Supplementary Material for "Probabilistic Machine Learning: Advanced Topics"

Kevin Murphy

April 29, 2022

Contents

1 Introduction **9**

I Fundamentals **11**

2 Probability **13**

2.1 Some common discrete distributions . . . 13
2.1.1 Bernoulli and binomial distributions . . . 13
2.1.2 Categorical and multinomial distributions . . . 13
2.1.3 Poisson distribution . . . 14
2.1.4 Negative binomial distribution . . . 14
2.1.5 Hyper-geometric distribution . . . 15
2.1.6 Polya's urn . . . 15
2.2 Some common continuous distributions . . . 15
2.2.1 Gaussian (normal) distribution . . . 15
2.2.2 Half-normal . . . 16
2.2.3 Student distribution . . . 16
2.2.4 Cauchy distribution . . . 17
2.2.5 Laplace distribution . . . 17
2.2.6 Sub-Gaussian and super-Gaussian distributions . . . 18
2.2.7 Beta distribution . . . 18
2.2.8 Gamma distribution . . . 19
2.2.9 Pareto distribution . . . 20
2.3 The multivariate Gaussian (normal) distribution . . . 22
2.3.1 Definition . . . 22
2.3.2 Typical sets and Gaussian shells . . . 23
2.3.3 Moment form and canonical form . . . 23
2.3.4 Marginals and conditionals of a MVN . . . 24
2.3.5 Bayes' rule for linear Gaussian systems . . . 26
2.3.6 Sensor fusion with known measurement noise . . . 28
2.3.7 Sensor fusion with unknown measurement noise . . . 28
2.3.8 Handling missing data . . . 31
2.4 Some other multivariate continuous distributions . . . 31
2.4.1 Multivariate Student distribution . . . 31
2.4.2 Circular normal (von Mises Fisher) distribution . . . 32
2.4.3 Matrix-variate Gaussian (MVG) distribution . . . 32
2.4.4 Wishart distribution . . . 32
2.4.5 Dirichlet distribution . . . 34
2.5 Google's PageRank algorithm . . . 36
2.5.1 Retrieving relevant pages using inverted indices . . . 36

2.5.2 The PageRank score . . . 37
2.5.3 Efficiently computing the PageRank vector . . . 38
2.5.4 Web spam . . . 39
2.5.5 Personalized PageRank . . . 39

3 Bayesian statistics **41**
3.1 Bayesian concept learning . . . 41
3.1.1 Learning a discrete concept: the number game . . . 41
3.1.2 Learning a continuous concept: the healthy levels game . . . 46
3.2 Conjugate Bayesian analysis for the multivariate Gaussian . . . 49
3.2.1 Posterior of $\boldsymbol{\mu}$ given $\boldsymbol{\Sigma}$ . . . 49
3.2.2 Posterior of $\boldsymbol{\Sigma}$ given $\boldsymbol{\mu}$ . . . 49
3.2.3 Posterior of $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$ . . . 50
3.2.4 Posterior using LKJ prior . . . 54
3.3 Informative priors . . . 54
3.3.1 Domain specific priors . . . 55
3.3.2 Gaussian prior . . . 55
3.3.3 Power-law prior . . . 55
3.3.4 Erlang prior . . . 56

4 Graphical models **59**
4.1 More examples of DGMs . . . 59
4.1.1 Water sprinkler . . . 59
4.1.2 Asia network . . . 60
4.1.3 The QMR network . . . 61
4.1.4 Genetic linkage analysis . . . 62
4.2 More examples of UGMs . . . 65
4.3 Restricted Boltzmann machines (RBMs) in more detail . . . 65
4.3.1 Binary RBMs . . . 65
4.3.2 Categorical RBMs . . . 66
4.3.3 Gaussian RBMs . . . 66
4.3.4 RBMs with Gaussian hidden units . . . 66

5 Information theory **67**

6 Optimization **69**
6.1 Proximal methods . . . 69
6.1.1 Proximal operators . . . 69
6.1.2 Computing proximal operators . . . 71
6.1.3 Proximal point methods (PPM) . . . 74
6.1.4 Proximal gradient method . . . 76
6.1.5 Alternating direction method of multipliers (ADMM) . . . 77
6.2 Mirror descent . . . 79
6.2.1 Bregman divergence . . . 79
6.2.2 Proximal point method . . . 80
6.2.3 PPM using Bregman divergence . . . 80
6.3 Dynamic programming . . . 80
6.3.1 Example: computing Fibonnaci numbers . . . 81
6.3.2 ML examples . . . 81
6.4 Conjugate duality . . . 81
6.4.1 Introduction . . . 82
6.4.2 Example: exponential function . . . 82

6.4.3 Conjugate of a conjugate . . . 83
6.4.4 Bounds for the logistic (sigmoid) function . . . 84

II Inference **87**

7 Inference algorithms: an overview **89**

8 Message passing inference **91**
8.1 Kalman filtering variants . . . 91
8.2 Inference based on local linearization . . . 92
8.2.1 Taylor series expansion . . . 92
8.2.2 The extended Kalman filter (EKF) . . . 94
8.2.3 The extended Kalman smoother . . . 97
8.2.4 Exponential-family EKF . . . 97
8.3 Inference based on the unscented transform . . . 100
8.3.1 The unscented transform . . . 100
8.3.2 The unscented Kalman filter (UKF) . . . 101
8.3.3 The unscented Kalman smoother . . . 103
8.4 Other variants of the Kalman filter . . . 103
8.4.1 Ensemble Kalman filter . . . 103
8.4.2 Robust Kalman filters . . . 105
8.4.3 Gaussian filtering . . . 105
8.5 Assumed density filtering for SLDS models . . . 106
8.6 More junction trees . . . 108
8.6.1 Tree decompositions of some common graph structures . . . 108
8.6.2 Junction tree algorithm applied to a chain . . . 108
8.6.3 JTA for temporal graphical models . . . 109
8.7 MAP estimation for discrete PGMs . . . 111
8.7.1 Notation . . . 111
8.7.2 The marginal polytope . . . 112
8.7.3 Linear programming relaxation . . . 112
8.7.4 Graphcuts . . . 115

9 Variational inference **119**
9.0.1 Exploiting partial conjugacy . . . 119
9.1 Exact and approximate inference for PGMs . . . 122
9.1.1 Exact inference as VI . . . 122
9.1.2 Mean field VI . . . 123
9.1.3 Loopy belief propagation as VI . . . 123
9.1.4 Convex belief propagation . . . 126
9.1.5 Tree-reweighted belief propagation . . . 127
9.1.6 Other tractable versions of convex BP . . . 128

10 Monte Carlo Inference **129**

11 Markov Chain Monte Carlo (MCMC) inference **131**

12 Sequential Monte Carlo (SMC) inference **133**

III Prediction 135

13 Predictive models: an overview 137

14 Generalized linear models 139
14.1 Variational inference for logistic regression . . . 139
14.1.1 Binary logistic regression . . . 139
14.1.2 Multinomial logistic regression . . . 140
14.2 Converting multinomial logistic regression to Poisson regression . . . 144
14.3 Case study: is Berkeley admissions biased against women? . . . 144
14.3.1 Binomial logistic regression . . . 144
14.3.2 Beta-binomial logistic regression . . . 146
14.3.3 Poisson regression . . . 148
14.3.4 GLMM (hierarchical Bayes) regression . . . 148

15 Deep neural networks 151

16 Bayesian neural networks 153

17 Gaussian processes 155

18 Beyond the iid assumption 157

IV Generation 159

19 Generative models: an overview 161

20 Variational autoencoders 163

21 Auto-regressive models 165

22 Normalizing flows 167

23 Energy-based models 169

24 Denoising diffusion models 171

25 Generative adversarial networks 173

V Discovery 175

26 Discovery methods: an overview 177

27 Latent factor models 179
27.1 Inference in topic models . . . 179
27.1.1 Collapsed Gibbs sampling for LDA . . . 179
27.1.2 Variational inference for LDA . . . 181

28 State-space models 185
28.1 HMMs: More applications . . . 185
28.1.1 Spelling correction . . . 185
28.1.2 Protein sequence alignment . . . 187
28.2 HMMs: parameter learning . . . 188

28.2.1 The Baum-Welch (EM) algorithm . . . 188
28.2.2 Parameter estimation using SGD . . . 191
28.2.3 Parameter estimation using spectral methods . . . 194
28.2.4 Bayesian parameter inference . . . 194
28.3 HMMs: Generalizations . . . 195
28.3.1 Hidden semi-Markov model (HSMM) . . . 195
28.3.2 HSMMs for changepoint detection . . . 197
28.3.3 Hierarchical HMMs . . . 199
28.3.4 Factorial HMMs . . . 201
28.3.5 Coupled HMMs . . . 203
28.3.6 Dynamic Bayes nets (DBN) . . . 204
28.4 Continuous time SSMs . . . 204
28.4.1 Ordinary differential equations . . . 204
28.4.2 Example: Noiseless 1d spring-mass system . . . 205
28.4.3 Example: tracking a moving object in continuous time . . . 206
28.5 LG-SSMs: parameter estimation . . . 208
28.5.1 Training with fully observed data . . . 208
28.5.2 EM for LG-SSM . . . 208
28.5.3 Subspace identification methods . . . 209
28.5.4 Ensuring stability of the dynamical system . . . 209
28.5.5 Laplace EM method . . . 209
28.6 Structured State Space Sequence model (S4) . . . 210

29 Graph learning **213**

30 Non-parametric Bayesian models **215**

31 Representation learning **217**

32 Interpretability **219**

VI Decision making **221**

33 Multi-step decision problems **223**

34 Reinforcement learning **225**

35 Causality **227**

Chapter 1

Introduction

Part I

Fundamentals

Chapter 2

Probability

2.1 Some common discrete distributions

In this section, we summarize some common discrete distributions.

2.1.1 Bernoulli and binomial distributions

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

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

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

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

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

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

2.1.2 Categorical and multinomial distributions

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

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

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

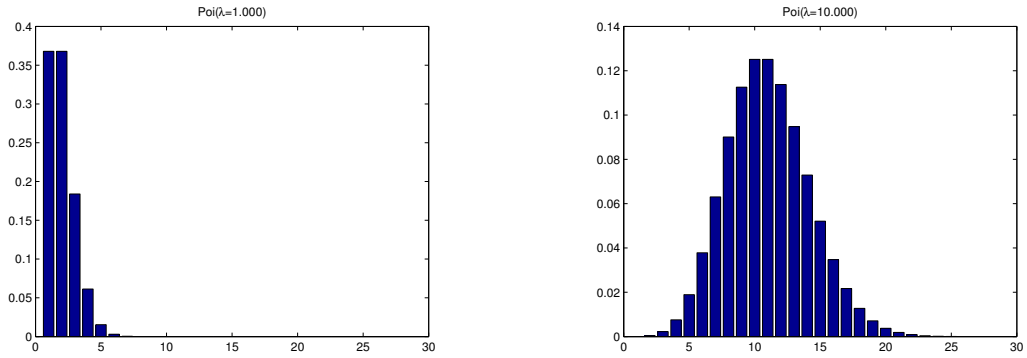
$$\text{Cat}(\boldsymbol{x}|\boldsymbol{\theta}) \triangleq \prod_{k=1}^{K} \theta_k^{x_k} \tag{2.4}$$

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

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

Name	N	Variable	Constraint
Bernoulli	1	$x \in \{0, 1\}$	
Binomial	N	$x \in \{0, 1, \ldots, N\}$	
Categorical	1	$x \in \{1, \ldots, K\}$	
Multinomial	N	$\boldsymbol{x} \in \{0, 1, \ldots, N\}^K$	$\sum_{k=1}^K x_k = N$

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



(a) *(b)*

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

where the **multinomial coefficient** is defined as

$$\binom{N}{k_1 \ldots k_m} \triangleq \frac{N!}{k_1! \ldots k_m!} \tag{2.6}$$

See Table 2.1 for a summary of the notation.[1]

2.1.3 Poisson distribution

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

$$\text{Poi}(x|\lambda) = e^{-\lambda} \frac{\lambda^x}{x!} \tag{2.7}$$

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

2.1.4 Negative binomial distribution

Suppose we have an "urn" with N balls, R of which are red and B of which are blue. Suppose we perform **sampling with replacement** until we get $n \geq 1$ balls. Let X be the number of these that are blue. It can be shown that $X \sim \text{Bin}(n, p)$, where $p = B/N$ is the fraction of blue balls; thus X follows the binomial distribution, discussed in Section 2.1.1.

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

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

$$\text{NegBinom}(x|r,p) \triangleq \binom{x+r-1}{x}(1-p)^r p^x \tag{2.8}$$

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

This distribution has the following moments:

$$\mathbb{E}[x] = \frac{p\,r}{1-p}, \; \mathbb{V}[x] = \frac{p\,r}{(1-p)^2} \tag{2.9}$$

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

2.1.5 Hyper-geometric distribution

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

$$\text{HypGeom}(x|N,n,B) \triangleq \frac{\binom{B}{x}\binom{N-B}{n-x}}{\binom{N}{n}} \tag{2.10}$$

for $x \in \{\max(0, n+B-N), \ldots, \min(n, B)\}$. This formula can be understood as follows. There are $\binom{B}{x}$ ways to choose x blue balls, and $\binom{N-B}{n-x}$ ways to choose the remaining red balls (recall that $N - B = R$). Finally, there are $\binom{N}{n}$ ways to choose n balls from the N regardless of color; this defines the normalization constant.

2.1.6 Polya's urn

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

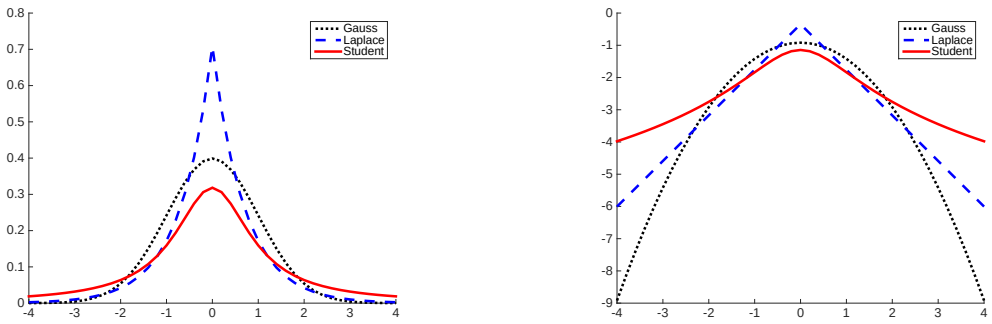
If $c = 1$, we get sampling with replacement. If $c = 0$, we get sampling without replacement. If $c > 1$, the urn will grow in size, and the colors that we sample early on will become more numerous, making them more likely to be sampled again (the opposite of sampling without replacement). This known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

2.2 Some common continuous distributions

In this section we summarize some common continuous distributions.

2.2.1 Gaussian (normal) distribution

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



(a) *(b)*

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

The pdf (probability density function) of the Gaussian is given by

$$\mathcal{N}(x|\mu,\sigma^2) \triangleq \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \tag{2.11}$$

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

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

$$\Phi(x;\mu,\sigma^2) \triangleq \int_{-\infty}^{x} \mathcal{N}(z|\mu,\sigma^2)dz \tag{2.12}$$

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

2.2.2 Half-normal

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

$$\mathcal{N}_+(y|\sigma) \triangleq 2\mathcal{N}(y|0,\sigma^2) = \frac{\sqrt{2}}{\sigma\sqrt{\pi}} \exp\left(-\frac{y^2}{2\sigma^2}\right) \quad y \geq 0 \tag{2.13}$$

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

$$\text{mean} = \frac{\sigma\sqrt{2}}{\sqrt{\pi}}, \ \text{var} = \sigma^2\left(1 - \frac{2}{\pi}\right) \tag{2.14}$$

2.2.3 Student distribution

One problem with the Gaussian distribution is that it is sensitive to outliers, since the probability decays exponentially fast with the (squared) distance from the center. A more robust distribution is the **Student**

t-**distribution**, which we shall call the **Student distribution** for short. Its pdf is as follows:

$$\mathcal{T}_\nu(x|\mu,\sigma^2) = \frac{1}{Z}\left[1+\frac{1}{\nu}\left(\frac{x-\mu}{\sigma}\right)^2\right]^{-(\frac{\nu+1}{2})} \tag{2.15}$$

$$Z = \frac{\sqrt{\nu\pi\sigma^2}\Gamma(\frac{\nu}{2})}{\Gamma(\frac{\nu+1}{2})} = \sqrt{\nu}\sigma B(\frac{1}{2},\frac{\nu}{2}) \tag{2.16}$$

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

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

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

$$\text{mean} = \mu,\ \text{mode} = \mu,\ \text{var} = \frac{\nu\sigma^2}{(\nu-2)} \tag{2.17}$$

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

2.2.4 Cauchy distribution

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

$$\mathcal{C}(x|\mu,\gamma) = \frac{1}{Z}\left[1+\left(\frac{x-\mu}{\gamma}\right)^2\right]^{-1} \tag{2.18}$$

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

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

$$\mathcal{C}_+(x|\gamma) \triangleq \frac{2}{\pi\gamma}\left[1+\left(\frac{x}{\gamma}\right)^2\right]^{-1} \tag{2.19}$$

2.2.5 Laplace distribution

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

$$\text{Lap}(x|\mu,b) \triangleq \frac{1}{2b}\exp\left(-\frac{|x-\mu|}{b}\right) \tag{2.20}$$

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

$$\text{mean} = \mu,\ \text{mode} = \mu,\ \text{var} = 2b^2 \tag{2.21}$$

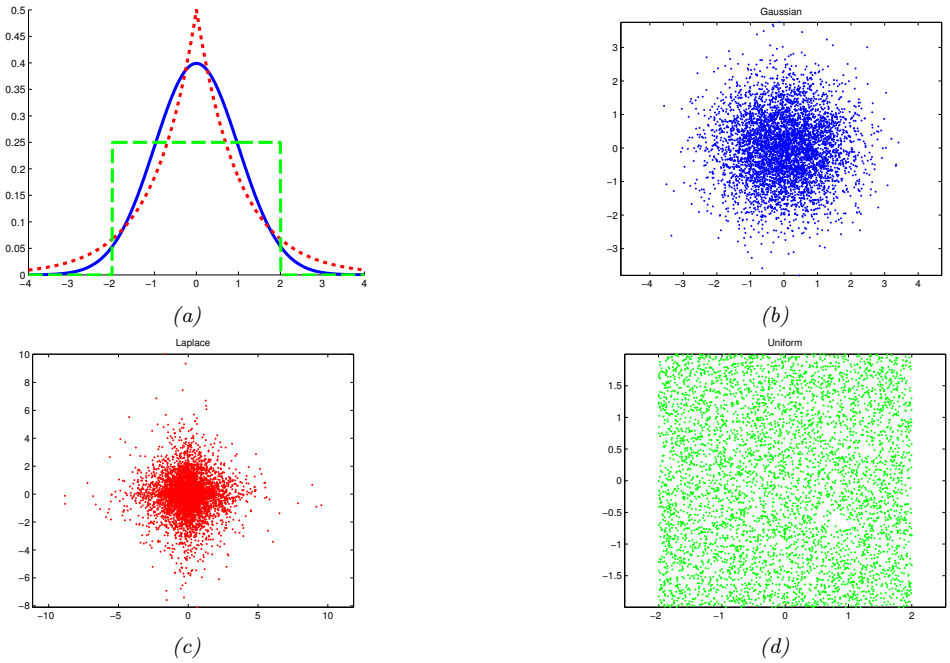


Figure 2.3: Illustration of Gaussian (blue), sub-Gaussian (uniform, green) and super-Gaussian (Laplace, red) distributions in 1d and 2d. Generated by sub_super_gauss_plot.py.

2.2.6 Sub-Gaussian and super-Gaussian distributions

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

$$\text{kurt}(z) \triangleq \frac{\mu_4}{\sigma^4} = \frac{\mathbb{E}\left[(Z-\mu)^4\right]}{(\mathbb{E}\left[(Z-\mu)^2\right])^2} \tag{2.22}$$

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

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

2.2.7 Beta distribution

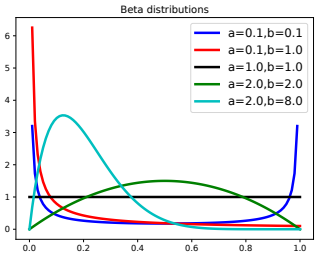
The **beta distribution** has support over the interval $[0, 1]$ and is defined as follows:

$$\text{Beta}(x|a,b) = \frac{1}{B(a,b)} x^{a-1}(1-x)^{b-1} \tag{2.23}$$

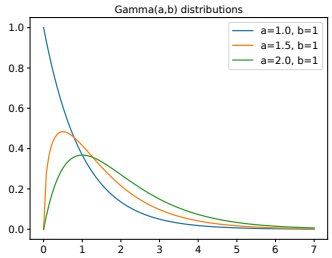
where $B(a, b)$ is the **beta function**, defined by

$$B(a,b) \triangleq \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)} \tag{2.24}$$

[2]This term is from [Kru13].



(a)



(b)

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

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

$$\Gamma(a) \triangleq \int_0^\infty x^{a-1} e^{-x} dx \tag{2.25}$$

See Figure 2.4a for plots of some beta distributions.

