, the instantaneous rate at any intermediate time determined by \mathbf{c}_i decaying exponentially
7 towards the baseline $\tilde{\mathbf{c}}_i$. This mechanism allows the intensity at any time to be influenced not just
8 by the number of events in the past, but also the waiting times between them. It can be extended
9 to marked point processes, where each event is also associated with a mark \mathbf{y} : now both a learned
10 embedding of the mark as well as the time since the last event is used to update state and long-term
11 memory. For more details, see also **du2016recurrent**.

13

14 31.8.4 Gibbs point process

15 **Gibbs point processes** [moller2007modern] from the statistical physics and spatial statistics
16 literature provide a general framework for modeling interacting point processes on higher-dimensional
17 spaces. Such spaces are more challenging than the real line, since there is now no ordering of points,
18 and thus no natural notion of history affecting future activity. Instead, Gibbs point processes use an
19 **energy function** E to quantify deviations from a Poisson process with rate 1. Specifically, under a
20 Gibbs process, the probability density of any configuration Ψ with respect to a rate-1 Poisson process
21 takes the form

$$\frac{23}{24} P_\beta(\Psi) = \frac{1}{Z_\beta} \exp(-\beta E(\Psi)) \quad (31.60) \quad \frac{25}{25}$$

26 where β is the inverse-temperature parameter, and $Z_\beta = \mathbb{E}_\pi[\exp(-\beta E(\Psi))]$ is the normalization
27 constant (the expectation is with respect to π , the unit-rate Poisson process). Under some conditions
28 on the energy function E , the expectation Z_β is finite, and P_β is a well-defined density, whose integral
29 with respect to π is 1. While the above equation resembles a Markov random field, the domain of Ψ
30 is much more complicated, and evaluating Z_β now involves solving an infinite dimensional integral.
31 Equation (31.60) states that configurations Ψ for which $E(\Psi)$ is small are more likely than under
32 a Poisson process, with β controlling how peaked this is. The most common energy functions are
33 **pairwise potentials**, taking the form

$$\frac{34}{35} E(\Psi) = \sum_{(s,s') \in \Psi} \phi(\|s - s'\|), \quad (31.61) \quad \frac{36}{37}$$

38 where the summation is over all pairs of events in Ψ , and $\phi : \mathbb{R}^+ \rightarrow \mathbb{R} \cup \infty$. The Strauss process is a
39 specific example, with energy function specified by positive parameters a and R as

$$\frac{40}{41} E(\Psi) = \sum_{s,s' \in \Psi} a \cdot \mathbb{I}(\|s - s'\| \leq R). \quad (31.62) \quad \frac{42}{43}$$

44 This is a repulsive process that penalizes configurations with events separated by distance less than
45 R , and as $a \rightarrow \infty$ becomes a **hardcore repulsive process**, forbidding configurations with two
46 points separated by less than R . More generally, the energy function can be piecewise-constant,
47

parameterized by a collection of pairs $(a_1, R_1), \dots, (a_n, R_n)$:

$$E(\Psi) = \sum_{s, s' \in \Psi} \sum_{i=1}^n a_i \cdot \mathbb{I}(\|s - s'\| \leq R_i). \quad (31.63)$$

Another natural option is to use smooth functions like a squared exponential kernel. Gibbs point processes can also involve higher-order interactions, examples being Geyer's triplet point process (which penalizes occurrences of 3 events that are all within some distance R), or area-interaction point processes (that center disks of radius R on each event, calculate the area of the union of these disks, and define E as some function of this area).

While Gibbs point processes are flexible and interpretable point process models, the intractable normalization constant Z_β makes estimating parameters like β a formidable challenge. In practice, these models have to be fit using approximate approaches such as maximizing a pseudo-likelihood function (instead of Equation (31.60)).

31.8.5 Determinantal point process

Determinantal point processes or **DPPs** are another approach to modeling repulsion, and have seen considerable popularity in the machine learning literature (see e.g., ([kulesza2012determinantal; macchi1975coincidence; borodin2009determinantal; lavancier2015determinantal])). Like any point process, a DPP is a probability distribution over subsets of a fixed set \mathcal{S} , and in the DPP literature this is often called the **ground set**. Point process applications typically have \mathcal{S} with uncountably infinite cardinality (for example, \mathcal{S} could be the real line), although machine learning applications of DPPs often focus on \mathcal{S} with a finite number of elements. For instance, \mathcal{S} could be a database of images, a collection of news articles, or a group of individuals. A sample from a DPP is a random subset of \mathcal{S} , produced, for instance, in response to a search query. The repulsive nature of DPPs ensures diversity and parsimony in the returned subset. This could be useful, for example, to ensure responses to a search query are not minor variations of the same image. Another application is clustering, where a DPP serves as a prior over the number of clusters and their locations, with the repulsiveness discouraging redundant clusters that are very similar to each other.

DPPs are parameterized by a similarity kernel K , whose element K_{ij} gives the similarity between elements i and j of the ground set. For finite ground sets, K is just a similarity matrix. DPPs require K to be positive definite, with largest eigenvalue less than 1, that is $0 \preceq K \preceq \mathbf{I}$. For simplicity, K is often assumed symmetric, though this is not necessary. Given a kernel K , the associated DPP is defined as follows: if Y is the random subset of \mathcal{S} drawn from the DPP, then the probability that Y contains any subset A of \mathcal{S} is given by

$$p(A \subseteq Y) = \det(K_A). \quad (31.64)$$

Here K_A is the submatrix obtained by restricting K to rows and columns in A , and we define $\det(K_\phi) = 1$ for the empty set ϕ . Observe that this probability is specified exactly, and not just up to a normalization constant. We immediately see that Y contains the empty set with probability one (this is trivially true since the empty set is a subset of every set). We also see that the probability that element i is selected in Y is the i th diagonal element of K :

$$p(i \in Y) = \det(K_{ii}) = K_{ii}, \quad (31.65)$$

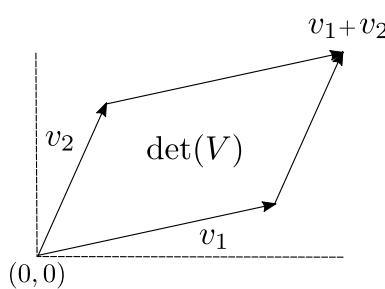


Figure 31.13: The determinant of a matrix V is the volume of the parallelogram spanned by the columns of V and the origin.

so that the diagonal of K gives the inclusion probabilities of the individual elements in \mathcal{S} . More interestingly, the probability a pair of elements $\{i, j\}$ are both contained in Y is given by

$$p(\{i, j\} \subseteq Y) = K_{ii}K_{jj} - K_{ij}^2. \quad (31.66)$$

The first term $K_{ii}K_{jj}$ is the probability of including i and j if they were independent, and this probability is adjusted by subtracting K_{ij}^2 , a measure of similarity between i and j . This is the source of repulsiveness or diversity in DPPs. More generally, the determinant of a set of vectors is the volume of the parallelogram spanned by them and the origin (see Figure 31.13), and by making the probability of inclusion proportional to the volume, DPPs encourage diversity. We mention for completeness that for uncountable ground sets like a Euclidean space, the determinant $\det(K_A)$ now gives the **product density function** of a realization A . This generalizes the intensity of a Poisson process to account for interactions between point process events, and we refer the interested reader to [lavancier2015determinantal](#) for more details.

Equation (31.64) characterizes the marginal probabilities of a determinantal point process: what is the probability that a realization Y contains some subset of \mathcal{S} . More often, one is interested in the probability that the realization Y equals some subset of \mathcal{S} . The latter is of interest for simulation, inference and parameter learning with DPPs. For this, it is typical to work with a narrower class of DPPs called **L-ensembles** [[borodin2009determinantal](#); [kulesza2012determinantal](#)]. Like general DPPs, such a process is characterized by a positive semidefinite matrix L , with the probability that Y equals some configuration A given by:

$$p(Y = A) \propto \det(L_A). \quad (31.67)$$

Note that this probability is specified only upto a normalization constant, and so we do not need to upperbound eigenvalues of L . In fact, it is not hard to show that the normalizer is given by $(I + \det(L))$, so that

$$p(Y = A) = \frac{\det(L_A)}{I + \det(L)}. \quad (31.68)$$

A similar calculation can be used to show that $p(A \subseteq Y) = \det(K)$, where $K = L(I + L)^{-1} = I - (I + L)^{-1}$, showing that L-ensembles are indeed a special kind of DPP. Equation (31.68) and

Equation (31.64) allow parameters of the DPP to be estimated from realizations of a point process, typically by gradient descent. Without additional structure, naively calculating determinants is cubic in the cardinality of \mathcal{S} , and this represents a substantial saving when one considers the number of possible subsets of \mathcal{S} . When even cubic scaling is too expensive, a number of approximation approaches can be adopted, and these are often closely related to approaches to solve the cubic cost of Gaussian processes.

So far, we have only discussed how to calculate the probability of samples from a DPP. Simulating from a DPP, while straightforward, is a bit less intuitive, and we refer the reader to [lavancier2015determinantal; kulesza2012determinantal]. At a high level, these approaches use the eigenstructure of K to express a DPP as a mixture of **determinantal projection point processes**. The latter are DPPs whose similarity kernel has binary eigenvalues (either 0 or 1) and are easier to sample from. Observe that any eigenvalue λ_i of the similarity kernel K must lie in the interval $[0, 1]$. This allows us to generate a random similarity kernel \hat{K} with the same eigenvectors as K but with binary eigenvalues as follows: for each i , replace eigenvalue λ_i of K with a binary variable $\hat{\lambda}_i$ simulated from a $\text{Bernoulli}(\lambda_i)$ distribution. One can show that the DPP simulated from \hat{K} after simulating \hat{K} from K in this fashion is distributed exactly as a DPP with kernel K . We refer the reader to [lavancier2015determinantal] for details on how to simulate a determinantal projection point process with similarity kernel \hat{K} .

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

32 Representation learning

33 Interpretability

PART VI

Decision making

34 Multi-step decision problems

35 Reinforcement learning

36 Causality