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

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

$$\text{mean} = \frac{a}{a+b}, \ \text{mode} \ = \frac{a-1}{a+b-2}, \ \text{var} = \frac{ab}{(a+b)^2(a+b+1)} \tag{2.26}$$

2.2.8 Gamma distribution

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

$$\text{Ga}(x|\text{shape} = a, \text{rate} = b) \triangleq \frac{b^a}{\Gamma(a)} x^{a-1} e^{-xb} \tag{2.27}$$

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

$$\text{Ga}(x|\text{shape} = a, \text{scale} = s) \triangleq \frac{1}{s^a \Gamma(a)} x^{a-1} e^{-x/s} \tag{2.28}$$

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

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

$$\text{mean} = \frac{a}{b}, \ \text{mode} = \frac{a-1}{b}, \ \text{var} = \frac{a}{b^2} \tag{2.29}$$

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

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

$$\text{Expon}(x|\lambda) \triangleq \text{Ga}(x|\text{shape} = 1, \text{rate} = \lambda) \tag{2.30}$$

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

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

$$\text{Erlang}(x|\lambda) \triangleq \text{Ga}(x|\text{shape} = 2, \text{rate} = \lambda) \tag{2.31}$$

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

$$\chi^2_\nu(x) \triangleq \text{Ga}(x|\text{shape} = \frac{\nu}{2}, \text{rate} = \frac{1}{2}) \tag{2.32}$$

where ν is called the degrees of freedom. This is the distribution of the sum of squared Gaussian random variables. More precisely, if $Z_i \sim \mathcal{N}(0,1)$, and $S = \sum_{i=1}^{\nu} Z_i^2$, then $S \sim \chi^2_\nu$. Hence if $X \sim \mathcal{N}(0, \sigma^2)$ then $X^2 \sim \sigma^2 \chi^2_1$. Since $\mathbb{E}\left[\chi^2_1\right] = 1$ and $\mathbb{V}\left[\chi^2_1\right] = 2$, we have

$$\mathbb{E}\left[X^2\right] = \sigma^2, \mathbb{V}\left[X^2\right] = 2\sigma^4 \tag{2.33}$$

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

$$\text{IG}(x|\text{shape} = a, \text{scale} = b) \triangleq \frac{b^a}{\Gamma(a)} x^{-(a+1)} e^{-b/x} \tag{2.34}$$

The distribution has these properties

$$\text{mean} = \frac{b}{a-1}, \text{mode} = \frac{b}{a+1}, \text{var} = \frac{b^2}{(a-1)^2(a-2)} \tag{2.35}$$

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

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

$$\chi^{-2}(x|\nu, \sigma^2) = \text{IG}(x|\text{shape} = \frac{\nu}{2}, \text{scale} = \frac{\nu\sigma^2}{2}) \tag{2.36}$$

$$= \frac{1}{\Gamma(\nu/2)} \left(\frac{\nu\sigma^2}{2}\right)^{\nu/2} x^{-\frac{\nu}{2}-1} \exp\left(-\frac{\nu\sigma^2}{2x}\right) \tag{2.37}$$

The distribution has these properties

$$\text{mean} = \frac{\nu\sigma^2}{\nu-2}, \text{mode} = \frac{\nu\sigma^2}{\nu+2}, \text{var} = \frac{\nu^2\sigma^4}{(\nu-2)^2(\nu-4)} \tag{2.38}$$

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

2.2.9 Pareto distribution

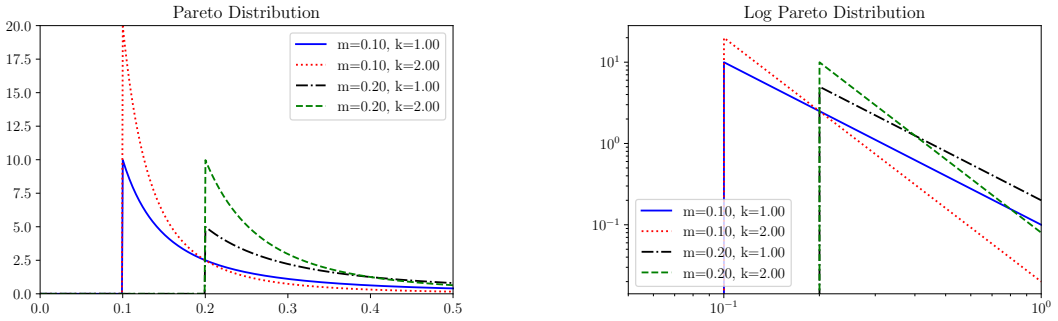
The **Pareto distribution** has the following pdf:

$$\text{Pareto}(x|m, \kappa) = \kappa m^\kappa \frac{1}{x^{(\kappa+1)}} \mathbb{I}(x \geq m) \tag{2.39}$$

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

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

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



(a) *(b)*

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

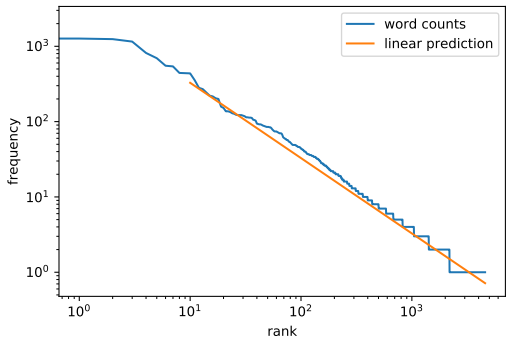


Figure 2.6: A log-log plot of the frequency vs the rank for the words in H. G. Wells' The Time Machine. *Generated by zipfs_law_plot.py. Adapted from a figure from [Zha+20, Sec 8.3].*

2.2.9.1 Modeling wealth distributions

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

2.2.9.2 Zipf's law

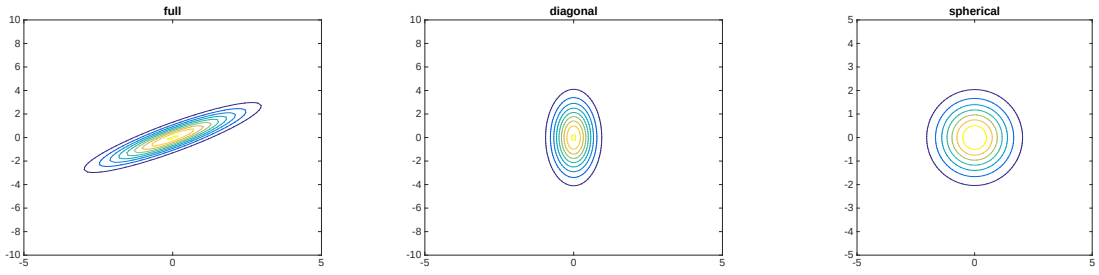
Zipf's law says that the most frequent word in a language (such as "the") occurs approximately twice as often as the second most frequent word ("of"), which occurs twice as often as the fourth most frequent word, etc. This corresponds to a Pareto distribution of the form

$$p(x = r) \propto \kappa r^{-a} \tag{2.40}$$

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

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

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



(a) *(b)* *(c)*

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

we will get a straight line with slope -1. This is in fact true, as illustrated in Figure 2.6.[5] See [Ada00] for further discussion of Zipf's law, and **??** for a discussion of language models.

2.3 The multivariate Gaussian (normal) distribution

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

2.3.1 Definition

The MVN density is defined by the following:

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) \triangleq \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right] \tag{2.41}$$

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

$$d_{\boldsymbol{\Sigma}}(\boldsymbol{x},\boldsymbol{\mu})^2 = (\boldsymbol{x}-\boldsymbol{\mu})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}) \tag{2.42}$$

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

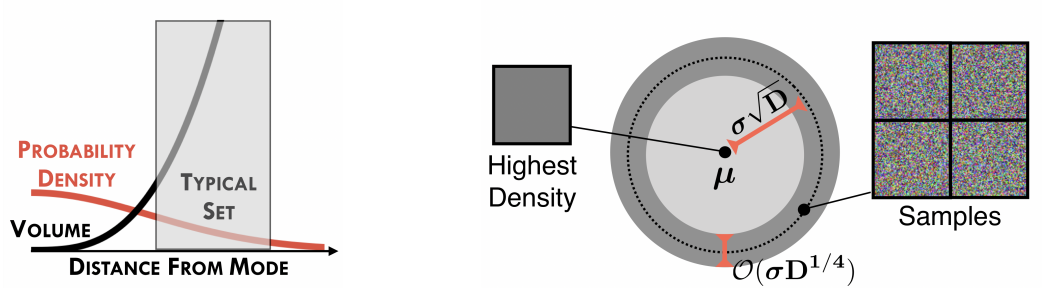
In 2d, the MVN is known as the **bivariate Gaussian** distribution. Its pdf can be represented as $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})$, where $\boldsymbol{x} \in \mathbb{R}^2$, $\boldsymbol{\mu} \in \mathbb{R}^2$ and

$$\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_1^2 & \sigma_{12}^2 \\ \sigma_{21}^2 & \sigma_2^2 \end{pmatrix} = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix} \tag{2.43}$$

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

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

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



(a) *(b)*

Figure 2.8: (a) Cartoon illustration of why the typical set of a Gaussian is not centered at the mode of the distribution. (b) Illustration of the typical set of a Gaussian, which is concentrated in a thin annulus of thickness $\sigma D^{1/4}$ and distance $\sigma D^{1/2}$ from the origin. We also show an image with the highest density (the all gray image on the left). as well as some high probability samples (the speckle noise images on the right). From Figure 1 of [Nal+19]. Used with kind permission of Eric Nalisnick.

2.3.2 Typical sets and Gaussian shells

Multivariate Gaussians can behave rather counterintuitively in high dimensions. In particular, we can ask: if we draw samples $\boldsymbol{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_D)$, where D is the number of dimensions, where do we expect most of the $\boldsymbol{x}$ to lie? Since the peak (mode) of the pdf is at the origin, it is natural to expect most samples to be near the origin. However, in high dimensions, the **typical set** of a Gaussian is a thin shell or **annulus** of radius (distance from origin) of $r = \sqrt{D}$, since although the density decays as $e^{-r^2/2}$, the volume of a sphere grows as r^D, and mass is density times volume. This is illustrated in Figure 2.8a.

In Figure 2.8b, we show some grayscale images that are sampled from a Gaussian of the form $\mathcal{N}(\boldsymbol{\mu}, \sigma^2\mathbf{I})$, where $\boldsymbol{\mu}$ corresponds to the all-gray image. However, it is extremely unlikely that randomly sampled images would be close to all-gray, as shown in the figure.

To see why the typical set for a Gaussian is concentrated in a thin annulus at radius $\sqrt{D}$, consider the squared distamce of a point $\boldsymbol{x}$ from the origin, $d(\boldsymbol{x}) = \sqrt{\sum_{i=1}^D x_i^2}$, where $x_i \sim \mathcal{N}(0, 1)$. The expected squared distance is given by $\mathbb{E}\left[d^2\right] = \sum_{i=1}^D \mathbb{E}\left[x_i^2\right] = D$, and the variance of the squared distance is given by $\mathbb{V}\left[d^2\right] = \sum_{i=1}^D \mathbb{V}\left[x_i^2\right] = D$. As D grows, the coefficient of variation (i.e., the SD relative to the mean) goes to zero:

$$\lim_{D\to\infty} \frac{\text{std}\left[d^2\right]}{\mathbb{E}\left[d^2\right]} = \lim_{D\to\infty} \frac{\sqrt{D}}{D} = 0 \tag{2.44}$$

Thus the expected square distance concentrates around D, so the expected distance concentrates around $\mathbb{E}\left[d(\boldsymbol{x})\right] = \sqrt{D}$. See [Ver18] for a more rigorous proof, and **??** for further discussion of typical sets.

2.3.3 Moment form and canonical form

It is common to parameterize the MVN in terms of the mean vector $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$. However, for reasons which are explained in Section 2.3.4, it is sometimes useful to represent the Gaussian distribution using **canonical parameters** or **natural parameters**, defined as

$$\boldsymbol{\Lambda} \triangleq \boldsymbol{\Sigma}^{-1}, \quad \boldsymbol{\xi} \triangleq \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} \tag{2.45}$$

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

$$\boldsymbol{\mu} = \boldsymbol{\Lambda}^{-1}\boldsymbol{\xi}, \quad \boldsymbol{\Sigma} = \boldsymbol{\Lambda}^{-1} \tag{2.46}$$

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

$$\mathcal{N}_c(\boldsymbol{x}|\boldsymbol{\xi},\boldsymbol{\Lambda}) \triangleq c \exp\left(\boldsymbol{x}^\mathsf{T}\boldsymbol{\xi} - \frac{1}{2}\boldsymbol{x}^\mathsf{T}\boldsymbol{\Lambda}\boldsymbol{x}\right) \tag{2.47}$$

$$c \triangleq \frac{\exp(-\frac{1}{2}\boldsymbol{\xi}^\mathsf{T}\boldsymbol{\Lambda}\boldsymbol{\xi})}{(2\pi)^{D/2}\sqrt{\det(\boldsymbol{\Lambda}^{-1})}} \tag{2.48}$$

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

2.3.4 Marginals and conditionals of a MVN

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

$$\boldsymbol{\mu} = \begin{pmatrix}\boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2\end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix}\boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22}\end{pmatrix} \tag{2.49}$$

The marginals of this distribution are given by the following:

$$p(\boldsymbol{x}_1) = \int \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma})d\boldsymbol{x}_2 = \mathcal{N}(\boldsymbol{x}_1|\boldsymbol{\mu}_1,\boldsymbol{\Sigma}_{11}) \tag{2.50}$$

$$p(\boldsymbol{x}_2) = \int \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma})d\boldsymbol{x}_1 = \mathcal{N}(\boldsymbol{x}_2|\boldsymbol{\mu}_2,\boldsymbol{\Sigma}_{22}) \tag{2.51}$$

Thus we just need to extract the relevant rows and columns from $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$.

The conditional distributions can be shown to have the following form (see Section 2.3.4.1 for the proof):

$$p(\boldsymbol{x}_1|\boldsymbol{x}_2) = \mathcal{N}(\boldsymbol{x}_1|\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2),\ \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}) \tag{2.52}$$

$$p(\boldsymbol{x}_2|\boldsymbol{x}_1) = \mathcal{N}(\boldsymbol{x}_2|\boldsymbol{\mu}_2 + \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}(\boldsymbol{x}_1 - \boldsymbol{\mu}_1),\ \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}) \tag{2.53}$$

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

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

$$p(\boldsymbol{x}_2) = \mathcal{N}_c(\boldsymbol{x}_2|\boldsymbol{\xi}_2 - \boldsymbol{\Lambda}_{21}\boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\xi}_1, \boldsymbol{\Lambda}_{22} - \boldsymbol{\Lambda}_{21}\boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12}) \tag{2.54}$$

$$p(\boldsymbol{x}_1|\boldsymbol{x}_2) = \mathcal{N}_c(\boldsymbol{x}_1|\boldsymbol{\xi}_1 - \boldsymbol{\Lambda}_{12}\boldsymbol{x}_2, \boldsymbol{\Lambda}_{11}) \tag{2.55}$$

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

2.3.4.1 Deriving the conditionals of an MVN

Consider a joint Gaussian of the form $p(\boldsymbol{x}_1,\boldsymbol{x}_2) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma})$, where

$$\boldsymbol{\mu} = \begin{pmatrix}\boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2\end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix}\boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22}\end{pmatrix} \tag{2.56}$$

In Section 2.3.4, we claimed that

$$p(\boldsymbol{x}_1|\boldsymbol{x}_2) = \mathcal{N}(\boldsymbol{x}_1|\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2),\ \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}) \tag{2.57}$$

In this section, we derive this result using Schur complenents.

Let us factor the joint $p(\boldsymbol{x}_1,\boldsymbol{x}_2)$ as $p(\boldsymbol{x}_2)p(\boldsymbol{x}_1|\boldsymbol{x}_2)$ as follows:

$$p(\boldsymbol{x}_1,\boldsymbol{x}_2) \propto \exp\left\{-\frac{1}{2}\begin{pmatrix}\boldsymbol{x}_1 - \boldsymbol{\mu}_1 \\ \boldsymbol{x}_2 - \boldsymbol{\mu}_2\end{pmatrix}^\mathsf{T}\begin{pmatrix}\boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22}\end{pmatrix}^{-1}\begin{pmatrix}\boldsymbol{x}_1 - \boldsymbol{\mu}_1 \\ \boldsymbol{x}_2 - \boldsymbol{\mu}_2\end{pmatrix}\right\} \tag{2.58}$$

Using **??** the above exponent becomes

$$p(\boldsymbol{x}_1, \boldsymbol{x}_2) \propto \exp\left\{ -\frac{1}{2} \begin{pmatrix} \boldsymbol{x}_1 - \boldsymbol{\mu}_1 \\ \boldsymbol{x}_2 - \boldsymbol{\mu}_2 \end{pmatrix}^\mathsf{T} \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ -\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21} & \mathbf{I} \end{pmatrix} \begin{pmatrix} (\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_{22}^{-1} \end{pmatrix} \right. \tag{2.59}$$

$$\left. \times \begin{pmatrix} \mathbf{I} & -\boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_1 - \boldsymbol{\mu}_1 \\ \boldsymbol{x}_2 - \boldsymbol{\mu}_2 \end{pmatrix} \right\} \tag{2.60}$$

$$= \exp\left\{ -\frac{1}{2} (\boldsymbol{x}_1 - \boldsymbol{\mu}_1 - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2))^\mathsf{T} (\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1} \right. \tag{2.61}$$

$$\left. (\boldsymbol{x}_1 - \boldsymbol{\mu}_1 - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2)) \right\} \times \exp\left\{ -\frac{1}{2} (\boldsymbol{x}_2 - \boldsymbol{\mu}_2)^\mathsf{T} \boldsymbol{\Sigma}_{22}^{-1} (\boldsymbol{x}_2 - \boldsymbol{\mu}_2) \right\} \tag{2.62}$$

This is of the form

$$\exp(\text{quadratic form in } \boldsymbol{x}_1, \boldsymbol{x}_2) \times \exp(\text{quadratic form in } \boldsymbol{x}_2) \tag{2.63}$$

Hence we have successfully factorized the joint as

$$p(\boldsymbol{x}_1, \boldsymbol{x}_2) = p(\boldsymbol{x}_1|\boldsymbol{x}_2) p(\boldsymbol{x}_2) \tag{2.64}$$

$$= \mathcal{N}(\boldsymbol{x}_1|\boldsymbol{\mu}_{1|2}, \boldsymbol{\Sigma}_{1|2}) \mathcal{N}(\boldsymbol{x}_2|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{22}) \tag{2.65}$$

where the parameters of the conditional distribution can be read off from the above equations using

$$\boldsymbol{\mu}_{1|2} = \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2) \tag{2.66}$$

$$\boldsymbol{\Sigma}_{1|2} = \boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22} = \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21} \tag{2.67}$$

We can also use the fact that $|\mathbf{M}| = |\mathbf{M}/\mathbf{H}||\mathbf{H}|$ to check the normalization constants are correct:

$$(2\pi)^{(d_1+d_2)/2} |\boldsymbol{\Sigma}|^{\frac{1}{2}} = (2\pi)^{(d_1+d_2)/2} (|\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22}|\ |\boldsymbol{\Sigma}_{22}|)^{\frac{1}{2}} \tag{2.68}$$

$$= (2\pi)^{d_1/2} |\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22}|^{\frac{1}{2}}\ (2\pi)^{d_2/2} |\boldsymbol{\Sigma}_{22}|^{\frac{1}{2}} \tag{2.69}$$

where $d_1 = \dim(\boldsymbol{x}_1)$ and $d_2 = \dim(\boldsymbol{x}_2)$.

We see that the equations for marginalization of an MVN are easy, but the equations for conditioning are complex. We can also write the MVN in **information form**, using the parameterization

$$\boldsymbol{\Lambda} \triangleq \boldsymbol{\Sigma}^{-1}, \quad \boldsymbol{\eta} \triangleq \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} \tag{2.70}$$

Here $\boldsymbol{\Lambda}$ is the precision matrix, and $\boldsymbol{\eta}$ is the precision-weighted mean. In this form, one can show that the marginals and conditions are given by

$$p(\boldsymbol{x}_2) = \mathcal{N}_c(\boldsymbol{x}_2|\boldsymbol{\eta}_2 - \boldsymbol{\Lambda}_{21}\boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\eta}_1, \boldsymbol{\Lambda}_{22} - \boldsymbol{\Lambda}_{21}\boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12}) \tag{2.71}$$

$$p(\boldsymbol{x}_1|\boldsymbol{x}_2) = \mathcal{N}_c(\boldsymbol{x}_1|\boldsymbol{\eta}_1 - \boldsymbol{\Lambda}_{12}\boldsymbol{x}_2, \boldsymbol{\Lambda}_{11}) \tag{2.72}$$

To show show this, let us partition $\boldsymbol{\eta}$ as follows:

$$\begin{pmatrix} \boldsymbol{\eta}_1 \\ \boldsymbol{\eta}_2 \end{pmatrix} = \begin{pmatrix} \boldsymbol{\Lambda}_{11} & \boldsymbol{\Lambda}_{12} \\ \boldsymbol{\Lambda}_{21} & \boldsymbol{\Lambda}_{22} \end{pmatrix} \begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix} \tag{2.73}$$

so

$$\boldsymbol{\eta}_1 = \boldsymbol{\Lambda}_{11}\boldsymbol{\mu}_1 + \boldsymbol{\Lambda}_{12}\boldsymbol{\mu}_2 \tag{2.74}$$

$$\boldsymbol{\eta}_2 = \boldsymbol{\Lambda}_{21}\boldsymbol{\mu}_1 + \boldsymbol{\Lambda}_{22}\boldsymbol{\mu}_2 \tag{2.75}$$

Hence

$$\begin{pmatrix} \boldsymbol{\Lambda}_{11} & \boldsymbol{\Lambda}_{12} \\ \boldsymbol{\Lambda}_{21} & \boldsymbol{\Lambda}_{22} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix}^{-1} \tag{2.76}$$

$$= \begin{pmatrix} (\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1} & -(\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1}\boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1} \\ -\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}(\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1} & \boldsymbol{\Sigma}_{22}^{-1} + \boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}(\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1}\boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-2} \end{pmatrix} \tag{2.77}$$

where the top left is

$$\boldsymbol{\Lambda}_{11} = (\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1} = (\boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21})^{-1} \tag{2.78}$$

Using the moment form for conditioning we have

$$\boldsymbol{\Lambda}_{1|2} = \boldsymbol{\Sigma}_{1|2}^{-1} = (\boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21})^{-1} = \boldsymbol{\Lambda}_{11} \tag{2.79}$$

$$\boldsymbol{\mu}_{1|2} = \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2) \tag{2.80}$$

but from Equation 2.77 and 2.78 we have

$$\boldsymbol{\Lambda}_{12} = -\boldsymbol{\Lambda}_{11}\boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1} \tag{2.81}$$

so

$$\boldsymbol{\mu}_{1|2} = \boldsymbol{\mu}_1 - \boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2) \tag{2.82}$$

$$\boldsymbol{\eta}_{1|2} = \boldsymbol{\Lambda}_{1|2}\boldsymbol{\mu}_{1|2} = \boldsymbol{\Lambda}_{11}\boldsymbol{\mu}_1 - \boldsymbol{\Lambda}_{12}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2) \tag{2.83}$$

$$= \boldsymbol{\Lambda}_{11}\boldsymbol{\mu}_1 + \boldsymbol{\Lambda}_{12}\boldsymbol{\mu}_2 - \boldsymbol{\Lambda}_{12}\boldsymbol{x}_2 = \boldsymbol{\eta}_1 - \boldsymbol{\Lambda}_{12}\boldsymbol{x}_2 \tag{2.84}$$

We will now derive the results for marginalizing in information form. Let

$$\begin{pmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\Lambda}_{11} & \boldsymbol{\Lambda}_{12} \\ \boldsymbol{\Lambda}_{21} & \boldsymbol{\Lambda}_{22} \end{pmatrix}^{-1} \tag{2.85}$$

$$= \begin{pmatrix} \boldsymbol{\Lambda}_{11}^{-1} + \boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12}(\boldsymbol{\Lambda}/\boldsymbol{\Lambda}_{11})^{-1}\boldsymbol{\Lambda}_{21}\boldsymbol{\Lambda}_{11}^{-1} & -\boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12}(\boldsymbol{\Lambda}/\boldsymbol{\Lambda}_{11})^{-1} \\ -(\boldsymbol{\Lambda}/\boldsymbol{\Lambda}_{11})^{-1}\boldsymbol{\Lambda}_{21}\boldsymbol{\Lambda}_{11}^{-1} & (\boldsymbol{\Lambda}/\boldsymbol{\Lambda}_{11})^{-1} \end{pmatrix} \tag{2.86}$$

Hence

$$\boldsymbol{\Lambda}_{22}^m = \boldsymbol{\Sigma}_{22}^{-1} = \boldsymbol{\Lambda}/\boldsymbol{\Lambda}_{11} = \boldsymbol{\Lambda}_{22} - \boldsymbol{\Lambda}_{21}\boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12} \tag{2.87}$$

$$\boldsymbol{\eta}_2^m = \boldsymbol{\Lambda}_{22}^m\boldsymbol{\mu}_2^m = (\boldsymbol{\Lambda}_{22} - \boldsymbol{\Lambda}_{21}\boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12})\boldsymbol{\mu}_2 \tag{2.88}$$

$$= \boldsymbol{\Lambda}_{22}\boldsymbol{\mu}_2 - \boldsymbol{\Lambda}_{21}\boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12}\boldsymbol{\mu}_2 \tag{2.89}$$

$$= (\boldsymbol{\Lambda}_{21}\boldsymbol{\mu}_1 + \boldsymbol{\Lambda}_{22}\boldsymbol{\mu}_2) - \boldsymbol{\Lambda}_{21}\boldsymbol{\Lambda}_{11}^{-1}(\boldsymbol{\Lambda}_{11}\boldsymbol{\mu}_1 + \boldsymbol{\Lambda}_{12}\boldsymbol{\mu}_2) \tag{2.90}$$

$$= \boldsymbol{\eta}_2 - \boldsymbol{\Lambda}_{21}\boldsymbol{\Lambda}_{11}\boldsymbol{\eta}_1 \tag{2.91}$$

2.3.5 Bayes' rule for linear Gaussian systems

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

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z) \tag{2.92}$$

$$p(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}|\mathbf{W}\boldsymbol{z} + \boldsymbol{b}, \boldsymbol{\Sigma}_x) \tag{2.93}$$

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

given by

$$p(\boldsymbol{z}, \boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}, \boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{2.94}$$

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_z \\ \mathbf{W}\boldsymbol{\mu}_z + \boldsymbol{b} \end{pmatrix} \tag{2.95}$$

$$\boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_z & \boldsymbol{\Sigma}_z \mathbf{W}^\mathsf{T} \\ \mathbf{W}\boldsymbol{\Sigma}_z & \boldsymbol{\Sigma}_x + \mathbf{W}\boldsymbol{\Sigma}_z \mathbf{W}^\mathsf{T} \end{pmatrix} \tag{2.96}$$

Now suppose $\boldsymbol{z}$ is latent and $\boldsymbol{x}$ is observed. It can be shown (see Section 2.3.5.1) that the posterior $p(\boldsymbol{z}|\boldsymbol{x})$ is given by

$$p(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_{z|x}, \boldsymbol{\Sigma}_{z|x}) \tag{2.97}$$

$$\boldsymbol{\Sigma}_{z|x}^{-1} = \boldsymbol{\Sigma}_z^{-1} + \mathbf{W}^\mathsf{T} \boldsymbol{\Sigma}_x^{-1} \mathbf{W} \tag{2.98}$$

$$\boldsymbol{\mu}_{z|x} = \boldsymbol{\Sigma}_{z|x}[\mathbf{W}^\mathsf{T} \boldsymbol{\Sigma}_x^{-1} (\boldsymbol{x} - \boldsymbol{b}) + \boldsymbol{\Sigma}_z^{-1} \boldsymbol{\mu}_z] \tag{2.99}$$

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

$$\begin{aligned} p(\boldsymbol{x}) &= \int \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z) \mathcal{N}(\boldsymbol{x}|\mathbf{W}\boldsymbol{z} + \boldsymbol{b}, \boldsymbol{\Sigma}_x) d\boldsymbol{z} \\ &= \mathcal{N}(\boldsymbol{x}|\mathbf{W}\boldsymbol{\mu}_z + \boldsymbol{b}, \boldsymbol{\Sigma}_x + \mathbf{W}\boldsymbol{\Sigma}_z \mathbf{W}^\mathsf{T}) \end{aligned} \tag{2.100}$$

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

2.3.5.1 Deriving Bayes rule for linear Gaussian systems

We now derive Bayes rule for Gaussians. The basic idea is to derive the joint distribution, $p(\boldsymbol{z}, \boldsymbol{y}) = p(\boldsymbol{z})p(\boldsymbol{y}|\boldsymbol{z})$, and then to use the results for conditioning Gaussians for computing $p(\boldsymbol{z}|\boldsymbol{y})$.

In more detail, we proceed as follows. The log of the joint distribution is as follows (dropping irrelevant constants):

$$\log p(\boldsymbol{z}, \boldsymbol{y}) = -\frac{1}{2}(\boldsymbol{z} - \boldsymbol{\mu}_z)^T \boldsymbol{\Sigma}_z^{-1} (\boldsymbol{z} - \boldsymbol{\mu}_z) - \frac{1}{2}(\boldsymbol{y} - \mathbf{W}\boldsymbol{z} - \boldsymbol{b})^T \boldsymbol{\Sigma}_y^{-1} (\boldsymbol{y} - \mathbf{W}\boldsymbol{z} - \boldsymbol{b}) \tag{2.101}$$

This is clearly a joint Gaussian distribution, since it is the exponential of a quadratic form.

Expanding out the quadratic terms involving $\boldsymbol{z}$ and $\boldsymbol{y}$, and ignoring linear and constant terms, we have

$$Q = -\frac{1}{2}\boldsymbol{z}^T \boldsymbol{\Sigma}_z^{-1} \boldsymbol{z} - \frac{1}{2}\boldsymbol{y}^T \boldsymbol{\Sigma}_y^{-1} \boldsymbol{y} - \frac{1}{2}(\mathbf{W}\boldsymbol{z})^T \boldsymbol{\Sigma}_y^{-1} (\mathbf{W}\boldsymbol{z}) + \boldsymbol{y}^T \boldsymbol{\Sigma}_y^{-1} \mathbf{W}\boldsymbol{z} \tag{2.102}$$

$$= -\frac{1}{2} \begin{pmatrix} \boldsymbol{z} \\ \boldsymbol{y} \end{pmatrix}^T \begin{pmatrix} \boldsymbol{\Sigma}_z^{-1} + \mathbf{W}^T \boldsymbol{\Sigma}_y^{-1} \mathbf{W} & -\mathbf{W}^T \boldsymbol{\Sigma}_y^{-1} \\ -\boldsymbol{\Sigma}_y^{-1} \mathbf{W} & \boldsymbol{\Sigma}_y^{-1} \end{pmatrix} \begin{pmatrix} \boldsymbol{z} \\ \boldsymbol{y} \end{pmatrix} \tag{2.103}$$

$$= -\frac{1}{2} \begin{pmatrix} \boldsymbol{z} \\ \boldsymbol{y} \end{pmatrix}^T \boldsymbol{\Sigma}^{-1} \begin{pmatrix} \boldsymbol{z} \\ \boldsymbol{y} \end{pmatrix} \tag{2.104}$$

where the precision matrix of the joint is defined as

$$\boldsymbol{\Sigma}^{-1} = \begin{pmatrix} \boldsymbol{\Sigma}_z^{-1} + \mathbf{W}^T \boldsymbol{\Sigma}_y^{-1} \mathbf{W} & -\mathbf{W}^T \boldsymbol{\Sigma}_y^{-1} \\ -\boldsymbol{\Sigma}_y^{-1} \mathbf{W} & \boldsymbol{\Sigma}_y^{-1} \end{pmatrix} \triangleq \boldsymbol{\Lambda} = \begin{pmatrix} \boldsymbol{\Lambda}_{xx} & \boldsymbol{\Lambda}_{xy} \\ \boldsymbol{\Lambda}_{yx} & \boldsymbol{\Lambda}_{yy} \end{pmatrix} \tag{2.105}$$

From Equation 2.52, and using the fact that $\boldsymbol{\mu}_y = \mathbf{W}\boldsymbol{\mu}_z + \boldsymbol{b}$, we have

$$
\begin{aligned}
p(\boldsymbol{z}|\boldsymbol{y}) &= \mathcal{N}(\boldsymbol{\mu}_{z|y}, \boldsymbol{\Sigma}_{z|y}) && (2.106)\\
\boldsymbol{\Sigma}_{z|y} &= \boldsymbol{\Lambda}_{xx}^{-1} = (\boldsymbol{\Sigma}_z^{-1} + \mathbf{W}^T\boldsymbol{\Sigma}_y^{-1}\mathbf{W})^{-1} && (2.107)\\
\boldsymbol{\mu}_{z|y} &= \boldsymbol{\Sigma}_{z|y}\left(\boldsymbol{\Lambda}_{xx}\boldsymbol{\mu}_z - \boldsymbol{\Lambda}_{xy}(\boldsymbol{y} - \boldsymbol{\mu}_y)\right) && (2.108)\\
&= \boldsymbol{\Sigma}_{z|y}\left(\boldsymbol{\Sigma}_z^{-1}\boldsymbol{\mu}_z + \mathbf{W}^\mathsf{T}\boldsymbol{\Sigma}_y^{-1}\mathbf{W}\boldsymbol{\mu}_z + \mathbf{W}^T\boldsymbol{\Sigma}_y^{-1}(\boldsymbol{y} - \boldsymbol{\mu}_y)\right) && (2.109)\\
&= \boldsymbol{\Sigma}_{z|y}\left(\boldsymbol{\Sigma}_z^{-1}\boldsymbol{\mu}_z + \mathbf{W}^\mathsf{T}\boldsymbol{\Sigma}_y^{-1}(\mathbf{W}\boldsymbol{\mu}_z + \boldsymbol{y} - \boldsymbol{\mu}_y)\right) && (2.110)\\
&= \boldsymbol{\Sigma}_{z|y}\left(\boldsymbol{\Sigma}_z^{-1}\boldsymbol{\mu}_z + \mathbf{W}^T\boldsymbol{\Sigma}_y^{-1}(\boldsymbol{y} - \boldsymbol{b})\right) && (2.111)
\end{aligned}
$$

2.3.6 Sensor fusion with known measurement noise

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

$$
p(\boldsymbol{z}, \boldsymbol{x}|\boldsymbol{\theta}) = p(\boldsymbol{z}) \prod_{m=1}^{M} \prod_{n=1}^{N_m} \mathcal{N}(\boldsymbol{x}_{n,m}|\boldsymbol{z}, \boldsymbol{\Sigma}_m) \tag{2.112}
$$

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

In this section, we assume $\boldsymbol{\theta} = (\boldsymbol{\Sigma}_x, \boldsymbol{\Sigma}_y)$ is known. See the supplementary material for the general case. To simplify notation, we assume we just have two different sensors, $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{z}, \boldsymbol{\Sigma}_x)$ and $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{z}, \boldsymbol{\Sigma}_y)$. Pictorially, we can represent this example as $\boldsymbol{x} \leftarrow \boldsymbol{z} \rightarrow \boldsymbol{y}$. We can combine $\boldsymbol{x}$ and $\boldsymbol{y}$ into a single vector $\boldsymbol{v}$, so the model can be represented as $\boldsymbol{z} \rightarrow \boldsymbol{v}$, where $p(\boldsymbol{v}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{v}|\mathbf{W}\boldsymbol{z}, \boldsymbol{\Sigma}_v)$, where $\mathbf{W} = [\mathbf{0}, \mathbf{I}; \mathbf{0}, \mathbf{I}]$ and $\boldsymbol{\Sigma}_v = [\boldsymbol{\Sigma}_x, \mathbf{0}; \mathbf{0}, \boldsymbol{\Sigma}_y]$ are block-structured matrices. We can then apply Bayes' rule for Gaussians (Section 2.3.5) to compute $p(\boldsymbol{z}|\boldsymbol{v})$.

Figure 2.9(a) gives a 2d example, where we set $\boldsymbol{\Sigma}_x = \boldsymbol{\Sigma}_y = 0.01\mathbf{I}_2$, so both sensors are equally reliable. In this case, the posterior mean is halfway between the two observations, $\boldsymbol{x}$ and $\boldsymbol{y}$. In Figure 2.9(b), we set $\boldsymbol{\Sigma}_x = 0.05\mathbf{I}_2$ and $\boldsymbol{\Sigma}_y = 0.01\mathbf{I}_2$, so sensor 2 is more reliable than sensor 1. In this case, the posterior mean is closer to $\boldsymbol{y}$. In Figure 2.9(c), we set

$$
\boldsymbol{\Sigma}_x = 0.01\begin{pmatrix} 10 & 1 \\ 1 & 1 \end{pmatrix}, \quad \boldsymbol{\Sigma}_y = 0.01\begin{pmatrix} 1 & 1 \\ 1 & 10 \end{pmatrix} \tag{2.113}
$$

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

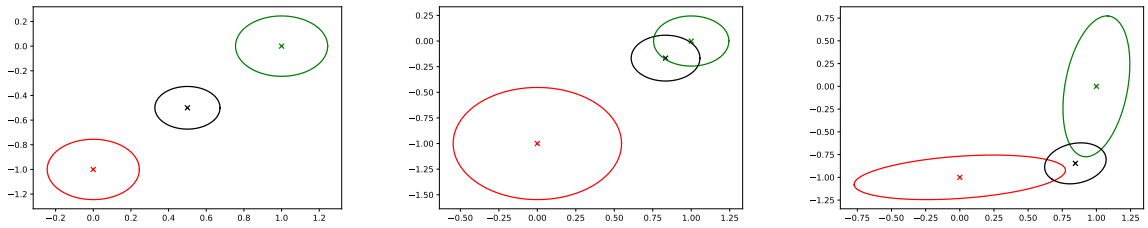
2.3.7 Sensor fusion with unknown measurement noise

In this section, we extend the sensor fusion results from Section 2.3.6 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 [Min01].

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

$$
x_1 = 1.1, x_2 = 1.9, y_1 = 2.9, y_2 = 4.1 \tag{2.114}
$$



(a) *(b)* *(c)*

Figure 2.9: We observe $\boldsymbol{x} = (0, -1)$ (red cross) and $\boldsymbol{y} = (1, 0)$ (green cross) and estimate $\mathbb{E}[\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}]$ (black cross). (a) Equally reliable sensors, so the posterior mean estimate is in between the two circles. (b) Sensor 2 is more reliable, so the estimate shifts more towards the green circle. (c) Sensor 1 is more reliable in the vertical direction, Sensor 2 is more reliable in the horizontal direction. The estimate is an appropriate combination of the two measurements. Generated by sensor_fusion_2d.py.

If the parameters $\boldsymbol{\theta}$ were known, then the posterior would be Gaussian:

$$p(z|\mathcal{D}, \lambda_x, \lambda_y) = \mathcal{N}(z|m_N, \lambda_N^{-1}) \tag{2.115}$$

$$\lambda_N = \lambda_0 + N_x\lambda_x + N_y\lambda_y \tag{2.116}$$

$$m_N = \frac{\lambda_x N_x \overline{x} + \lambda_y N_y \overline{y}}{N_x\lambda_x + N_y\lambda_y} \tag{2.117}$$

where $N_x = 2$ is the number of x measurements, $N_y = 2$ is the number of y measurements, $\overline{x} = \frac{1}{N_x}\sum_{n=1}^{N_x} x_n = 1.5$ and $\overline{y} = \frac{1}{N_y}\sum_{n=1}^{N_y} y_n = 3.5$. 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

$$\ell(z, \lambda_x, \lambda_y) = \frac{N_x}{2}\log\lambda_x - \frac{\lambda_x}{2}\sum_n (x_n - z)^2 + \frac{N_y}{2}\log\lambda_y - \frac{\lambda_y}{2}\sum_n (y_n - z)^2 \tag{2.118}$$

The MLE is obtained by solving the following simultaneous equations:

$$\frac{\partial \ell}{\partial z} = \lambda_x N_x(\overline{x} - z) + \lambda_y N_y(\overline{y} - z) = 0 \tag{2.119}$$

$$\frac{\partial \ell}{\partial \lambda_x} = \frac{1}{\lambda_x} - \frac{1}{N_x}\sum_{n=1}^{N_x}(x_n - z)^2 = 0 \tag{2.120}$$

$$\frac{\partial \ell}{\partial \lambda_y} = \frac{1}{\lambda_y} - \frac{1}{N_y}\sum_{n=1}^{N_y}(y_n - z)^2 = 0 \tag{2.121}$$

This gives

$$\hat{z} = \frac{N_x\hat{\lambda}_x\overline{x} + N_y\hat{\lambda}_y\overline{y}}{N_x\hat{\lambda}_x + N_y\hat{\lambda}_y} \tag{2.122}$$

$$1/\hat{\lambda}_x = \frac{1}{N_x}\sum_n (x_n - \hat{z})^2 \tag{2.123}$$

$$1/\hat{\lambda}_y = \frac{1}{N_y}\sum_n (y_n - \hat{z})^2 \tag{2.124}$$

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 $s_x^2 = \frac{1}{N_x}\sum_{n=1}^{N_x}(x_n - \overline{x})^2 = 0.16$ and $s_y^2 = \frac{1}{N_y}\sum_{n=1}^{N_y}(y_n - \overline{y})^2 = 0.36$. 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 2.10(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 $\mathbb{E}\left[z|\mathcal{D}, \hat{\lambda}_x, \hat{\lambda}_y\right] \approx \overline{x}$, so we effectively ignore the y sensor.

Now we will adopt a Bayesian approach and integrate out the unknown precisions, following **??**. That is, we compute

$$p(z|\mathcal{D}) \propto p(z)\left[\int p(\mathcal{D}_x|z,\lambda_x)p(\lambda_x|z)d\lambda_x\right]\left[\int p(\mathcal{D}_y|z,\lambda_y)p(\lambda_y|z)d\lambda_y\right] \tag{2.125}$$

We will use uninformative Jeffrey priors (**??**) $p(z) \propto 1$, $p(\lambda_x|z) \propto 1/\lambda_x$ 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

$$I = \int p(\mathcal{D}_x|z,\lambda_x)p(\lambda_x|z)d\lambda_x \tag{2.126}$$

$$\propto \int \lambda_x^{-1}\lambda_x^{N_x/2}\exp\left(-\frac{N_x}{2}\lambda_x(\overline{x}-z)^2 - \frac{N_x}{2}s_x^2\lambda_x\right)d\lambda_x \tag{2.127}$$

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

$$I = \int \lambda_x^{-1}\lambda_x^1\exp(-\lambda_x[(\overline{x}-z)^2 + s_x^2])d\lambda_x \tag{2.128}$$

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

$$\mathrm{Ga}(\lambda|a,b) \propto \lambda^{a-1}e^{-\lambda b} \tag{2.129}$$

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

$$I \propto \int p(\mathcal{D}_x|z,\lambda_x)p(\lambda_x|z)d\lambda_x \propto \left((\overline{x}-z)^2 + s_x^2\right)^{-1} \tag{2.130}$$

and the posterior becomes

$$p(z|\mathcal{D}) \propto \frac{1}{(\overline{x}-z)^2 + s_x^2}\ \frac{1}{(\overline{y}-z)^2 + s_y^2} \tag{2.131}$$

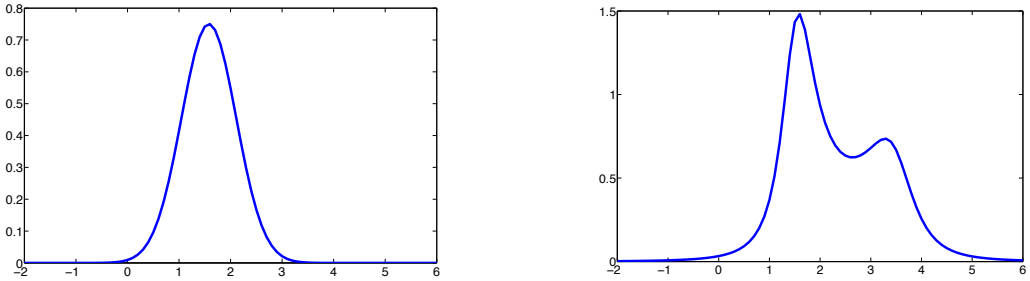
The exact posterior is plotted in Figure 2.10(b). We see that it has two modes, one near $\overline{x} = 1.5$ and one near $\overline{y} = 3.5$. These correspond to the beliefs that the x sensor is more reliable than the y one, and vice versa. The weight of the first mode is larger, since the data from 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:

$$p(\boldsymbol{z}|\mathcal{D}) \propto \int p(\boldsymbol{z}|\mathcal{D},\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta} \tag{2.132}$$

$$\approx \frac{1}{S}\sum_s p(\boldsymbol{z}|\mathcal{D},\boldsymbol{\theta}^s) \tag{2.133}$$

where $\boldsymbol{\theta}^s \sim p(\boldsymbol{\theta}|\mathcal{D})$. Note that $p(\boldsymbol{z}|\mathcal{D},\boldsymbol{\theta}^s)$ 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 **??**.



(a) *(b)*

Figure 2.10: Posterior for z. (a) Plug-in approximation. (b) Exact posterior. Generated by sensor_fusion_unknown_prec.py.

2.3.8 Handling missing data

Suppose we have a linear Gaussian system where we only observe part of $\boldsymbol{y}$, call it $\boldsymbol{y}_1$, while the other part, $\boldsymbol{y}_2$, is hidden. That is, we generalize Equation (2.93) is as follows:

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z) \tag{2.134}$$

$$p\left(\begin{pmatrix}\boldsymbol{y}_1\\ \boldsymbol{y}_2\end{pmatrix}|\boldsymbol{z}\right) = \mathcal{N}\left(\begin{pmatrix}\boldsymbol{y}_1\\ \boldsymbol{y}_2\end{pmatrix}|\begin{pmatrix}\mathbf{W}_1\\ \mathbf{W}_2\end{pmatrix}\boldsymbol{z} + \begin{pmatrix}\boldsymbol{b}_1\\ \boldsymbol{b}_2\end{pmatrix}, \begin{pmatrix}\boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12}\\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22}\end{pmatrix}\right) \tag{2.135}$$

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

$$p(\boldsymbol{z}|\boldsymbol{y}_1) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_{z|1}, \boldsymbol{\Sigma}_{z|1}) \tag{2.136}$$

$$\boldsymbol{\Sigma}_{z|1}^{-1} = \boldsymbol{\Sigma}_z^{-1} + \mathbf{W}_1^{\mathsf{T}}\boldsymbol{\Sigma}_{11}^{-1}\mathbf{W}_1 \tag{2.137}$$

$$\boldsymbol{\mu}_{z|1} = \boldsymbol{\Sigma}_{z|1}[\mathbf{W}_1^{\mathsf{T}}\boldsymbol{\Sigma}_{11}^{-1}\,(\boldsymbol{y}_1 - \boldsymbol{b}_1) + \boldsymbol{\Sigma}_z^{-1}\boldsymbol{\mu}_z] \tag{2.138}$$

2.4 Some other multivariate continuous distributions

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

2.4.1 Multivariate Student distribution

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

$$\mathcal{T}_\nu(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{Z}\left[1 + \frac{1}{\nu}(\boldsymbol{x} - \boldsymbol{\mu})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\right]^{-(\frac{\nu+D}{2})} \tag{2.139}$$

$$Z = \frac{\Gamma(\nu/2)}{\Gamma(\nu/2 + D/2)}\frac{\nu^{D/2}\pi^{D/2}}{|\boldsymbol{\Sigma}|^{-1/2}} \tag{2.140}$$

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

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

$$\text{mean} = \boldsymbol{\mu},\ \text{mode} = \boldsymbol{\mu}, \text{cov} = \frac{\nu}{\nu - 2}\boldsymbol{\Sigma} \tag{2.141}$$

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

2.4.2 Circular normal (von Mises Fisher) distribution

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

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

$$\text{vMF}(\boldsymbol{x}|\boldsymbol{\mu}, \kappa) \triangleq \frac{1}{Z}\exp(\kappa \boldsymbol{\mu}^\mathsf{T}\boldsymbol{x}) \tag{2.142}$$

$$Z = \frac{(2\pi)^{D/2} I_{D/2-1}(\kappa)}{\kappa^{D/2-1}} \tag{2.143}$$

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

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

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

$$\text{vMF}(x|\mu, \kappa) = \frac{1}{Z}\exp(\kappa\cos(x-\mu)) \tag{2.144}$$

$$Z = 2\pi I_0(\kappa) \tag{2.145}$$

2.4.3 Matrix-variate Gaussian (MVG) distribution

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

$$\mathcal{MN}(\mathbf{X}|\mathbf{M}, \mathbf{U}, \mathbf{V}) = \frac{1}{Z}\exp\left(\frac{1}{2}\text{tr}\left[\mathbf{V}^{-1}(\mathbf{X}-\mathbf{M})^\mathsf{T}\mathbf{U}^{-1}(\mathbf{X}-\mathbf{M})\right]\right) \tag{2.146}$$

$$Z = (2\pi)^{np}|\mathbf{V}|^{n/2}|\mathbf{U}|^{p/2} \tag{2.147}$$

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

$$\text{vec}(\boldsymbol{w}) \sim \mathcal{N}(\text{vec}(\mathbf{M}), \mathbf{V}\otimes\mathbf{U}) \tag{2.148}$$

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

2.4.4 Wishart distribution

The **Wishart** distribution is the generalization of the Gamma distribution to positive definite matrices. Press [Pre05, p107] has said "The Wishart distribution ranks next to the (multivariate) normal distribution in order of importance and usefuleness in multivariate statistics". We will mostly use it to model our uncertainty when estimating covariance matrices (see Section 3.2).

The pdf of the Wishart is defined as follows:

$$\text{Wi}(\boldsymbol{\Sigma}|\mathbf{S}, \nu) \triangleq \frac{1}{Z}|\boldsymbol{\Sigma}|^{(\nu-D-1)/2}\exp\left(-\frac{1}{2}\text{tr}(\boldsymbol{\Sigma}\mathbf{S}^{-1})\right) \tag{2.149}$$

$$Z \triangleq |\mathbf{S}|^{-\nu/2}2^{\nu D/2}\Gamma_D(\nu/2) \tag{2.150}$$

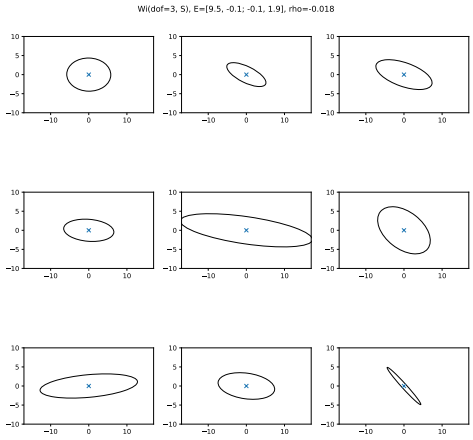


Figure 2.11: Some samples from the Wishart distribution, $\mathbf{\Sigma} \sim \text{Wi}(\mathbf{S}, \nu)$, where $\mathbf{S} = [3.1653, -0.0262; -0.0262, 0.6477]$, and $\nu = 3$. Generated by wishart_plot.py.

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

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

The distribution has these properties:

$$\text{mean} = \nu \mathbf{S}, \ \text{mode} = (\nu - D - 1)\mathbf{S} \tag{2.151}$$

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

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

$$\text{Wi}(\lambda | s^{-1}, \nu) = \text{Ga}(\lambda | \text{shape} = \frac{\nu}{2}, \text{rate} = \frac{1}{2s}) \tag{2.152}$$

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

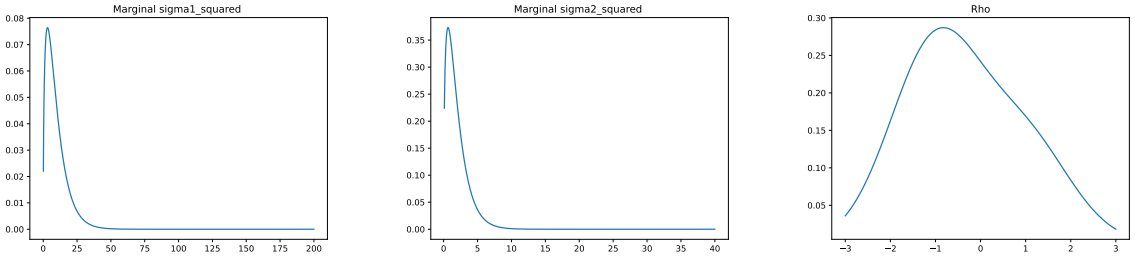
2.4.4.1 Visualizing the Wishart distribution

Since the Wishart is a distribution over matrices, it is hard to plot as a density function. However, we can easily sample from it, and in the 2d case, we can use the eigenvectors of the resulting matrix to define an ellipse. See Figure 2.11 for some examples. For $\nu = 3$. the sampled matrices are highly variable, and some are nearly singular. As ν increases, the sampled matrices are more concentrated on the prior $\mathbf{S}$.

For higher dimensional matrices, we can plot marginals of the distribution. The diagonals of a Wishart distributed matrix have Gamma distributions, given in Equation (2.152), so are easy to plot. It is hard in general to work out the distribution of the off-diagonal elements, but we can sample matrices from the distribution, and then compute the distribution empirically. In particular, we can convert each sampled matrix to a correlation matrix, and thus compute a Monte Carlo approximation to the distribution of the correlation coefficients using

$$R_{ij} = \frac{\mathbf{\Sigma}_{ij}}{\sqrt{\mathbf{\Sigma}_{ii}\mathbf{\Sigma}_{jj}}} \tag{2.153}$$

We can then use kernel density estimation to produce a smooth approximation to the univariate density $p(R_{ij})$ for plotting purposes. See Figure 2.12 for some examples.



(a) *(b)* *(c)*

Figure 2.12: Marginals of the Wishart distribution in Figure 2.11. (a) $p(\Sigma_{11})$. (b) $p(\Sigma_{22})$. (c) $p(\rho)$, where $\rho = \frac{\Sigma_{12}}{\sqrt{\Sigma_{11}\Sigma_{22}}}$ is the correlation coefficient. Generated by wishart_plot.py.

2.4.4.2 Inverse Wishart distribution

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

$$\text{IW}(\mathbf{\Sigma}|\mathbf{S}, \nu) = \frac{1}{Z}|\mathbf{\Sigma}|^{-(\nu+D+1)/2} \exp\left(-\frac{1}{2}\text{tr}(\mathbf{S}\mathbf{\Sigma}^{-1})\right) \tag{2.154}$$

$$Z_{\text{IW}} = |\mathbf{S}|^{-\nu/2} 2^{\nu D/2} \Gamma_D(\nu/2) \tag{2.155}$$

One can show that the distribution has these properties

$$\text{mean} = \frac{\mathbf{S}}{\nu - D - 1}, \ \text{mode} = \frac{\mathbf{S}}{\nu + D + 1} \tag{2.156}$$

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

$$\text{IW}(\sigma^2|s^{-1}, \nu) = \text{IG}(\sigma^2|\nu/2, s/2) \tag{2.157}$$

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

2.4.5 Dirichlet distribution

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

$$S_K = \{\boldsymbol{x} : 0 \leq x_k \leq 1, \sum_{k=1}^{K} x_k = 1\} \tag{2.158}$$

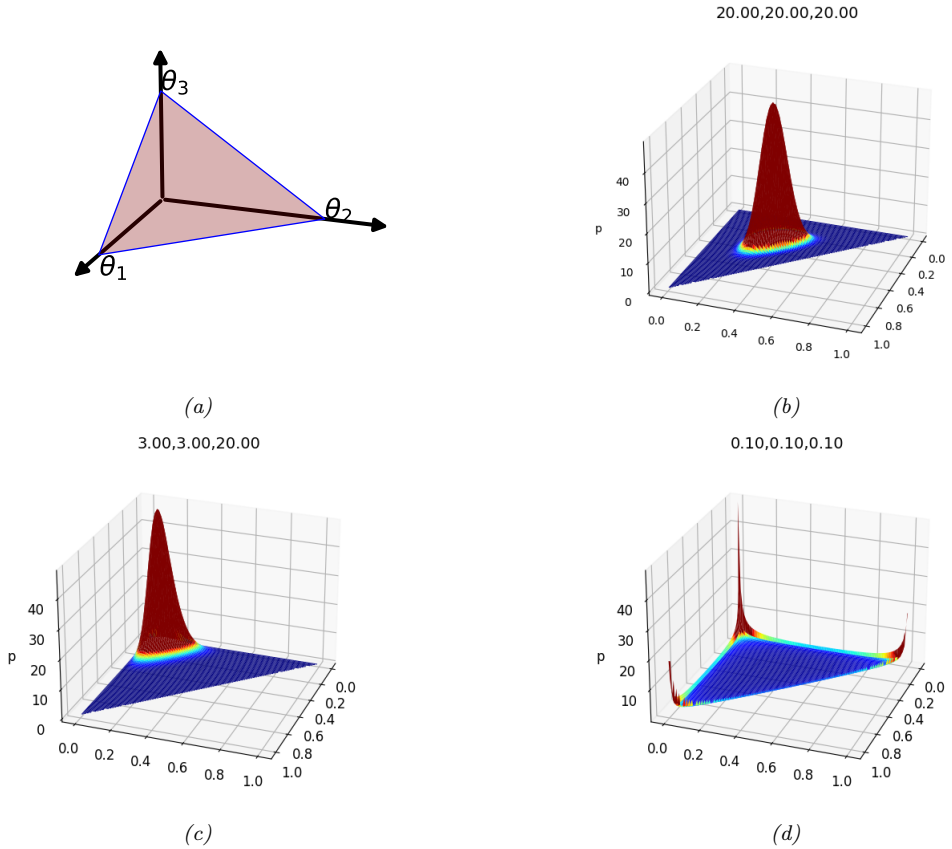
The pdf is defined as follows:

$$\text{Dir}(\boldsymbol{x}|\boldsymbol{\alpha}) \triangleq \frac{1}{B(\boldsymbol{\alpha})} \prod_{k=1}^{K} x_k^{\alpha_k - 1} \mathbb{I}(\boldsymbol{x} \in S_K) \tag{2.159}$$

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

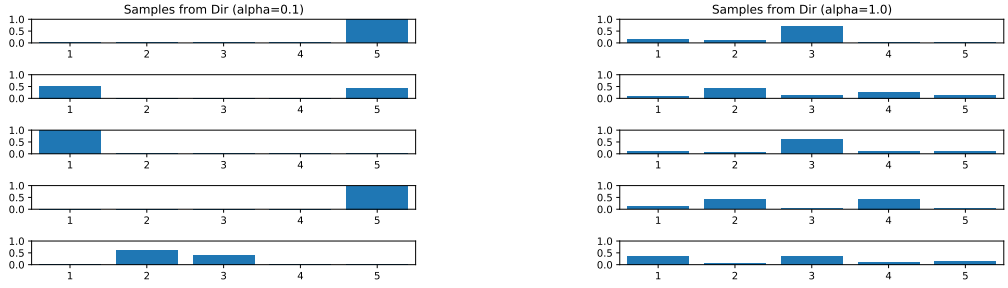
$$B(\boldsymbol{\alpha}) \triangleq \frac{\prod_{k=1}^{K} \Gamma(\alpha_k)}{\Gamma(\sum_{k=1}^{K} \alpha_k)} \tag{2.160}$$

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



(a) *(b)* *(c)* *(d)*

Figure 2.13: (a) The Dirichlet distribution when $K = 3$ defines a distribution over the simplex, which can be represented by the triangular surface. Points on this surface satisfy $0 \leq \theta_c \leq 1$ and $\sum_{c=1}^{3} \theta_c = 1$. Generated by dirichlet_3d_triangle_plot.py. (b) Plot of the Dirichlet density for $\boldsymbol{\alpha} = (20, 20, 20)$. (c) Plot of the Dirichlet density for $\boldsymbol{\alpha} = (3, 3, 20)$. (d) Plot of the Dirichlet density for $\boldsymbol{\alpha} = (0.1, 0.1, 0.1)$. Generated by dirichlet_3d_spiky_plot.py.



(a) *(b)*

Figure 2.14: Samples from a 5-dimensional symmetric Dirichlet distribution for different parameter values. (a) $\boldsymbol{\alpha} = (0.1, \ldots, 0.1)$. This results in very sparse distributions, with many 0s. (b) $\boldsymbol{\alpha} = (1, \ldots, 1)$. This results in more uniform (and dense) distributions. Generated by dirichlet_samples_plot.py.

Figure 2.13 shows some plots of the Dirichlet when $K = 3$. We see that $\alpha_0 = \sum_k \alpha_k$ controls the strength of the distribution (how peaked it is), and the α_k control where the peak occurs. For example, $\text{Dir}(1, 1, 1)$ is a uniform distribution, $\text{Dir}(2, 2, 2)$ is a broad distribution centered at $(1/3, 1/3, 1/3)$, and $\text{Dir}(20, 20, 20)$ is a narrow distribution centered at $(1/3, 1/3, 1/3)$. $\text{Dir}(3, 3, 20)$ is an asymmetric distribution that puts more density in one of the corners. If $\alpha_k < 1$ for all k, we get "spikes" at the corners of the simplex. Samples from the distribution when $\alpha_k < 1$ will be sparse, as shown in Figure 2.14.

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

$$\mathbb{E}[x_k] = \frac{\alpha_k}{\alpha_0}, \; \text{mode}[x_k] = \frac{\alpha_k - 1}{\alpha_0 - K}, \; \mathbb{V}[x_k] = \frac{\alpha_k(\alpha_0 - \alpha_k)}{\alpha_0^2(\alpha_0 + 1)} \tag{2.161}$$

where $\alpha_0 = \sum_k \alpha_k$.

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

2.5 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 [BL06] for a more detailed explanation.

2.5.1 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.[7] 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:

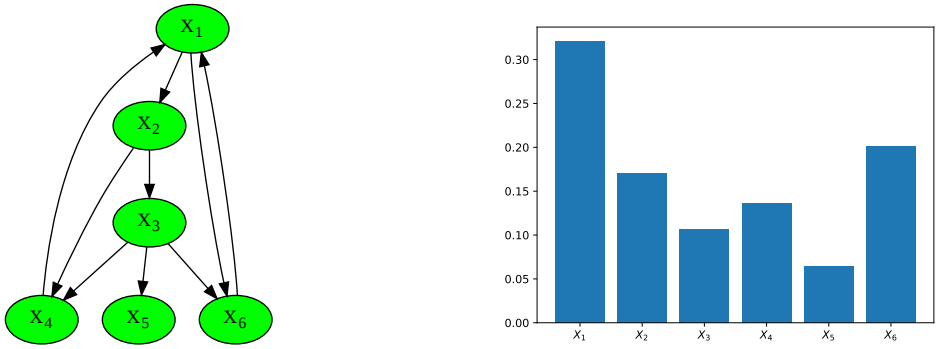
```
"a":      {(2, 2)}
"banana": {(2, 3)}
"is":     {(0, 1), (0, 4), (1, 1), (2, 1)}
"it":     {(0, 0), (0, 3), (1, 2), (2, 0)}
"what":   {(0, 2), (1, 0)}
```

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:

$$\{D_0, D_1\} \cap \{D_0, D_1, D_2\} \cap \{D_0, D_1, D_2\} = \{D_0, D_1\} \tag{2.162}$$

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

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



(a) *(b)*

Figure 2.15: (a) A very small world wide web. Generated by pagerank_small_plot_graph.py (b) The corresponding stationary distribution. Generated by pagerank_demo_small.py.

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**.

2.5.2 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 a 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**:

$$\pi_j = \sum_i A_{ij} \pi_i \tag{2.163}$$

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 (2.163) 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).

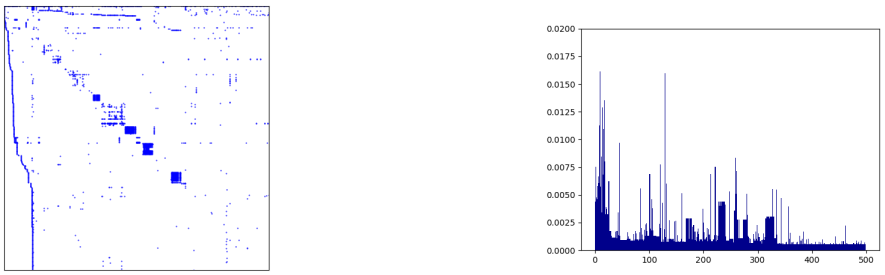
We discuss efficient methods for computing the leading eigenvector of this giant matrix below. Here we ignore computational issues, and just give some examples.

First, consider the small web in Figure 2.15. We find that the stationary distribution is

$$\boldsymbol{\pi} = (0.3209, 0.1706, 0.1065, 0.1368, 0.0643, 0.2008) \tag{2.164}$$

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

As a slightly larger example, Figure 2.16(a) shows a web graph, derived from the root of `harvard.edu`. Figure 2.16(b) shows the corresponding PageRank vector.



(a) *(b)*

Figure 2.16: (a) Web graph of 500 sites rooted at www.harvard.edu. (b) Corresponding page rank vector. Generated by pagerank_demo_harvard.py.

2.5.3 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:

$$M_{ij} = \begin{cases} pG_{ij}/c_j + \delta & \text{if } c_j \neq 0 \\ 1/n & \text{if } c_j = 0 \end{cases} \tag{2.165}$$

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 $\mathbf{M} = \mathbf{A}^\mathsf{T}$ in our earlier notation.

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

$$d_{jj} = \begin{cases} 1/c_j & \text{if } c_j \neq 0 \\ 0 & \text{if } c_j = 0 \end{cases} \tag{2.166}$$

Define the vector $\boldsymbol{z}$ with components

$$z_j = \begin{cases} \delta & \text{if } c_j \neq 0 \\ 1/n & \text{if } c_j = 0 \end{cases} \tag{2.167}$$

Then we can rewrite Equation (2.165) as follows:

$$\mathbf{M} = p\mathbf{G}\mathbf{D} + \mathbf{1}\boldsymbol{z}^\mathsf{T} \tag{2.168}$$

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 $\boldsymbol{v} = \mathbf{M}\boldsymbol{v}$, where $\boldsymbol{v} = \boldsymbol{\pi}^\mathsf{T}$. 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:

$$\boldsymbol{v} \propto \mathbf{M}\boldsymbol{v} = p\mathbf{G}\mathbf{D}\boldsymbol{v} + \mathbf{1}\boldsymbol{z}^\mathsf{T}\boldsymbol{v} \tag{2.169}$$

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 $\boldsymbol{v} = \mathbf{M}\boldsymbol{v}$. Applying this to the data in Figure 2.16(a) yields

the stationary distribution in Figure 2.16(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 $\boldsymbol{v}$ off at the old distribution; this is called warm starting [LM06].

For details on how to perform this Monte Carlo power method in a parallel distributed computing environment, see e.g., [RU10].

2.5.4 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 [Seg11]. 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.

2.5.5 Personalized PageRank

The PageRank algorithm computes a single global notion of importance of each web page. In some cases, it is useful for each user to define his own notion of importance. The **Personalized PageRank** algorithm (aka **random walks with restart**) computes a stationary distribution relative to node k, by returning with some probability to a specific starting node k rather than a random node. The corresponding stationary distribution, $\boldsymbol{\pi}^k$, gives a measure of how important each node is relative to k. See [Lof15] for details. (A similar system is used by Pinterest to infer the similarity of one "pin" (bookmarked webpage) to another, as explained in [Eks+18]).

Chapter 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 technalities.

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 [XT07]. 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 $\boldsymbol{\theta}$ 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 [Jia+13] 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 $h_{\text{even}} = \{2, 4, 6, \ldots\}$, and the concept "powers of two " is defined by $h_{\text{two}} = \{2, 4, 8, 16, \ldots\}$. 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 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

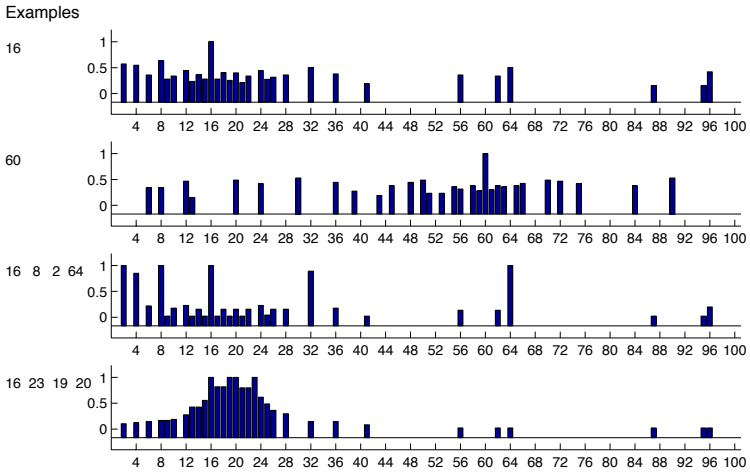


Figure 3.1: Empirical membership distribution in the numbers game, derived from predictions from 8 humans. First two rows: after seeing $\mathcal{D} = \{16\}$ and $\mathcal{D} = \{60\}$. This illustrates diffuse similarity. Third row: after seeing $\mathcal{D} = \{16, 8, 2, 64\}$. This illustrates rule-like 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 [Ten99]. Used with kind permission of Josh Tenenbaum.

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 [Mit97].

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 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

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

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

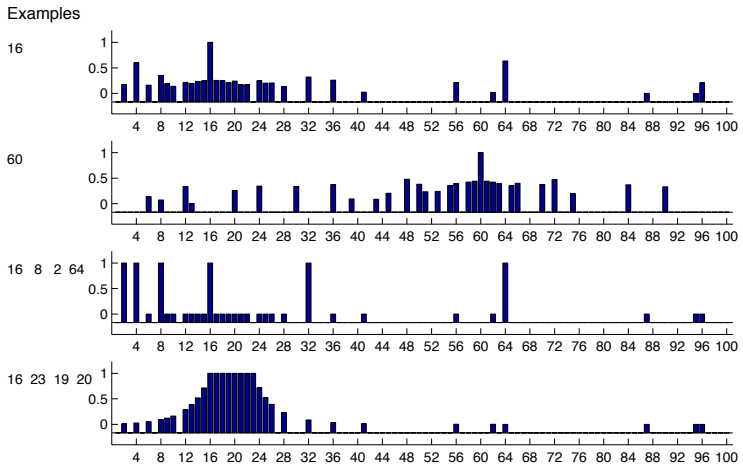


Figure 3.2: Posterior membership probabilities derived using the full hypothesis space. Compare to Figure 3.1. The predictions of the Bayesian model are only plotted for those values for which human data is available; this is why the top line looks sparser than Figure 3.4. From Figure 5.6 of [Ten99]. Used with kind permission of Josh Tenenbaum.

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

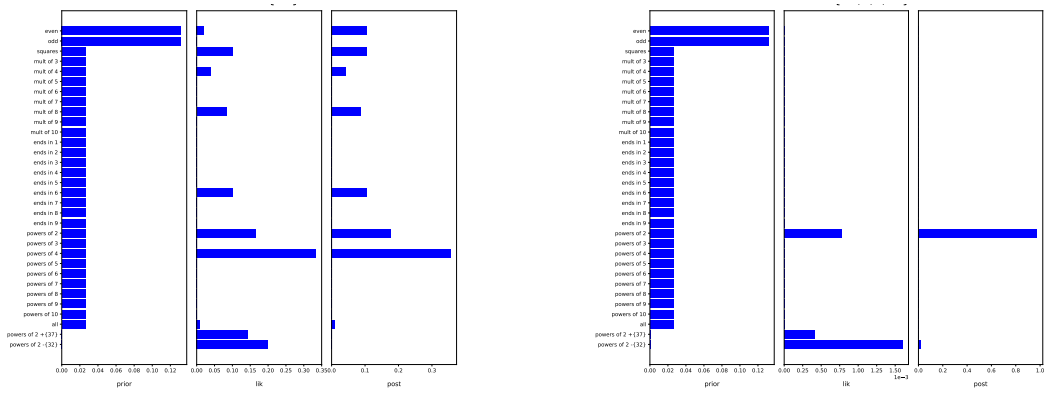
3.1.1.2 Prior

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

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

So, what prior should we use? We will initially consider 30 simple arithmetical concepts, such as "even numbers", "odd numbers", "prime numbers", or "numbers ending in 9". We could use a uniform prior over these concepts; however, for illustration purposes, we make the concepts even and odd more likely apriori, and use a uniform prior over the others. We also include two "unnatural" concepts, namely "powers of 2, plus

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



(a) *(b)*

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 [Ten99]. Generated by numbers_game.py.

37" and "powers of 2, except 32", but give them low prior weight. See Figure 3.3a(bottom row) for a plot of this prior.

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

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

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

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

3.1.1.3 Posterior

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

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

[2] Humans have a natural desire to figure things out; Alison Gopnik, in her paper "Explanation as orgasm" [Gop98], argued that evolution has ensured that we enjoy reducing our posterior uncertainty.

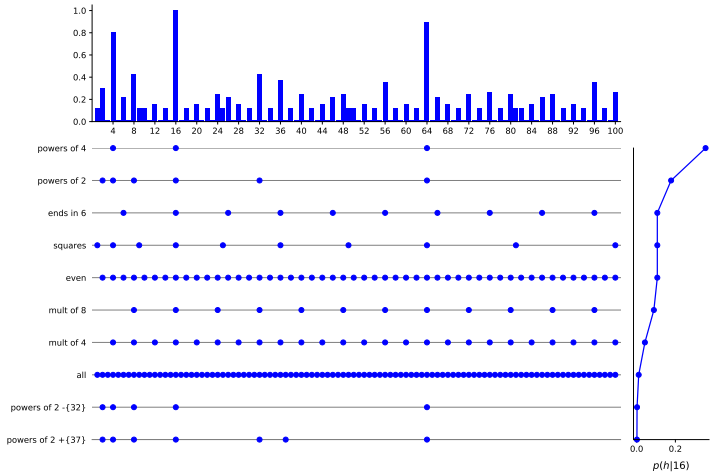


Figure 3.4: Posterior over hypotheses, and the induced posterior over membership, after seeing one example, $\mathcal{D} = \{16\}$. A dot means this number is consistent with this hypothesis. The graph $p(h|\mathcal{D})$ on the right is the weight given to hypothesis h. By taking a weighed sum of dots, we get $p(y \in h|\mathcal{D})$ (top). Adapted from Figure 2.9 of [Ten99]. Generated by numbers_game.py.

This is called **Bayes model averaging** [Hoe+99]. 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.

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:

$$h_{\text{map}} \triangleq \operatorname*{argmax}_h p(h|\mathcal{D}) \tag{3.4}$$

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

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

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

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

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

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

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

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

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

This is called a **plug-in approximation**, and is very widely used, due to its simplicity, as we discuss further in **??**.

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 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.

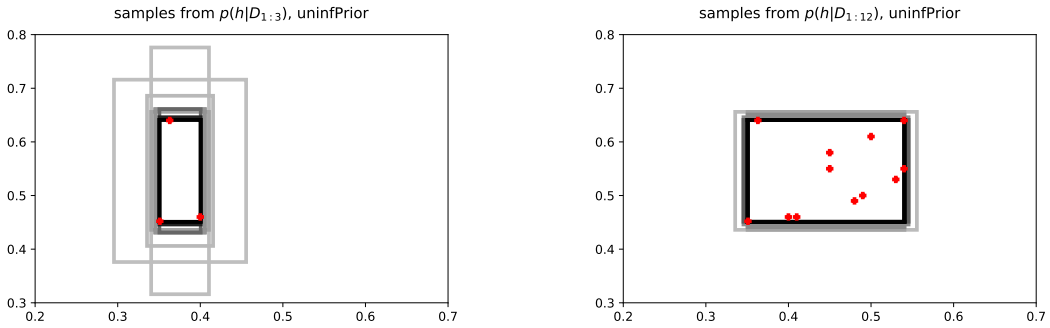
We illustrate these ideas by considering another example of concept learning called the **healthy levels game**, also due to Tenenbaum. The idea is this: we measure two continuous variables, representing the cholesterol and insulin levels of some randomly chosen healthy patients. We would like to know 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 [Mit97]. 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 concepts [Li+19]. 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 [Ten+11].)

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



(a) *(b)*

Figure 3.5: Samples from the posterior in 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 [Ten99]. Generated by healthy_levels_plots.py.

is $p(\mathcal{D}|\ell, s) = (1/s)^N$ if all points are inside the interval, otherwise it is 0. Hence

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

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

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

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 **??**, this means we should use a prior of the form $p(h) \propto \frac{1}{s_1}\frac{1}{s_2}$.

3.1.2.3 Posterior

The posterior is given by

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

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

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

3.1.2.4 Posterior predictive distribution

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

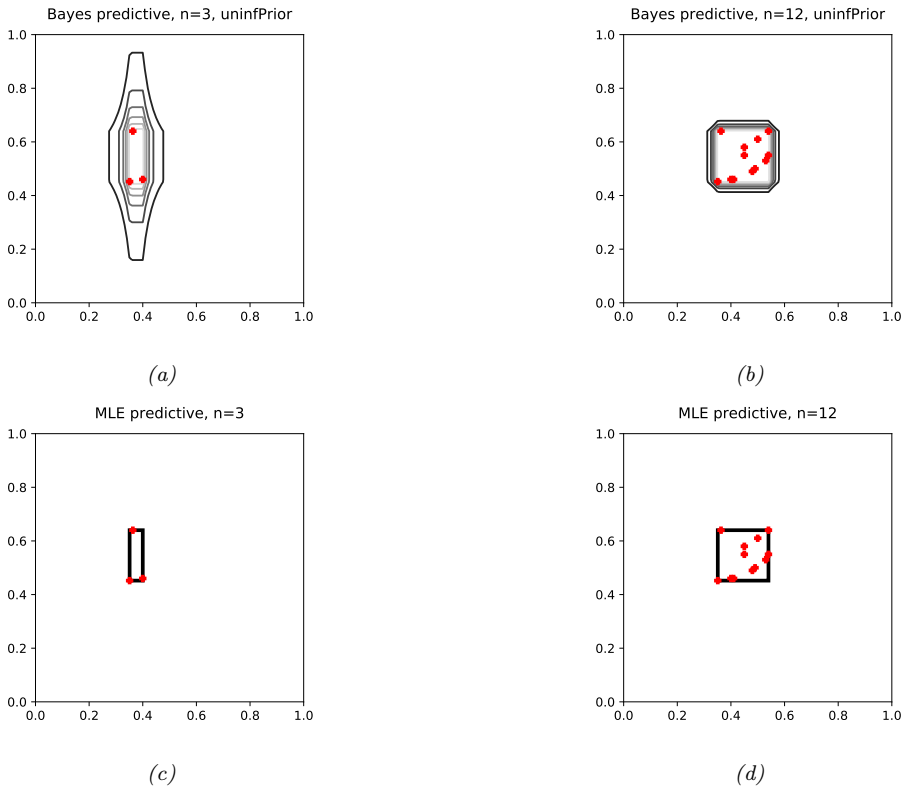


Figure 3.6: Posterior predictive distribution for the healthy levels game. Red crosses are observed data points. Left column: $N = 3$. Right column: $N = 12$. First row: Bayesian prediction. Second row: Plug-in prediction using MLE (smallest enclosing rectangle). We see that the Bayesian prediction goes 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 [Ten99]. Generated by healthy_levels_plot.py.

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 the posterior predictive distributxion is given by

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

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 along dimension j. Thus $p(\boldsymbol{y}|\mathcal{D}) = 1$ if $\boldsymbol{y}$ is inside the support of the training data; if $\boldsymbol{y}$ is outside the support, the probability density drops off, at a rate that depends on N.

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

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

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

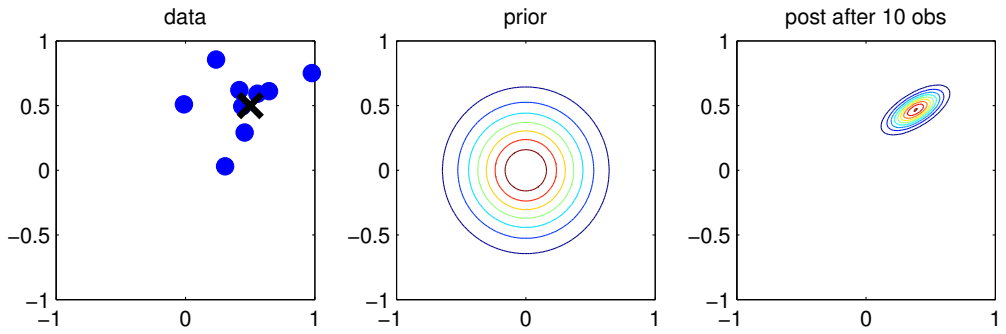


Figure 3.7: Illustration of Bayesian inference for the mean of a 2d Gaussian. (a) The data is generated from $\boldsymbol{y}_n \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$*, where* $\boldsymbol{\mu} = [0.5, 0.5]^\mathsf{T}$ *and* $\boldsymbol{\Sigma} = 0.1[2, 1; 1, 1]$*). (b) The prior is* $p(\boldsymbol{\mu}) = \mathcal{N}(\boldsymbol{\mu}|\mathbf{0}, 0.1\mathbf{I}_2)$*. (c) We show the posterior after 10 data points have been observed. Generated by gaussInferParamsMean2d.py.*

3.1.2.5 Plugin approximation

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

3.2 Conjugate Bayesian analysis for the multivariate Gaussian

In this section, we derive the posterior $p(\boldsymbol{\mu}, \boldsymbol{\Sigma}|\mathcal{D})$ for a multivariate Gaussian. For simplicity, we consider this in three steps: inferring just $\boldsymbol{\mu}$, inferring just $\boldsymbol{\Sigma}$, and then inferring both.

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

The likelihood has the form

$$p(\mathcal{D}|\boldsymbol{\mu}) = \mathcal{N}(\overline{\boldsymbol{y}}|\boldsymbol{\mu}, \frac{1}{N}\boldsymbol{\Sigma}) \tag{3.13}$$

For simplicity, we will use a conjugate prior, which in this case is a Gaussian. In particular, if $p(\boldsymbol{\mu}) = \mathcal{N}(\boldsymbol{\mu}|\, \breve{\boldsymbol{m}}, \breve{\mathbf{V}})$ then we can derive a Gaussian posterior for $\boldsymbol{\mu}$ based on the results in Section 2.3.5 We get

$$p(\boldsymbol{\mu}|\mathcal{D}, \boldsymbol{\Sigma}) = \mathcal{N}(\boldsymbol{\mu}|\, \widehat{\boldsymbol{m}}, \widehat{\mathbf{V}}) \tag{3.14}$$

$$\widehat{\mathbf{V}}^{-1} = \breve{\mathbf{V}}^{-1} + N\boldsymbol{\Sigma}^{-1} \tag{3.15}$$

$$\widehat{\boldsymbol{m}} = \widehat{\mathbf{V}}\,(\boldsymbol{\Sigma}^{-1}(N\overline{\boldsymbol{y}}) + \breve{\mathbf{V}}^{-1}\breve{\boldsymbol{m}}) \tag{3.16}$$

Figure 3.7 gives a 2d example of these results.

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

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

3.2.2.1 Likelihood

We can rewrite the likelihood as follows:

$$p(\mathcal{D}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \propto |\boldsymbol{\Sigma}|^{-\frac{N}{2}} \exp\left(-\frac{1}{2}\text{tr}(\mathbf{S}_\mu \boldsymbol{\Sigma}^{-1})\right) \tag{3.17}$$

where

$$\mathbf{S}_\mu \triangleq \sum_{n=1}^{N} (\boldsymbol{y}_n - \boldsymbol{\mu})(\boldsymbol{y}_n - \boldsymbol{\mu})^\mathsf{T} \tag{3.18}$$

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

3.2.2.2 Prior

The conjugate prior is known as the **inverse Wishart** distribution, which is a distribution over positive definite matrices, as we explained in Section 2.4.4. This has the following pdf:

$$\text{IW}(\boldsymbol{\Sigma}|\, \breve{\mathbf{S}}, \breve{\nu}) \propto |\boldsymbol{\Sigma}|^{-(\breve{\nu}+D+1)/2} \exp\left(-\frac{1}{2}\text{tr}(\breve{\mathbf{S}}\, \boldsymbol{\Sigma}^{-1})\right) \tag{3.19}$$

Here $\breve{\nu} > D-1$ is the degrees of freedom (dof), and $\breve{\mathbf{S}}$ is a symmetric pd matrix. We see that $\breve{\mathbf{S}}$ plays the role of the prior scatter matrix, and $N_0 \triangleq \breve{\nu} + D + 1$ controls the strength of the prior, and hence plays a role analogous to the sample size N.

3.2.2.3 Posterior

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

$$\begin{aligned}
p(\boldsymbol{\Sigma}|\mathcal{D}, \boldsymbol{\mu}) &\propto |\boldsymbol{\Sigma}|^{-\frac{N}{2}} \exp\left(-\frac{1}{2}\text{tr}(\boldsymbol{\Sigma}^{-1}\mathbf{S}_\mu)\right) |\boldsymbol{\Sigma}|^{-(\breve{\nu}+D+1)/2} \\
&\quad \exp\left(-\frac{1}{2}\text{tr}(\boldsymbol{\Sigma}^{-1}\, \breve{\mathbf{S}})\right) && (3.20)\\
&= |\boldsymbol{\Sigma}|^{-\frac{N+(\breve{\nu}+D+1)}{2}} \exp\left(-\frac{1}{2}\text{tr}\left[\boldsymbol{\Sigma}^{-1}(\mathbf{S}_\mu + \breve{\mathbf{S}})\right]\right) && (3.21)\\
&= \text{IW}(\boldsymbol{\Sigma}|\, \widehat{\mathbf{S}}, \widehat{\nu}) && (3.22)\\
\widehat{\nu} &= \breve{\nu} + N && (3.23)\\
\widehat{\mathbf{S}} &= \breve{\mathbf{S}} + \mathbf{S}_\mu && (3.24)
\end{aligned}$$

In words, this says that the posterior strength $\widehat{\nu}$ is the prior strength $\breve{\nu}$ plus the number of observations N, and the posterior scatter matrix $\widehat{\mathbf{S}}$ is the prior scatter matrix $\breve{\mathbf{S}}$ plus the data scatter matrix $\mathbf{S}_\mu$.

3.2.3 Posterior of $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$

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

3.2.3.1 Likelihood

The likelihood is given by

$$p(\mathcal{D}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \propto |\boldsymbol{\Sigma}|^{-\frac{N}{2}} \exp\left(-\frac{1}{2}\sum_{n=1}^{N}(\boldsymbol{y}_n - \boldsymbol{\mu})^\mathsf{T}\boldsymbol{\Sigma}^{-1}(\boldsymbol{y}_n - \boldsymbol{\mu})\right) \tag{3.25}$$

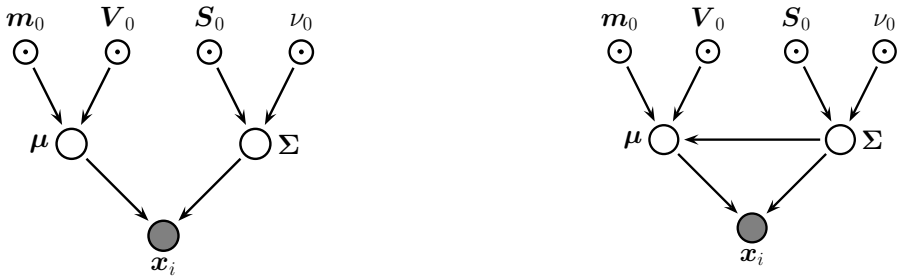


Figure 3.8: Graphical models representing different kinds of assumptions about the parameter priors. (a) A semi-conjugate prior for a Gaussian. (b) A conjugate prior for a Gaussian.

One can show that

$$\sum_{n=1}^{N}(\boldsymbol{y}_n-\boldsymbol{\mu})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{y}_n-\boldsymbol{\mu})=\mathrm{tr}(\boldsymbol{\Sigma}^{-1}\mathbf{S}_{\overline{\boldsymbol{y}}})+N(\overline{\boldsymbol{y}}-\boldsymbol{\mu})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\overline{\boldsymbol{y}}-\boldsymbol{\mu}) \tag{3.26}$$

where

$$\mathbf{S}_{\overline{\boldsymbol{y}}}\triangleq\sum_{n=1}^{N}(\boldsymbol{y}_n-\overline{\boldsymbol{y}})(\boldsymbol{y}_n-\overline{\boldsymbol{y}})^{\mathsf{T}}=\mathbf{Y}^{\mathsf{T}}\mathbf{C}_N\mathbf{Y} \tag{3.27}$$

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

$$\mathbf{C}_N\triangleq\mathbf{I}_N-\frac{1}{N}\mathbf{1}_N\mathbf{1}_N^{\mathsf{T}} \tag{3.28}$$

Hence we can rewrite the likelihood as follows:

$$p(\mathcal{D}|\boldsymbol{\mu},\boldsymbol{\Sigma})\propto|\boldsymbol{\Sigma}|^{-\frac{N}{2}}\exp\left(-\frac{N}{2}(\boldsymbol{\mu}-\overline{\boldsymbol{y}})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}-\overline{\boldsymbol{y}})\right)\exp\left(-\frac{1}{2}\mathrm{tr}(\boldsymbol{\Sigma}^{-1}\mathbf{S}_{\overline{\boldsymbol{y}}})\right) \tag{3.29}$$

We will use this form below.

3.2.3.2 Prior

The obvious prior to use is the following

$$p(\boldsymbol{\mu},\boldsymbol{\Sigma})=\mathcal{N}(\boldsymbol{\mu}|\,\breve{\boldsymbol{m}},\breve{\mathbf{C}})\mathrm{IW}(\boldsymbol{\Sigma}|\,\breve{\mathbf{S}},\breve{\nu}) \tag{3.30}$$

where IW is the inverse Wishart distribution. Unfortunately, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ appear together in a non-factorized way in the likelihood in Equation (3.29) (see the first exponent term), so the factored prior in Equation (3.30) is not conjugate to the likelihood.[3]

The above prior is sometimes called **conditionally conjugate**, since both conditionals, $p(\boldsymbol{\mu}|\boldsymbol{\Sigma})$ and $p(\boldsymbol{\Sigma}|\boldsymbol{\mu})$, are individually conjugate. To create a fully conjugate prior, we need to use a prior where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are dependent on each other. We will use a joint distribution of the form $p(\boldsymbol{\mu},\boldsymbol{\Sigma})=p(\boldsymbol{\mu}|\boldsymbol{\Sigma})p(\boldsymbol{\Sigma})$. Looking at the form of the likelihood equation, Equation (3.29), we see that a natural conjugate prior has the form of a

[3] Using the language of directed graphical models, we see that $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ become dependent when conditioned on $\mathcal{D}$ due to explaining away. See Figure 3.8(a).

Normal-inverse-Wishart or **NIW** distribution, defined as follows:

$$\text{NIW}(\boldsymbol{\mu}, \boldsymbol{\Sigma}| \breve{\boldsymbol{m}}, \breve{\kappa}, \breve{\nu}, \breve{\mathbf{S}}) \triangleq \mathcal{N}(\boldsymbol{\mu}| \breve{\boldsymbol{m}}, \frac{1}{\breve{\kappa}}\boldsymbol{\Sigma}) \times \text{IW}(\boldsymbol{\Sigma}| \breve{\mathbf{S}}, \breve{\nu}) \tag{3.31}$$

$$= \frac{1}{Z_{\text{NIW}}}|\boldsymbol{\Sigma}|^{-\frac{1}{2}} \exp\left(-\frac{\breve{\kappa}}{2}(\boldsymbol{\mu} - \breve{\boldsymbol{m}})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu} - \breve{\boldsymbol{m}})\right)$$

$$\times |\boldsymbol{\Sigma}|^{-\frac{\breve{\nu}+D+1}{2}} \exp\left(-\frac{1}{2}\text{tr}(\boldsymbol{\Sigma}^{-1}\,\breve{\mathbf{S}})\right) \tag{3.32}$$

where the normalization constant is given by

$$Z_{\text{NIW}} \triangleq 2^{\breve{\nu}D/2}\Gamma_D(\breve{\nu}\,/2)(2\pi/\,\breve{\kappa})^{D/2}|\,\breve{\mathbf{S}}\,|^{-\breve{\nu}/2} \tag{3.33}$$

The parameters of the NIW can be interpreted as follows: $\breve{\boldsymbol{m}}$ is our prior mean for $\boldsymbol{\mu}$, and $\breve{\kappa}$ is how strongly we believe this prior; $\breve{\mathbf{S}}$ is (proportional to) our prior mean for $\boldsymbol{\Sigma}$, and $\breve{\nu}$ is how strongly we believe this prior.[4]

3.2.3.3 Posterior

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

$$\mathbf{S}_0 \triangleq \sum_{n=1}^{N} \boldsymbol{y}_n \boldsymbol{y}_n^{\mathsf{T}} = \mathbf{Y}^{\mathsf{T}}\mathbf{Y} \tag{3.34}$$

so we can rewrite the scatter matrix as follows:

$$\mathbf{S}_{\overline{\boldsymbol{y}}} = \mathbf{S}_0 - \frac{1}{N}(\sum_{n=1}^{N}\boldsymbol{y}_n)(\sum_{n=1}^{N}\boldsymbol{y}_n)^{\mathsf{T}} = \mathbf{S}_0 - N\overline{\boldsymbol{y}}\overline{\boldsymbol{y}}^{\mathsf{T}} \tag{3.35}$$

Now we can multiply the likelihood and the prior to give

$$p(\boldsymbol{\mu}, \boldsymbol{\Sigma}|\mathcal{D}) \propto |\boldsymbol{\Sigma}|^{-\frac{N}{2}} \exp\left(-\frac{N}{2}(\boldsymbol{\mu} - \overline{\boldsymbol{y}})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu} - \overline{\boldsymbol{y}})\right) \exp\left(-\frac{1}{2}\text{tr}(\boldsymbol{\Sigma}^{-1}\mathbf{S}_{\overline{\boldsymbol{y}}})\right) \tag{3.36}$$

$$\times |\boldsymbol{\Sigma}|^{-\frac{\breve{\nu}+D+2}{2}} \exp\left(-\frac{\breve{\kappa}}{2}(\boldsymbol{\mu} - \breve{\boldsymbol{m}})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu} - \breve{\boldsymbol{m}})\right) \exp\left(-\frac{1}{2}\text{tr}(\boldsymbol{\Sigma}^{-1}\,\breve{\mathbf{S}})\right) \tag{3.37}$$

$$= |\boldsymbol{\Sigma}|^{-(N+\breve{\nu}+D+2)/2} \exp(-\frac{1}{2}\text{tr}(\boldsymbol{\Sigma}^{-1}\mathbf{M})) \tag{3.38}$$

where

$$\mathbf{M} \triangleq N(\boldsymbol{\mu} - \overline{\boldsymbol{y}})(\boldsymbol{\mu} - \overline{\boldsymbol{y}})^{\mathsf{T}} + \breve{\kappa}\,(\boldsymbol{\mu} - \breve{\boldsymbol{m}})(\boldsymbol{\mu} - \breve{\boldsymbol{m}})^{\mathsf{T}} + \mathbf{S}_{\overline{\boldsymbol{y}}} + \breve{\mathbf{S}} \tag{3.39}$$

$$= (\breve{\kappa} + N)\boldsymbol{\mu}\boldsymbol{\mu}^{\mathsf{T}} - \boldsymbol{\mu}(\breve{\kappa}\breve{\boldsymbol{m}} + N\overline{\boldsymbol{y}})^{\mathsf{T}} - (\breve{\kappa}\breve{\boldsymbol{m}} + N\overline{\boldsymbol{x}})\boldsymbol{\mu}^{\mathsf{T}} + \breve{\kappa}\breve{\boldsymbol{m}}\breve{\boldsymbol{m}}^{\mathsf{T}} + \mathbf{S}_0 + \breve{\mathbf{S}} \tag{3.40}$$

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

$$(\breve{\kappa} + N)\boldsymbol{\mu}\boldsymbol{\mu}^{\mathsf{T}} - \boldsymbol{\mu}(\breve{\kappa}\breve{\boldsymbol{m}} + N\overline{\boldsymbol{y}})^{\mathsf{T}} - (\breve{\kappa}\breve{\boldsymbol{m}} + N\overline{\boldsymbol{x}})\boldsymbol{\mu}^{\mathsf{T}} \tag{3.41}$$

$$= (\breve{\kappa} + N)\left(\boldsymbol{\mu} - \frac{\breve{\kappa}\breve{\boldsymbol{m}} + N\overline{\boldsymbol{y}}}{\breve{\kappa} + N}\right)\left(\boldsymbol{\mu} - \frac{\breve{\kappa}\breve{\boldsymbol{m}} + N\overline{\boldsymbol{x}}}{\breve{\kappa} + N}\right)^{\mathsf{T}} \tag{3.42}$$

$$- \frac{(\breve{\kappa}\breve{\boldsymbol{m}} + N\overline{\boldsymbol{x}})(\breve{\kappa}\breve{\boldsymbol{m}} + N\overline{\boldsymbol{y}})^{\mathsf{T}}}{\breve{\kappa} + N} \tag{3.43}$$

$$= \widehat{\kappa}\,(\boldsymbol{\mu} - \widehat{\boldsymbol{m}})(\boldsymbol{\mu} - \widehat{\boldsymbol{m}})^{\mathsf{T}} - \widehat{\kappa}\widehat{\boldsymbol{m}}\widehat{\boldsymbol{m}}^{\mathsf{T}} \tag{3.44}$$

[4] Note that our uncertainty in the mean is proportional to the covariance. In particular, if we believe that the variance is large, then our uncertainty in μ must be large too. This makes sense intuitively, since if the data has large spread, it will be hard to pin down its mean.

Hence we can rewrite the posterior as follows:

$$p(\boldsymbol{\mu}, \boldsymbol{\Sigma}|\mathcal{D}) \propto |\boldsymbol{\Sigma}|^{(\widehat{\nu}+D+1)/2} \exp\left(-\frac{1}{2}\text{tr}\left[\boldsymbol{\Sigma}^{-1}\left(\widehat{\kappa}\ (\boldsymbol{\mu}-\widehat{\boldsymbol{m}})(\boldsymbol{\mu}-\widehat{\boldsymbol{m}})^{\mathsf{T}}+\widehat{\mathbf{S}}\right)\right]\right) \tag{3.45}$$

$$= \text{NIW}(\boldsymbol{\mu}, \boldsymbol{\Sigma}|\ \widehat{\boldsymbol{m}}, \widehat{\kappa}, \widehat{\nu}, \widehat{\mathbf{S}}) \tag{3.46}$$

where

$$\widehat{\boldsymbol{m}} = \frac{\breve{\kappa}\breve{\boldsymbol{m}} + N\overline{\boldsymbol{y}}}{\widehat{\kappa}} = \frac{\breve{\kappa}}{\breve{\kappa}+N}\ \breve{\boldsymbol{m}} + \frac{N}{\breve{\kappa}+N}\overline{\boldsymbol{y}} \tag{3.47}$$

$$\widehat{\kappa} = \breve{\kappa} + N \tag{3.48}$$

$$\widehat{\nu} = \breve{\nu} + N \tag{3.49}$$

$$\widehat{\mathbf{S}} = \breve{\mathbf{S}} + \mathbf{S}_{\overline{\boldsymbol{y}}} + \frac{\breve{\kappa}\ N}{\breve{\kappa}+N}(\overline{\boldsymbol{y}}-\breve{\boldsymbol{m}})(\overline{\boldsymbol{y}}-\breve{\boldsymbol{m}})^{\mathsf{T}} \tag{3.50}$$

$$= \breve{\mathbf{S}} + \mathbf{S}_0 + \breve{\kappa}\breve{\boldsymbol{m}}\breve{\boldsymbol{m}}^{\mathsf{T}} - \widehat{\kappa}\widehat{\boldsymbol{m}}\widehat{\boldsymbol{m}}^{\mathsf{T}} \tag{3.51}$$

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

3.2.3.4 Posterior marginals

We have computed the joint posterior

$$p(\boldsymbol{\mu}, \boldsymbol{\Sigma}|\mathcal{D}) = \mathcal{N}(\boldsymbol{\mu}|\boldsymbol{\Sigma}, \mathcal{D})p(\boldsymbol{\Sigma}|\mathcal{D}) = \mathcal{N}(\boldsymbol{\mu}|\ \widehat{\boldsymbol{m}}, \frac{1}{\widehat{\kappa}}\boldsymbol{\Sigma})\text{IW}(\boldsymbol{\Sigma}|\ \widehat{\mathbf{S}}, \widehat{\nu}) \tag{3.52}$$

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

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

$$p(\boldsymbol{\Sigma}|\mathcal{D}) = \int p(\boldsymbol{\mu}, \boldsymbol{\Sigma}|\mathcal{D})d\boldsymbol{\mu} = \text{IW}(\boldsymbol{\Sigma}|\ \widehat{\mathbf{S}}, \widehat{\nu}) \tag{3.53}$$

For the mean, one can show that

$$p(\boldsymbol{\mu}|\mathcal{D}) = \int p(\boldsymbol{\mu}, \boldsymbol{\Sigma}|\mathcal{D})d\boldsymbol{\Sigma} = \mathcal{T}(\boldsymbol{\mu}|\ \widehat{\boldsymbol{\mu}}, \frac{\widehat{\mathbf{S}}}{\widehat{\kappa}\widehat{\nu}'}, \widehat{\nu}') \tag{3.54}$$

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

3.2.3.5 Posterior predictive

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

$$p(\boldsymbol{y}|\mathcal{D}) = \int \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\text{NIW}(\boldsymbol{\mu}, \boldsymbol{\Sigma}|\ \widehat{\boldsymbol{m}}, \widehat{\kappa}, \widehat{\nu}, \widehat{\mathbf{S}})d\boldsymbol{\mu}d\boldsymbol{\Sigma} \tag{3.55}$$

$$= \mathcal{T}(\boldsymbol{y}|\ \widehat{\boldsymbol{m}}, \frac{\widehat{\mathbf{S}}\ (\widehat{\kappa}+1)}{\widehat{\kappa}\widehat{\nu}'}, \widehat{\nu}') \tag{3.56}$$

where $\widehat{\nu}' = \widehat{\nu} - D + 1$.

3.2.3.6 Posterior of $\boldsymbol{\Lambda}$ and $\boldsymbol{\mu}$

For completeness, we also state the results in terms of the precision matrix, $\boldsymbol{\Lambda} \triangleq \boldsymbol{\Sigma}^{-1}$, following [DeG70, p178]. If $\boldsymbol{\Sigma} \sim \text{IW}(\mathbf{S}, \nu + D + 1)$, then $\boldsymbol{\Sigma}^{-1} \sim \text{Wi}(\mathbf{S}^{-1}, \nu)$, where Wi is the Wishart distribution given in Equation (2.149). Similarly, we define the **normal Wishart** distribution as follows:

$$\text{NWI}(\boldsymbol{\mu}, \boldsymbol{\Lambda} | \breve{\boldsymbol{m}}, \breve{\kappa}, \breve{\nu}, \breve{\mathbf{S}}) \triangleq \mathcal{N}(\boldsymbol{\mu} | \breve{\boldsymbol{m}}, (\breve{\kappa}\, \boldsymbol{\Lambda})^{-1}) \times \text{Wi}(\boldsymbol{\Lambda} | \breve{\mathbf{S}}, \breve{\nu}) \tag{3.57}$$

The corresponding posterior is given by

$$p(\boldsymbol{\mu}, \boldsymbol{\Lambda} | \mathcal{D}) = \text{NWI}(\boldsymbol{\mu}, \boldsymbol{\Lambda} | \widehat{\boldsymbol{m}}, \widehat{\kappa}, \widehat{\nu}, \widehat{\mathbf{S}}) \tag{3.58}$$

where the posterior parameters are defined in Equation (3.47)–Equation (3.51). The posterior marginals are given by

$$p(\boldsymbol{\Lambda} | \mathcal{D}) = \text{Wi}(\boldsymbol{\Lambda} | \widehat{\mathbf{S}}, \widehat{\nu}) \tag{3.59}$$

$$p(\boldsymbol{\mu} | \mathcal{D}) = \mathcal{T}(\boldsymbol{\mu} | \widehat{\boldsymbol{\mu}}, \frac{\widehat{\mathbf{S}}}{\widehat{\kappa}\widehat{\nu}'}, \widehat{\nu}') \tag{3.60}$$

where $\widehat{\nu}' \triangleq \widehat{\nu} - D + 1$.

3.2.4 Posterior using LKJ prior

Although the (inverse) Wishart prior is conjugate, it has heavy tails, and is hard to sample from. Consequently it is common to use the **LKJ distribution** as a prior instead (this is named after the authors of [LKJ09]). This is a prior on correlation matrices $\mathbf{C}$ of the form $p(\mathbf{C}|\eta) \propto |\mathbf{C}|^{\eta-1}$, so $\eta = 1$ leads to a uniform a distribution. We can combine this with a suitable prior for the standard deviation, such as half-Cauchy prior (**??**), to derive a prior for $\boldsymbol{\Sigma} = \sigma\mathbf{C}$. We can no longer derive the posterior in closed form, but we can still use approximate inference methods, such as MCMC.

3.3 Informative priors

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

For example, consider the classic **taxicab problem** [Jay03, p190]: 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 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 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 [GT06].

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

$$p(T|t) = \frac{p(t|T)p(T)}{p(t)} \tag{3.61}$$

where

$$p(t) = \int_0^\infty p(t|T)p(T)dT = \int_t^\infty \frac{p(T)}{T} dT \tag{3.62}$$

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

$$p(T \geq \hat{T}|t) = \int_{\hat{T}}^\infty p(T|t)dT = 0.5 \tag{3.63}$$

Note that the posterior median is often a better summary of the posterior than the posterior mode, for reasons explained in **??**.

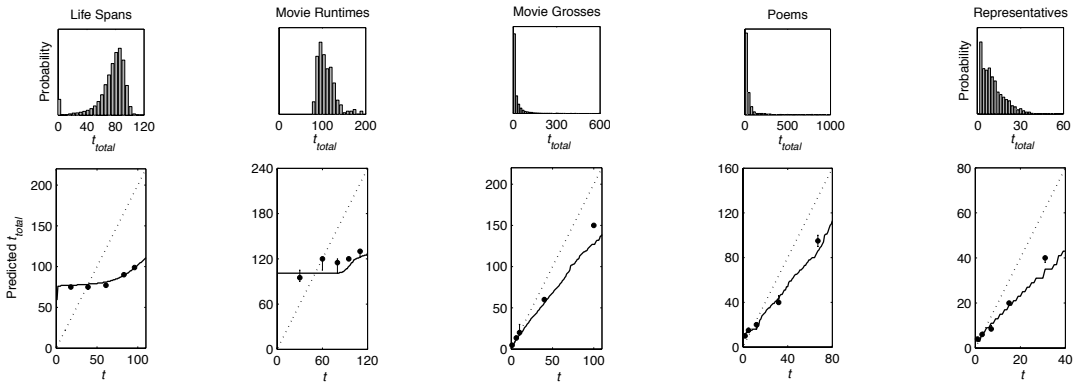


Figure 3.9: Top: empirical distribution of various durational quantities. Bottom: predicted total duration as a function of observed duration, $p(T|t)$. Dots are observed median responses of people. Solid line: Bayesian prediction using informed prior. Dotted line: Bayesian prediction using uninformative prior. From Figure 2a of [GT06]. Used with kind permission of Tom Griffiths.

3.3.1 Domain specific priors

At the top of Figure 3.9, 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 poem, and the number of years someone serves in the US house of Representatives. (The sources for these data is listed in [GT06].)

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 [GT06]. In the sections below, we discuss some suitable parametric priors which catpure this behavior. In [GT06], 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.

3.3.2 Gaussian prior

Looking at Figure 3.9(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 3.10 (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 $\hat{T}(80) \approx 80 + \epsilon$.

3.3.3 Power-law prior

Looking at Figure 3.9(c-d), it seems clear that movie grosses and poem length can be modeled by a **power law** distribution of the form $p(T) \propto T^{-\gamma}$ for $\gamma > 0$. (If $\gamma > 1$, this is called a Pareto distribution, see Section 2.2.9.) Power-laws are characterized by having very **long tails**. This captures the fact that most

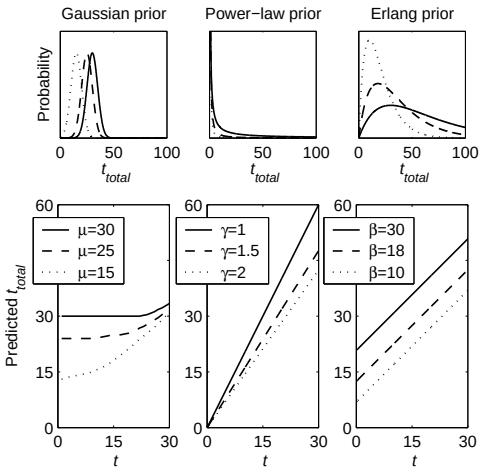


Figure 3.10: Top: three different prior distributions, for three different parameter values. Bottom: corresponding predictive distributions. From Figure 1 of [GT06]. Used with kind permission of Tom Griffiths.

movies make very little money, but a few blockbusters make a lot. The number of lines in various poems also has this shape, since there are a few epic poems, such as Homer's *Odyssey*, but most are short, like haikus. Wealth has a similarly skewed distribution in many countries, especially in plutocracies such as the USA (see e.g., inequality.org).

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

$$p(t) \propto \int_t^\infty T^{-(\gamma+1)} dT = -\frac{1}{\gamma} T^{-\gamma}|_t^\infty = \frac{1}{\gamma} t^{-\gamma} \tag{3.64}$$

Hence the posterior becomes

$$p(T|t) = \frac{T^{-(\gamma+1)}}{\frac{1}{\gamma} t^{-\gamma}} = \frac{\gamma t^\gamma}{T^{\gamma+1}} \tag{3.65}$$

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

$$p(T > T_M|t) = \int_{T_M}^\infty \frac{\gamma t^\gamma}{T^{\gamma+1}} dT = -\left(\frac{t^\gamma}{T}\right)|_{T_M}^\infty = \left(\frac{t}{T_M}\right)^\gamma \tag{3.66}$$

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

This is plotted in Figure 3.10 (bottom middle). We see that the predicted duration is some constant multiple of the observed duration. For the particular value of γ that best fits the empirical distribution of movie grosses, the optimal prediction is about 50% larger than the observed quantity. So if we observe that a movie has made \$40M to date, we predict that it will make \$60M in total.

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

3.3.4 Erlang prior

Looking at Figure 3.9(e), it seems clear that the number of years a US Representative is approximately modeled by a gamma distribution (Section 2.2.8). Griffiths and Tenenbaum use a special case of the Gamma distributon, where the shape parameter is $a = 2$; this is known as the **Erlang distribution**:

$$p(T) = \text{Ga}(T|2, 1/\beta) \propto T e^{-T/\beta} \tag{3.67}$$

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

$$p(t) \propto \int_t^\infty \exp(-T/\beta)dT = -\beta \exp(-T/\beta)|_t^\infty = \beta \exp(-t/\beta) \tag{3.68}$$

so the posterior has the form

$$p(T|t) = \frac{\exp(-T/\beta)}{\beta \exp(-t/\beta)} = \frac{1}{\beta} \exp(-(T-t)/\beta) \tag{3.69}$$

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

$$p(T > T_M|t) = \int_{T_M}^\infty \frac{1}{\beta} \exp(-(T-t)/\beta)dT = -\exp(-(T-t)/\beta)|_{T_M}^\infty = \exp(-(T_M - t)/\beta) \tag{3.70}$$

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 3.10 (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.

Chapter 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 $C \to R$ arc. We know that the cloudy season makes turning on a water sprinkler less likely, so we add a $C \to S$ arc. We know that either rain or sprinklers can cause the grass to get wet, so we add $S \to W$ and $R \to W$ edges. Finally, we know that wet grass can be slippery, so we add a $W \to L$ edge. See Figure 4.1 for the resulting DAG.

Formally, this defines the following joint distribution:

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

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

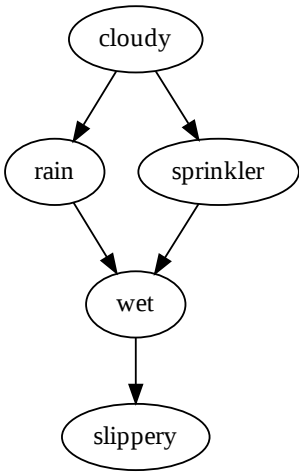
In addition to the graph structure, we need to specify the CPDs. For discrete random variables, we can represent the CPD as a table, which means we have a separate row (i.e., a separate categorical distribution) for each **conditioning case**, i.e., for each combination of parent values. This is known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

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

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, $i \in [V]$; k indexes node states, $k \in [K_i]$, where K_i is the number of states for node i; and j indexes joint parent states, $j \in [J_i]$, where $J_i = \prod_{p \in \text{pa}(i)} K_p$. 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 we see that the grass is wet: our belief that it rained changes to $p(R = 1|W = 1) = 0.7079$. Now suppose we also notice the water sprinkler was turned on: our belief that it rained goes down to $p(R = 1|W = 1, S = 1) = 0.3204$. This negative mutual



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

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

interaction between multiple causes of some observations is called the **explaining away** effect, also known as **Berkson's paradox** (see **??** for details).

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

Note also that inference requires a fully-specified model. This means we need to know the graph structure G and the parameters of the CPDs $\boldsymbol{\theta}$. We discuss how to learn the parameters in **??** and the structure in **??**.

4.1.2 Asia network

In this section, we consider a hypothetical medical model proposed in [LS88] which is known in the literature as the "**Asia network**". (The name comes from the fact that was designed to diagnoise various lung diseases in Western patients returning from a trip to Asia. Note that this example predates the COVID-19 pandemic by many years, and is a purely fictitious model.)

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

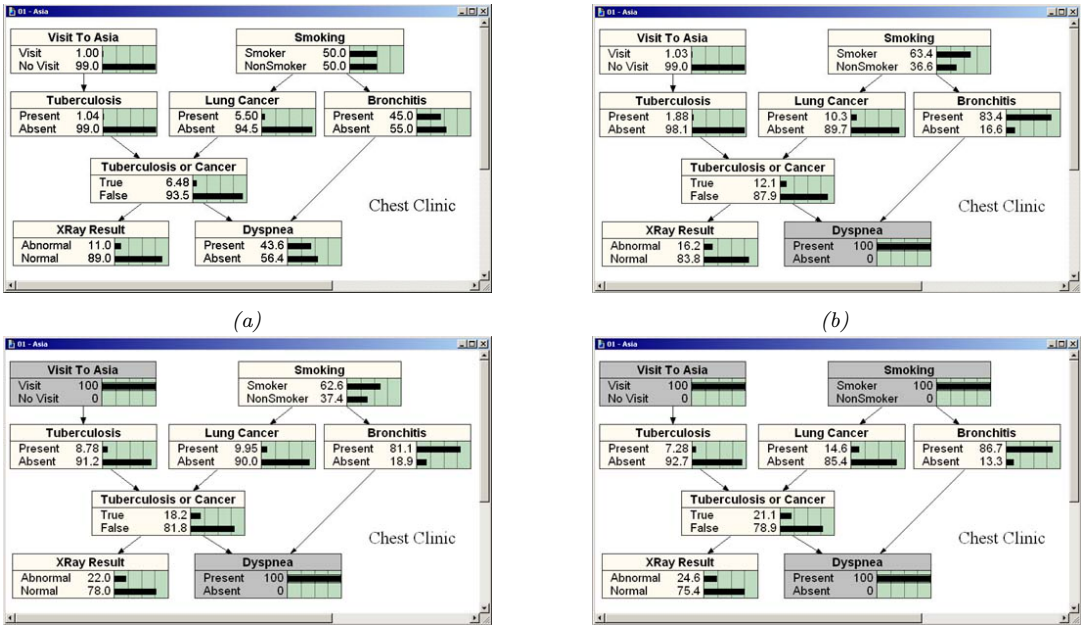


Figure 4.2: Illustration of belief updating in the "Asia" PGM. The histograms show the marginal distribution of each node. (a) Prior. (b) Posterior after conditioning on Dyspnea=Present. (c) Posterior after also conditioning on VisitToAsia=True. (d) Posterior after also conditioning on Smoking=True. Generated by asia_pgm.ipynb.

4.1.3 The QMR network

In this section, we describe the PGM-D known as the **quick medical reference** or **QMR** network [Shw+91]. This is a model of infectious diseases and is shown (in simplified form) in Figure 4.3. (We 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:

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

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., $p(z_k|\boldsymbol{x}_v)$, where $\boldsymbol{x}_v$ 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 (**??**), 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, $p(x_d|\boldsymbol{z}_{\text{pa}(d)}) = \text{Ber}(x_d|\sigma(\boldsymbol{w}_d^{\mathsf{T}}\boldsymbol{z}_{\text{pa}(d)}))$. 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 $k \to d$ link fails. The only way for the child to be off is if all the links from all parents

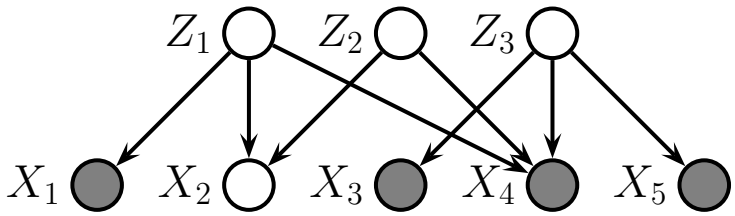


Figure 4.3: A small version of the QMR network. All nodes are binary. The hidden nodes z_k represent diseases, and the visible nodes x_d represent symptoms. In the full network, there are 570 hidden (disease) nodes and 4075 visible (symptom) nodes. The shaded (solid gray) leaf nodes are observed; in 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.

that are on fail independently at random. Thus

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

Obviously, $p(x_d = 1|\boldsymbol{z}) = 1 - p(x_d = 0|\boldsymbol{z})$. In particular, let us define $q_{kd} = 1 - \theta_{kd} = p(x_d = 1|z_k = 1, \boldsymbol{z}_{-k} = 0)$; this is the probability that k can activate d "on its own"; this is sometimes called its "**causal power**" (see e.g., [KNH11]).

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

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

See Table 4.1 for a numerical example.

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

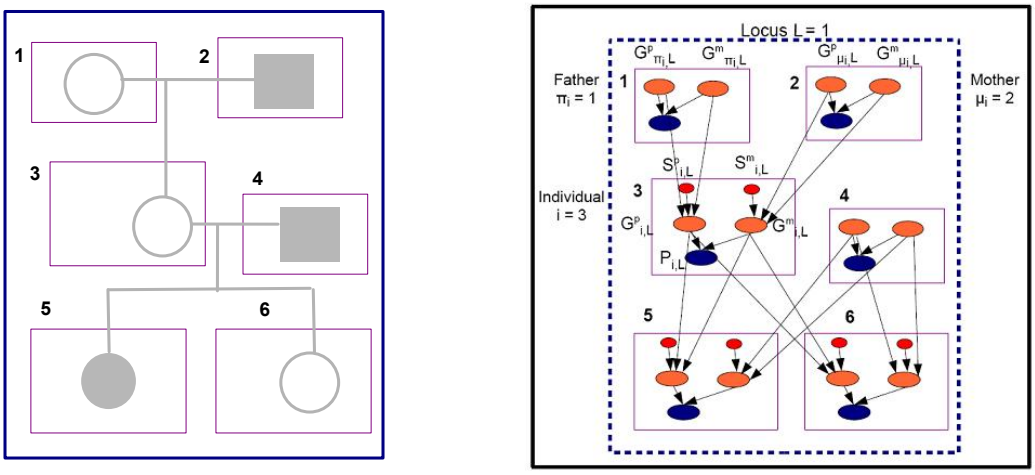
$$p(x_d = 1|\boldsymbol{z}) = 1 - \exp\left(w_{0d} + \sum_k z_k w_{kd}\right) \tag{4.6}$$

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 **??** (see also [Nea92; MH97]).

4.1.4 Genetic linkage analysis

PGM-D'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.



(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 [FGL00].

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.

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.

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 $p(P_{ij}|G_{ij}^m, G_{ij}^p)$ 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 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

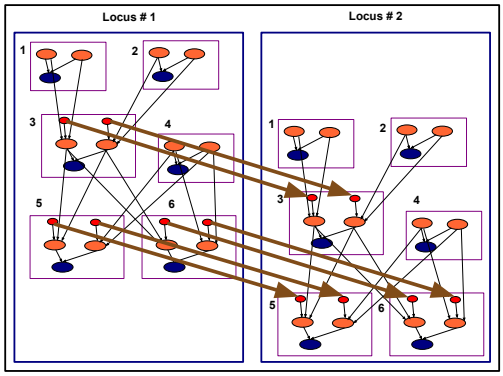


Figure 4.5: Extension of Figure 4.4 to two loci, showing how the switching variables are spatially correlated. This is indicated by the $S^m_{ij} \to S^m_{i,j+1}$ and $S^p_{ij} \to S^p_{i,j+1}$ edges. Adapted from Figure 3 from [FGL00].

variable that specifies the choice. Then we can use the following CPD, known as the **inheritance model**:

$$p(G^m_{ij}|G^m_{kj}, G^p_{kj}, S^m_{ij}) = \begin{cases} \mathbb{I}\left(G^m_{ij} = G^m_{kj}\right) & \text{if } S^m_{ij} = m \\ \mathbb{I}\left(G^m_{ij} = G^p_{kj}\right) & \text{if } S^m_{ij} = p \end{cases} \tag{4.7}$$

We can define $p(G^p_{ij}|G^m_{kj}, G^p_{kj}, S^p_{ij})$ 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 $G^p_{i,j}$, $G^m_{i,j}$, $S^p_{i,j}$ and $S^m_{i,j}$ constitute the **haplotype** of person i at locus j. (The genotype $G^p_{i,j}$ and $G^m_{i,j}$ without the switching variables $S^p_{i,j}$ and $S^m_{i,j}$ is called the "unphased" genotype.)

Next, we need to specify the prior for the root nodes, $p(G^m_{ij})$ and $p(G^p_{ij})$. This is called the **founder model**, and represents the overall prevalence of difference 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 amonst 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 [FG02] and the references therein for some exact methods for this task; these are based on the variable elimination algorithm, which we discuss in Section **??**. Unfortunately, for reasons we explain in Section **??**,

exact methods can be computationally intractable if the number of individuals and/or loci is large. See [ALK06] 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** [SH03].

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:

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

$$\mathcal{E}(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{\theta}) \triangleq -\sum_{d=1}^{D}\sum_{k=1}^{K} x_d z_k W_{dk} - \sum_{d=1}^{D} x_d b_d - \sum_{k=1}^{K} z_k c_k \tag{4.9}$$

$$= -(\boldsymbol{x}^\mathsf{T}\mathbf{W}\boldsymbol{z} + \boldsymbol{x}^\mathsf{T}\boldsymbol{b} + \boldsymbol{z}^\mathsf{T}\boldsymbol{c}) \tag{4.10}$$

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

where $\mathcal{E}$ is the energy function, $\mathbf{W}$ is a $D \times K$ weight matrix, $\boldsymbol{b}$ are the visible bias terms, $\boldsymbol{c}$ are the hidden bias terms, and $\boldsymbol{\theta} = (\mathbf{W}, \boldsymbol{b}, \boldsymbol{c})$ 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 $\boldsymbol{w}_{0,:} = \boldsymbol{c}$ and $\boldsymbol{w}_{:,0} = \boldsymbol{b}$. Note that naively computing $Z(\boldsymbol{\theta})$ takes $O(2^D 2^K)$ time but we can reduce this to $O(\min\{D2^K, K2^D\})$ time using the structure of the graph.

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

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

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

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

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

$$\mathbb{E}\left[\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{\theta}\right] = \sigma(\mathbf{W}^\mathsf{T}\boldsymbol{x}) \tag{4.14}$$

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

The weights in $\mathbf{W}$ are called the **generative weights**, since they are used to generate the observations, and the weights in $\mathbf{W}^\mathsf{T}$ 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 $\boldsymbol{x}$ "looks like" the weight vector $\boldsymbol{w}_{:,k}$ (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 $\boldsymbol{x}$ 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 (see Figure **??**). 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 [SMH07; SH10]:

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

The full conditionals are given by

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

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

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:

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

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 Section 2.3.3):

$$\mathcal{N}_c(\boldsymbol{x}|\boldsymbol{\eta}, \boldsymbol{\Lambda}) \propto \exp(\boldsymbol{\eta}^\mathsf{T}\boldsymbol{x} - \frac{1}{2}\boldsymbol{x}^\mathsf{T}\boldsymbol{\Lambda}\boldsymbol{x}) \tag{4.20}$$

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

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

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

More powerful models, which make the (co)variance depend on the hidden states, can also be developed [RH10].

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 (**??**). Interestingly, this is mathematically equivalent to the standard directed version [MM01].

If we use Gaussian latent variables and categorical observed variables, we get an undirected version of categorical PCA (Section **??**). In [SMH07], 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.

Chapter 5

Information theory

Chapter 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., [BWL19] uses proximal methods for training quantized DNNs, [Yao+20] uses proximal methods for efficient neural architecture search, [Sch+17; WHT19] uses proximal methods for policy gradient optimization, etc.).

Our presentation is based in part on the tutorial in [PB+14]. For another good review, see [PSW15].

6.1.1 Proximal operators

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

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

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

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

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.

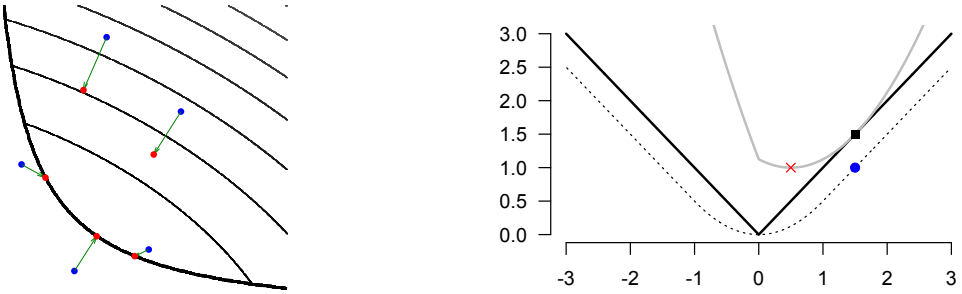
For example, suppose f is the indicator function for the convex set $\mathcal{C}$, i.e.,

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

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

$$\text{proj}_{\mathcal{C}}(\boldsymbol{x}) = \underset{\boldsymbol{z} \in \mathcal{C}}{\text{argmin}} \, ||\boldsymbol{z} - \boldsymbol{x}||_2 \tag{6.4}$$

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



(a) *(b)*

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 [PB+14]. 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 [PSW15]. Used with kind permission of Nicholas Polson.

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

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

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

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

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:

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

where $\mathbf{A}$ is a psd matrix.

6.1.1.1 Moreau envelope

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

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

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

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 $z^*(x_0) = \text{argmin}_z\, f_{x_0}^1(z) = 0.5$, 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.

Now let us evaluate the approximation at this proximal point:

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

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

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

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

6.1.1.2 Prox operator on a linear approximation yields gradient update

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

$$\hat{f}(\boldsymbol{x}) = f(\boldsymbol{x}_t) + \boldsymbol{g}_t^\mathsf{T}(\boldsymbol{x} - \boldsymbol{x}_t) \tag{6.10}$$

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

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

Solving $\nabla_{\boldsymbol{z}} \hat{f}^\eta_{\boldsymbol{x}}(\boldsymbol{z}) = \boldsymbol{0}$ yields the standard gradient update:

$$\text{prox}_{\eta \hat{f}}(\boldsymbol{x}) = \boldsymbol{x} - \eta \boldsymbol{g}_t \tag{6.12}$$

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

6.1.1.3 Prox operator on a quadratic approximation yields regularized Newton update

Now suppose we use a second order approximation at $\boldsymbol{x}_t$:

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

The prox operator for this is

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

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

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

Hence the gradient of the Moreau envelope is given by

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

Thus we can rewrite the prox operator as

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

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 [PB+14; PSW15].

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

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

where f^* is the convex conjugate of f (see Section 6.4).

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

$$\mathcal{B} = \{\boldsymbol{x} : ||\boldsymbol{x}||^* \leq 1\} \tag{6.19}$$

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

$$||\boldsymbol{z}||^* = \sup\{\boldsymbol{z}^\mathsf{T}\boldsymbol{x} : ||\boldsymbol{x}|| \leq 1\} \tag{6.20}$$

Hence

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

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 $\mathcal{C} = \{\boldsymbol{x} : \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}\}$ 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 to constrain values along that dimension.) The projection operator is easy to compute elementwise by simply thresholding at the boundaries:

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

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

$$\text{proj}_{\mathcal{C}}(\boldsymbol{x}) = \boldsymbol{x}_+ = [\max(x_1, 0), \ldots, \max(x_D, 0)] \tag{6.23}$$

6.1.2.3 ℓ_1 norm

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

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

One can show that the solution to this is given by

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

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

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

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

6.1.2.4 ℓ_2 norm

Now consider the ℓ_2 norm $f(\boldsymbol{x}) = ||\boldsymbol{x}||_2 = \sqrt{\sum_{d=1}^{D} x_d^2}$. 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:

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

Hence by the Moreau decomposition we have

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

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), $f(\boldsymbol{x}) = \frac{1}{2}||\boldsymbol{x}||_2^2 = \frac{1}{2}\sum_{d=1}^{D} x_d^2$. One can show that

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

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

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

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

A special case of this is if f is affine, $f(\boldsymbol{x}) = \boldsymbol{b}^\mathsf{T}\boldsymbol{x} + c$. Then we have $\text{prox}_{\lambda f}(\boldsymbol{x}) = \boldsymbol{x} - \lambda\boldsymbol{b}$. 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 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

$$\text{prox}_{\lambda f}(\mathbf{A}) = \sum_i (\sigma_i - \lambda)_+ \boldsymbol{u}_i \boldsymbol{v}_i^\mathsf{T} \tag{6.31}$$

where $\mathbf{A} = \sum_i \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$ 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

$$\text{proj}_{\mathcal{C}}(\mathbf{A}) = \sum_i (\lambda_i)_+ \boldsymbol{u}_i \boldsymbol{u}_i^\mathsf{T} \tag{6.32}$$

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

6.1.2.8 Projection onto probability simplex

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

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

The value $\nu \in \mathbb{R}$ must be found using bisection search. See [PB+14, p.183] 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:

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

where we assume $\mathcal{L} : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ 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 Stochastic and incremental PPM

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

$$\boldsymbol{\theta}_{t+1} = \text{prox}_{\eta_t \ell_t}(\boldsymbol{\theta}_t) = \underset{\boldsymbol{\theta}}{\text{argmin}}\, \ell_t(\boldsymbol{\theta}) + \frac{1}{2\eta_t}||\boldsymbol{\theta} - \boldsymbol{\theta}_t||_2^2 \tag{6.35}$$

where $\ell_t(\boldsymbol{\theta}) = \ell(\boldsymbol{\theta}, \boldsymbol{z}_t)$ and $\boldsymbol{z}_t \sim q$. The resulting method is known as **stochastic PPM** (see e.g., [PN18]). If q is the empirical distribution associated with a finite-sum objective, this is called the **incremental proximal point method** [Ber15]. It is often more stable than SGD.

In the case where the cost function is a linear least squares problem, one can show [AEM18] that the IPPM is equivalent to the Kalman filter (**??**), 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 (Section 8.2.2).

6.1.3.2 SGD 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

$$\hat{\ell}_t(\boldsymbol{\theta}) = \ell_t(\boldsymbol{\theta}_t) + \boldsymbol{g}_t^{\mathsf{T}}(\boldsymbol{\theta} - \boldsymbol{\theta}_t) \tag{6.36}$$

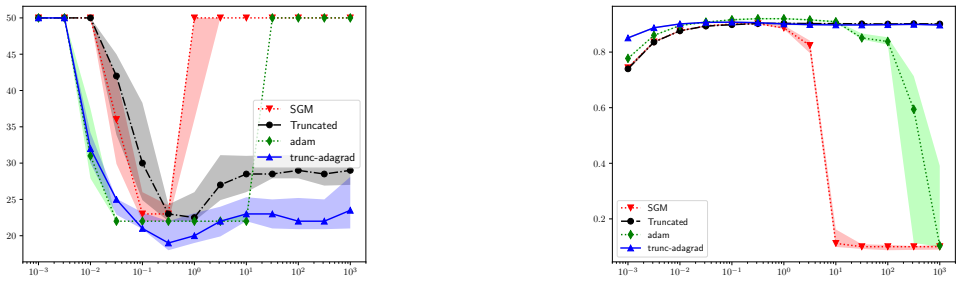
where $\boldsymbol{g}_t = \nabla_{\boldsymbol{\theta}} \ell_t(\boldsymbol{\theta}_t)$. Now we compute a proximal update to this approximate objective:

$$\boldsymbol{\theta}_{t+1} = \text{prox}_{\eta_t \hat{\ell}_t}(\boldsymbol{\theta}_t) = \underset{\boldsymbol{\theta}}{\text{argmin}}\, \hat{\ell}_t(\boldsymbol{\theta}) + \frac{1}{2\eta_t}||\boldsymbol{\theta} - \boldsymbol{\theta}_t||_2^2 \tag{6.37}$$

We have

$$\nabla_{\boldsymbol{\theta}} \left[\ell_t(\boldsymbol{\theta}_t) + \boldsymbol{g}_t^{\mathsf{T}}(\boldsymbol{\theta} - \boldsymbol{\theta}_t) + \frac{1}{2\eta_t}||\boldsymbol{\theta} - \boldsymbol{\theta}_t||_2^2 \right] = \boldsymbol{g}_t + \frac{1}{\eta_t}(\boldsymbol{\theta} - \boldsymbol{\theta}_t) \tag{6.38}$$

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



(a) *(b)*

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 satisifes $\mathcal{L}(\boldsymbol{\theta}_t) - \mathcal{L}(\boldsymbol{\theta}^*) \leq \epsilon$ 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 [AD19c]. Used with kind permission of Hilal Asi.*

6.1.3.3 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 [AD19b; AD19a; AD19c]. For example, suppose we know a lower bound on the loss, $\ell_t^{\min} = \min_{\boldsymbol{\theta}} \ell_t(\boldsymbol{\theta})$. For example, when using squared error, or cross-entropy loss for discrete labels, we have $\ell_t(\boldsymbol{\theta}) \geq 0$. Let us therefore define the **truncated model**

$$\hat{\ell}_t(\boldsymbol{\theta}) = \max\left(\ell_t(\boldsymbol{\theta}) + \boldsymbol{g}_t^{\mathsf{T}}(\boldsymbol{\theta} - \boldsymbol{\theta}_t), \ell_t^{\min}\right) \tag{6.39}$$

We can further improve things by replacing the Euclidean norm with a scaled Euclidean norm, where the diagonal scaling matrix is given by $\mathbf{A}_t = \text{diag}(\sum_{i=1}^t \boldsymbol{g}_i \boldsymbol{g}_i^{\mathsf{T}})^{\frac{1}{2}}$, as in AdaGrad [DHS11]. If $\ell_t^{\min} = 0$, the resulting proximal update becomes

$$\boldsymbol{\theta}_{t+1} = \underset{\boldsymbol{\theta}}{\text{argmin}} \left[\ell_t(\boldsymbol{\theta}_t) + \boldsymbol{g}_t^{\mathsf{T}}(\boldsymbol{\theta} - \boldsymbol{\theta}_t)\right]_+ + \frac{1}{2\eta_t}(\boldsymbol{\theta} - \boldsymbol{\theta}_t)^{\mathsf{T}} \mathbf{A}_t (\boldsymbol{\theta} - \boldsymbol{\theta}_t) \tag{6.40}$$

$$= \boldsymbol{\theta}_t - \min(\eta_t, \frac{\ell_t(\boldsymbol{\theta}_t)}{\boldsymbol{g}_t^{\mathsf{T}} \mathbf{A}_t^{-1} \boldsymbol{g}_t}) \boldsymbol{g}_t \tag{6.41}$$

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

[AD19c] 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 (**??**) 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 analysison the value of $\hat{\ell}_t(\boldsymbol{\theta}_{t+1})$, where $\hat{\ell}_t$ is the truncated linear model. If $\hat{\ell}_t(\boldsymbol{\theta}_{t+1}) > 0$, then setting the gradient to zero yields the usual SGD update, $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \boldsymbol{g}_t$. (We assume $\mathbf{A}_t = \mathbf{I}$ for simplicity.) Otherwise we must have $\hat{\ell}_t(\boldsymbol{\theta}_{t+1}) = 0$. But we know that $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda \boldsymbol{g}_t$ for some λ, so we solve $\hat{\ell}_t(\boldsymbol{\theta}_t - \lambda \boldsymbol{g}_t) = 0$ to get $\lambda = \hat{\ell}_t(\boldsymbol{\theta}_t)/||\boldsymbol{g}_t||_2^2$.

6.1.4 Proximal gradient method

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

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_s(\boldsymbol{\theta}) + \mathcal{L}_r(\boldsymbol{\theta}) \tag{6.42}$$

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 6.1.4.1).

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

$$\boldsymbol{\theta}_{t+1} = \text{prox}_{\eta_t \mathcal{L}_r}(\boldsymbol{\theta}_t - \eta_t \nabla \mathcal{L}_s(\boldsymbol{\theta}_t)) \tag{6.43}$$

If $\mathcal{L}_r = I_{\mathcal{C}}$, 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:

$$\tilde{\boldsymbol{\theta}}_{t+1} = \boldsymbol{\theta}_t + \beta_t(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}) \tag{6.44}$$

$$\boldsymbol{\theta}_{t+1} = \text{prox}_{\eta_t \mathcal{L}_r}(\tilde{\boldsymbol{\theta}}_{t+1} - \eta_t \nabla \mathcal{L}_s(\tilde{\boldsymbol{\theta}}_{t+1})) \tag{6.45}$$

See e.g., [Tse08].

Now we consider the stochastic case, where $\mathcal{L}_s(\boldsymbol{\theta}) = \mathbb{E}\left[\mathcal{L}_s(\boldsymbol{\theta}, \boldsymbol{z})\right]$. (We assume $\mathcal{L}_r$ is deterministic.) In this setting, we can use the following stochastic update:

$$\boldsymbol{\theta}_{t+1} = \text{prox}_{\eta_t \mathcal{L}_r}(\boldsymbol{\theta}_t - \eta_t \nabla \mathcal{L}_s(\boldsymbol{\theta}_t, \boldsymbol{z}_t)) \tag{6.46}$$

where $\boldsymbol{z}_t \sim q$. This is called the **stochastic proximal gradient method**. If q is the empirical distribution, this is called the **incremental proximal gradient method** [Ber15]. Both methods can also be accelerated (see e.g., [Nit14]).

If $\mathcal{L}_s$ is not convex, we can compute a locally convex approximation, as in Section 6.1.3.3. (We assume $\mathcal{L}_r$ remains convex.) The accelerated version of this is studied in [LL15]. In the stochastic case, we can similarly make a locally convex approximation to $\mathcal{L}_s(\boldsymbol{\theta}, \boldsymbol{z})$. This is studied in [Red+16; LL18]. An EKF interpretation in the incremental case (where $q = p_{\mathcal{D}}$) is given in [Aky+19].

6.1.4.1 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 (**??**). One way to implement this is to add the ℓ_1-norm of the parameters as a (non-smooth) penalty term, $\mathcal{L}_r(\boldsymbol{\theta}) = ||\boldsymbol{\theta}||_1 = \sum_{d=1}^{D} |\boldsymbol{\theta}_d|$. Thus the objective is

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_s(\boldsymbol{\theta}) + \mathcal{L}_r(\boldsymbol{\theta}) = \frac{1}{2}||\mathbf{X}\boldsymbol{\theta} - \boldsymbol{y}||_2^2 + \lambda||\boldsymbol{\theta}||_1 \tag{6.47}$$

The proximal gradient descent update can be written as

$$\boldsymbol{\theta}_{t+1} = \text{SoftThreshold}_{\eta_t \lambda}(\boldsymbol{\theta}_t - \eta_t \nabla \mathcal{L}_s(\boldsymbol{\theta}_t)) \tag{6.48}$$

where the soft thresholding operator (Equation (6.26)) is applied elementwise, and $\nabla \mathcal{L}_s(\boldsymbol{\theta}) = \mathbf{X}^{\mathsf{T}}(\mathbf{X}\boldsymbol{\theta} - \boldsymbol{y})$. This is called the **iterative soft thresholding algorithm** or **ISTA** [DDDM04; Don95]. If we combine this with Nesterov acceleration, we get the method known as "fast ISTA" or **FISTA** [BT09], which is widely used to fit sparse linear models.

6.1.5 Alternating direction method of multipliers (ADMM)

Consider the problem of optimizing $\mathcal{L}(\boldsymbol{x}) = \mathcal{L}_s(\boldsymbol{x}) + \mathcal{L}_r(\boldsymbol{x})$ where now both $\mathcal{L}_s$ and $\mathcal{L}_r$ may be non-smooth (but we asssume 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:

$$\text{minimize} \;\; \mathcal{L}_s(\boldsymbol{x}) + \mathcal{L}_r(\boldsymbol{z}) \;\; \text{s.t.} \;\; \boldsymbol{x} - \boldsymbol{z} = \boldsymbol{0} \tag{6.49}$$

This is called **consensus form**.

The corresponding **augmented Langragian** is given by

$$L_\rho(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y}) = \mathcal{L}_s(\boldsymbol{x}) + \mathcal{L}_r(\boldsymbol{z}) + \boldsymbol{y}^\mathsf{T}(\boldsymbol{x} - \boldsymbol{z}) + \frac{\rho}{2}||\boldsymbol{x} - \boldsymbol{z}||_2^2 \tag{6.50}$$

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

$$\boldsymbol{x}_{t+1} = \operatorname*{argmin}_{\boldsymbol{x}} L_\rho(\boldsymbol{x}, \boldsymbol{z}_t, \boldsymbol{y}_t) \tag{6.51}$$

$$\boldsymbol{z}_{t+1} = \operatorname*{argmin}_{\boldsymbol{z}} L_\rho(\boldsymbol{x}_{t+1}, \boldsymbol{z}, \boldsymbol{y}_t) \tag{6.52}$$

$$\boldsymbol{y}_{t+1} = \boldsymbol{y}_t + \rho(\boldsymbol{x}_{t+1} - \boldsymbol{z}_{t+1}) \tag{6.53}$$

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

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

$$\boldsymbol{x}_{t+1} = \operatorname*{argmin}_{\boldsymbol{x}} \left(\mathcal{L}_s(\boldsymbol{x}) + \boldsymbol{y}_t^\mathsf{T}\boldsymbol{x} + \frac{\rho}{2}||\boldsymbol{x} - \boldsymbol{z}_t||_2^2 \right) \tag{6.54}$$

$$\boldsymbol{z}_{t+1} = \operatorname*{argmin}_{\boldsymbol{z}} \left(\mathcal{L}_r(\boldsymbol{z}) - \boldsymbol{y}_t^\mathsf{T}\boldsymbol{z} + \frac{\rho}{2}||\boldsymbol{x}_{t+1} - \boldsymbol{z}||_2^2 \right) \tag{6.55}$$

If we combine the linear and quadratic terms, we get

$$\boldsymbol{x}_{t+1} = \operatorname*{argmin}_{\boldsymbol{x}} \left(\mathcal{L}_s(\boldsymbol{x}) + \frac{\rho}{2}||\boldsymbol{x} - \boldsymbol{z}_t + (1/\rho)\boldsymbol{y}_t||_2^2 \right) \tag{6.56}$$

$$\boldsymbol{z}_{t+1} = \operatorname*{argmin}_{\boldsymbol{z}} \left(\mathcal{L}_r(\boldsymbol{z}) + \frac{\rho}{2}||\boldsymbol{x}_{t+1} - \boldsymbol{z} - (1/\rho)\boldsymbol{y}_t||_2^2 \right) \tag{6.57}$$

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

$$\boldsymbol{x}_{t+1} = \text{prox}_{\lambda\mathcal{L}_s}(\boldsymbol{z}_t - \boldsymbol{u}_t) \tag{6.58}$$

$$\boldsymbol{z}_{t+1} = \text{prox}_{\lambda\mathcal{L}_r}(\boldsymbol{x}_{t+1} + \boldsymbol{u}_t) \tag{6.59}$$

$$\boldsymbol{u}_{t+1} = \boldsymbol{u}_t + \boldsymbol{x}_{t+1} - \boldsymbol{z}_{t+1} \tag{6.60}$$

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 [ZK14].

6.1.5.1 Example: robust PCA

In this section, we give an example of ADMM from [PB+14, Sec. 7.2].

Consider the following **matrix decomposition problem**:

$$\underset{\mathbf{X}_{1:J}}{\text{minimize}} \sum_{j=1}^{J} \gamma_j \phi_j(\mathbf{X}_j) \;\; \text{s.t.} \;\; \sum_{j=1}^{J} \mathbf{X}_j = \mathbf{A} \tag{6.61}$$

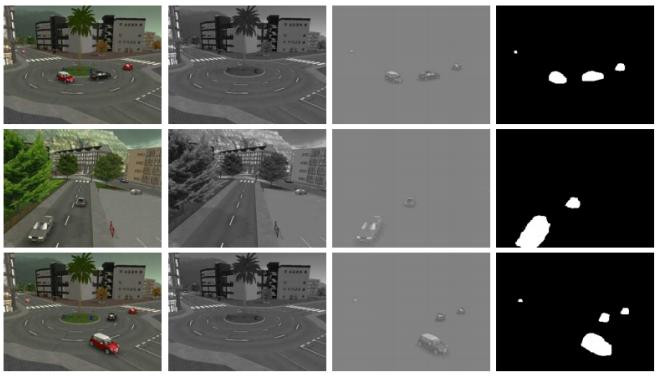


Figure 6.3: Robust PCA applied to some frames from a surveillance video. First column is input image. Second column is low-rank background model. Third model is sparse foreground model. Last column is derived foreground mask. From Figure 1 of [Bou+17]. Used with kind permission of Thierry Bouwmans.

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** [Can+11], 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., [Bou+17] 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:

$$\text{minimize} \;\; ||\mathbf{A} - (\mathbf{L} + \mathbf{S})||_F^2 + \gamma_L ||\mathbf{L}||_* + \gamma_S ||\mathbf{S}||_1 \tag{6.62}$$

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:

$$\phi_1(\mathbf{X}_1) = ||\mathbf{X}_1||_*, \;\; \phi_2(\mathbf{X}_2) = ||\mathbf{X}_2||_1, \;\; \phi_3(\mathbf{X}_3) = ||\mathbf{X}_3||_F^2 \tag{6.63}$$

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 $\mathcal{L}_r(\mathbf{X}) = I_{\mathcal{C}}(\mathbf{X})$, where $\mathbf{X} = (\mathbf{X}_1, \ldots, \mathbf{X}_J)$ and $\mathcal{C} = \{\mathbf{X}_{1:J} : \sum_{j=1}^J \mathbf{X}_j = \mathbf{A}\}$. The overall algorithm becomes

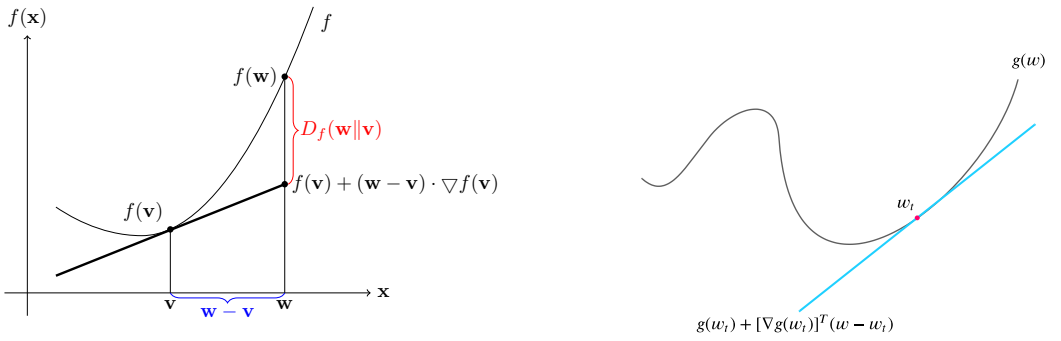
$$\mathbf{X}_{j,t+1} = \text{prox}_{\eta_t \phi_j}(\mathbf{X}_{j,t} - \overline{\mathbf{X}_t} + \frac{1}{N}\mathbf{A} - \mathbf{U}_t) \tag{6.64}$$

$$\mathbf{U}_{t+1} = \mathbf{U}_t + \overline{\mathbf{X}}_{t+1} - \frac{1}{J}\mathbf{A} \tag{6.65}$$

where $\overline{\mathbf{X}}$ is the elementwise average of $\mathbf{X}_1, \ldots, \mathbf{X}_J$. 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:

$$\text{proj}_{\mathcal{C}}(\mathbf{X}_1, \ldots, \mathbf{X}_J) = (\mathbf{X}_1, \ldots, \mathbf{X}_J) - \overline{\mathbf{X}} + \frac{1}{J}\mathbf{A} \tag{6.66}$$



(a) *(b)*

Figure 6.4: (a) Illustration of Bregman divergence. (b) A locally linear approximation to a non-convex function.

An alternative to using ℓ_1 minimization in the inner loop is to use hard thresholding [CGJ17]. 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 [Fat18; AS19].

6.2 Mirror descent

In this section, we discuss **mirror descent**, which is like gradient descent, but can leverage non-Euclidean geometry to potentially speed up convergence, or enforce certain constaints. But to explain the method, we first need to introduce some background concepts.

6.2.1 Bregman divergence

Let $f : \Omega \to \mathbb{R}$ be a continuously differentiable, strictly convex function defined on a closed convex set Ω. We define the **Bregman divergence** associated with f as follows [Bre67]:

$$D_f(\boldsymbol{w}, \boldsymbol{v}) = f(\boldsymbol{w}) - f(\boldsymbol{v}) - (\boldsymbol{w} - \boldsymbol{v})^{\mathsf{T}} \nabla f(\boldsymbol{v}) \tag{6.67}$$

To understand this, let

$$\hat{f}_{\boldsymbol{v}}(\boldsymbol{w}) = f(\boldsymbol{v}) + (\boldsymbol{w} - \boldsymbol{v})^{\mathsf{T}} \nabla f(\boldsymbol{v}) \tag{6.68}$$

be a first order Taylor series approximation to f centered at $\boldsymbol{v}$. Then the Bregman divergence is the difference from this linear approximation:

$$D_f(\boldsymbol{w}, \boldsymbol{v}) = f(\boldsymbol{w}) - \hat{f}_{\boldsymbol{v}}(\boldsymbol{w}) \tag{6.69}$$

See Figure 6.4a for an illustration. Since f is convex, we have $D_f(\boldsymbol{v}||\boldsymbol{w}) \geq 0$, since $\hat{f}_v$ is a linear lower bound on f.

Below we mention some important special cases of Bregman divergences.

- If $f(\boldsymbol{w}) = ||\boldsymbol{w}||^2$, then $D_f(\boldsymbol{w}, \boldsymbol{v}) = ||\boldsymbol{w} - \boldsymbol{v}||^2$ is the squared Euclidean distance.
- If $f(\boldsymbol{w}) = \boldsymbol{w}^{\mathsf{T}}\mathbf{Q}\boldsymbol{w}$, then $D_f(\boldsymbol{w}, \boldsymbol{v})$ is the squared Mahalanobis distance (Section 2.3.1).
- If $\boldsymbol{w}$ are the natural parameters of an exponential family distribution, amd $f(\boldsymbol{w}) = \log Z(\boldsymbol{w})$ is the log normalizer, then the Bregman divergence is the same as the Kullback Leibler divergence, as we show in **??**.

6.2.2 Proximal point method

Suppose we make a locally linear approximation to the objective at step t centered at $\boldsymbol{\theta}_t$:

$$\hat{\mathcal{L}}_t(\boldsymbol{\theta}) = \mathcal{L}_t(\boldsymbol{\theta}_t) + \boldsymbol{g}_t^\mathsf{T}(\boldsymbol{\theta} - \boldsymbol{\theta}_t) \tag{6.70}$$

where $\boldsymbol{g}_t = \nabla_{\boldsymbol{\theta}} \mathcal{L}_t(\boldsymbol{\theta}_t)$. This is shown in Figure 6.4b.

If we optimize this approximation using gradient descent, we may end up at $-\infty$. To prevent this, we can use a **proximal update**, which adds a quadratic penalty to ensure we don't move too far from where the locally linear approximation is valid:

$$\boldsymbol{\theta}_{t+1} = \text{prox}_{\eta_t \hat{\mathcal{L}}_t}(\boldsymbol{\theta}_t) \triangleq \underset{\boldsymbol{\theta}}{\text{argmin}}\, \hat{\mathcal{L}}_t(\boldsymbol{\theta}) + \frac{1}{2\eta_t}||\boldsymbol{\theta} - \boldsymbol{\theta}_t||_2^2 \tag{6.71}$$

We can solve this optimization problem by setting the gradient to 0:

$$\nabla_{\boldsymbol{\theta}} \left[\mathcal{L}_t(\boldsymbol{\theta}_t) + \boldsymbol{g}_t^\mathsf{T}(\boldsymbol{\theta} - \boldsymbol{\theta}_t) + \frac{1}{2\eta_t}||\boldsymbol{\theta} - \boldsymbol{\theta}_t||_2^2 \right] = \boldsymbol{g}_t + \frac{1}{\eta_t}(\boldsymbol{\theta} - \boldsymbol{\theta}_t) = \mathbf{0} \tag{6.72}$$

This yields the standard gradient descent update:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \boldsymbol{g}_t \tag{6.73}$$

However, by changing from Euclidean norm to another distance metric, we can derive mirror descent, as we show below.

6.2.3 PPM using Bregman divergence

Suppose we replace the Euclidean distance term $||\boldsymbol{\theta} - \boldsymbol{\theta}_t||_2^2$ by a more general Bregman divergence (Equation (6.67)),

$$D_h(\boldsymbol{x}, \boldsymbol{y}) = h(\boldsymbol{x}) - \left[h(\boldsymbol{y}) + \nabla h(\boldsymbol{y})^\mathsf{T}(\boldsymbol{x} - \boldsymbol{y}) \right] \tag{6.74}$$

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

$$\boldsymbol{\theta}_{t+1} = \underset{\boldsymbol{\theta}}{\text{argmin}}\, \hat{\mathcal{L}}(\boldsymbol{\theta}) + \frac{1}{\eta_t} D_h(\boldsymbol{\theta}, \boldsymbol{\theta}_t) \tag{6.75}$$

$$= \underset{\boldsymbol{\theta}}{\text{argmin}}\, \eta_t \boldsymbol{g}_t^\mathsf{T} \boldsymbol{\theta} + D_h(\boldsymbol{\theta}, \boldsymbol{\theta}_t) \tag{6.76}$$

This is known as **mirror descent** [NY83; BT03]. This can easily be extended to the stochastic setting in the obvious way.

One can show that natural gradient descent (**??**) is a form of mirror descent [RM15]. More precisely, mirror descent in the mean parameter space is equivalent to natural gradient descent in the canonical parameter space.

6.3 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.

6.3.1 Example: computing Fibonnaci numbers

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

$$F_i = F_{i-1} + F_{i-2} \tag{6.77}$$

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 Fibbonaci numbers is shown in Algorithm 1. Unfortunately, this takes exponential time. For example, evaluating fib(5) proceeds as follows:

$$\begin{aligned} F_5 &= F_4 + F_3 && (6.78)\\ &= (F_3 + F_2) + (F_2 + F_1) && (6.79)\\ &= ((F_2 + F_1) + (F_1 + F_0)) + ((F_1 + F_0) + F_1) && (6.80)\\ &= (((F_1 + F_0) + F_1) + (F_1 + F_0))((F_1 + F_0) + F_1) && (6.81) \end{aligned}$$

We see that there is a lot of repeated computation. For example, 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 2.

Algorithm 1: Fibbonaci numbers, top down

```
function fib(n);
if n = 0 or n = 1 then
    return 1
else
    return (fib(n − 1) + fib(n − 2))
```

Algorithm 2: Fibbonaci numbers, bottom up

```
function fib(n);
F_0 := 1, F_1 := 2 ;
for i = 2, . . . , n do
    F_i := F_{i−1} + F_{i−2}
return F_n
```

6.3.2 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 (**??**), the Viterbi algorithm for MAP sequence estimation in HMMs (**??**), inference in more general graphical models (**??**), reinforcement learning (**??**), etc.

6.4 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 [Bis06, Sec. 10.5].

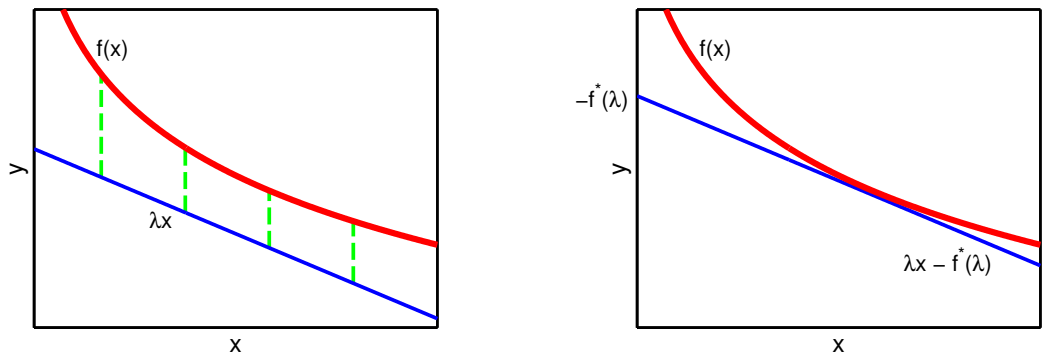


Figure 6.5: Illustration of a conjugate functon. Red line is original function $f(x)$, and the blue line is a linear lower bound λx. To make the bound tight, we find the x where $\nabla f(x)$ is parallel to λ, and slide the line up to touch there; the amount we slide up is given by $f^*(\lambda)$. Adapted from Figure 10.11 of [Bis06].

6.4.1 Introduction

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

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) \triangleq \boldsymbol{\lambda}^\mathsf{T} \boldsymbol{x} - f^*(\boldsymbol{\lambda}) \leq f(\boldsymbol{x}) \tag{6.82}$$

where $\boldsymbol{\lambda}$ is the slope, which we choose, and $f^*(\boldsymbol{\lambda})$ is the intercept, which we solve for below. See Figure 6.5(a) for an illustration.

For a fixed λ, we can find the point $\boldsymbol{x}_\lambda$ where the lower bound is tight by "sliding" the line upwards until it touches the curve at $\boldsymbol{x}_\lambda$, as shown in Figure 6.5(b). At $\boldsymbol{x}_\lambda$, we minimize the distance between the function and the lower bound:

$$\boldsymbol{x}_\lambda \triangleq \operatorname*{argmin}_{\boldsymbol{x}} f(x) - \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = \operatorname*{argmin}_{\boldsymbol{x}} f(x) - \boldsymbol{\lambda}^\mathsf{T} \boldsymbol{x} \tag{6.83}$$

Since the bound is tight at this point, we have

$$f(\boldsymbol{x}_\lambda) = \mathcal{L}(\boldsymbol{x}_\lambda, \boldsymbol{\lambda}) = \boldsymbol{\lambda}^\mathsf{T} \boldsymbol{x}_\lambda - f^*(\boldsymbol{\lambda}) \tag{6.84}$$

and hence

$$f^*(\boldsymbol{\lambda}) = \boldsymbol{\lambda}^\mathsf{T} \boldsymbol{x}_\lambda - f(\boldsymbol{x}_\lambda) = \max_{\boldsymbol{x}} \boldsymbol{\lambda}^\mathsf{T} \boldsymbol{x} - f(\boldsymbol{x}) \tag{6.85}$$

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 $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) \leq f(\boldsymbol{x})$, with equality at $\boldsymbol{x} = \boldsymbol{x}_\lambda$, for any function $f : \mathbb{R}^D \to \mathbb{R}$. For any given $\boldsymbol{x}$, we can optimize over $\boldsymbol{\lambda}$ to make the bound as tight as possible, giving us a fixed function $\mathcal{L}(\boldsymbol{x})$; this is called a **variational approximation**. We can then try to maximize this lower bound wrt $\boldsymbol{x}$ instead of maximizing $f(\boldsymbol{x})$. This method is used extensively in approximate Bayesian inference, as we discuss in **??**.

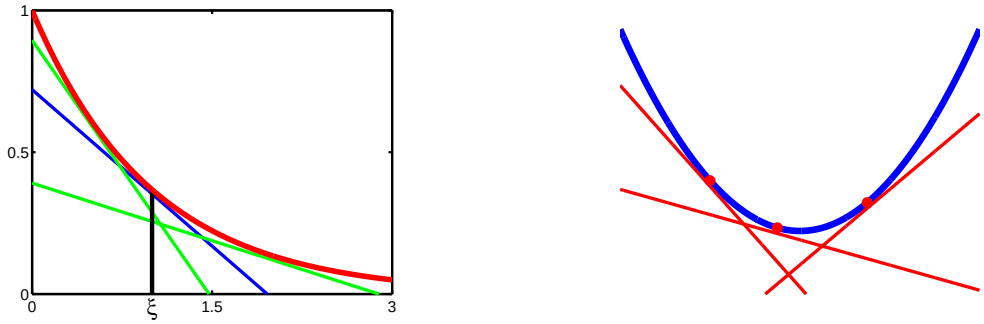
6.4.2 Example: exponential function

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

$$\mathcal{L}(x, \lambda) = \lambda x - f^\dagger(\lambda) \tag{6.86}$$

where the conjugate function is given by

$$f^\dagger(\lambda) = \max_x \lambda x - f(x) = -\lambda \log(-\lambda) + \lambda \tag{6.87}$$



(a) *(b)*

Figure 6.6: (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 [Bis06]. Generated by opt_lower_bound.py. (b) For a convex function $f(x)$, its epipgraph 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 [JJ99].

as illustrated in Figure 6.6(a).

To see this, define

$$J(x, \lambda) = \lambda x - f(x) \tag{6.88}$$

We have

$$\frac{\partial J}{\partial x} = \lambda x - f'(x) = \lambda + e^{-x} \tag{6.89}$$

Setting the derivative to zero gives

$$x_\lambda = \arg\max_x J(x, \lambda) = -\log(-\lambda) \tag{6.90}$$

Hence

$$f^\dagger(\lambda) = J(x_\lambda, \lambda) = \lambda(-\log(-\lambda)) - e^{\log(-\lambda)} = -\lambda \log(-\lambda) + \lambda \tag{6.91}$$

6.4.3 Conjugate of a conjugate

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

$$f^{**}(\boldsymbol{x}) = \max_{\boldsymbol{\lambda}} \boldsymbol{\lambda}^\mathsf{T} \boldsymbol{x} - f^\dagger(\boldsymbol{\lambda}) \tag{6.92}$$

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

$$f^{**}(\boldsymbol{x}) = \max_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) \leq f(\boldsymbol{x}) \tag{6.93}$$

Since we are free to modify $\boldsymbol{\lambda}$ for each $\boldsymbol{x}$, we can make the lower bound tight at each $\boldsymbol{x}$. This perfectly characterizes f, since the epigraph of a convex function is an intersection of half-planes defined by linear lower bounds, as shown in Figure 6.6(b).

Let us demonstrate this using the example from Section 6.4.2. We have

$$f^{**}(x) = \max_\lambda \lambda x - f^\dagger(\lambda) = \max_\lambda \lambda x + \lambda \log(-\lambda) - \lambda \tag{6.94}$$

Define

$$J^*(x, \lambda) = \lambda x - f^\dagger(x) = \lambda x + \lambda \log(-\lambda) - \lambda \tag{6.95}$$

We have

$$\frac{\partial}{\partial \lambda} J^*(x,\lambda) = x + \log(-\lambda) + \lambda\left(\frac{-1}{-\lambda}\right) - 1 = 0 \tag{6.96}$$

$$x = -\log(-\lambda) \tag{6.97}$$

$$\lambda_x = -e^{-x} \tag{6.98}$$

Substituting back we find

$$f^{**}(x) = J^*(x,\lambda_x) = (-e^{-x})x + (-e^{-x})(-x) - (-e^{-x}) = e^{-x} = f(x) \tag{6.99}$$

6.4.4 Bounds for the logistic (sigmoid) function

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

6.4.4.1 Exponential upper bound

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

$$f(x) = \min_\eta \eta x - f^\dagger(\eta) \tag{6.100}$$

where

$$f^\dagger(\eta) = \min_x \eta x - f(x) \tag{6.101}$$

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

$$f^\dagger(\eta) = -\eta \ln \eta - (1-\eta)\ln(1-\eta) \tag{6.102}$$

which is the binary entropy function. Hence

$$\log \sigma(x) \leq \eta x - f^\dagger(\eta) \tag{6.103}$$

$$\sigma(x) \leq \exp(\eta x - f^\dagger(\eta)) \tag{6.104}$$

This exponential upper bound on $\sigma(x)$ is illustrated in Figure 6.7(a).

6.4.4.2 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 [JJ96].

First we write

$$\log \sigma(x) = -\log(+e^{-x}) = -\log\left(e^{-x/2}(e^{x/2} + e^{-x/2})\right) \tag{6.105}$$

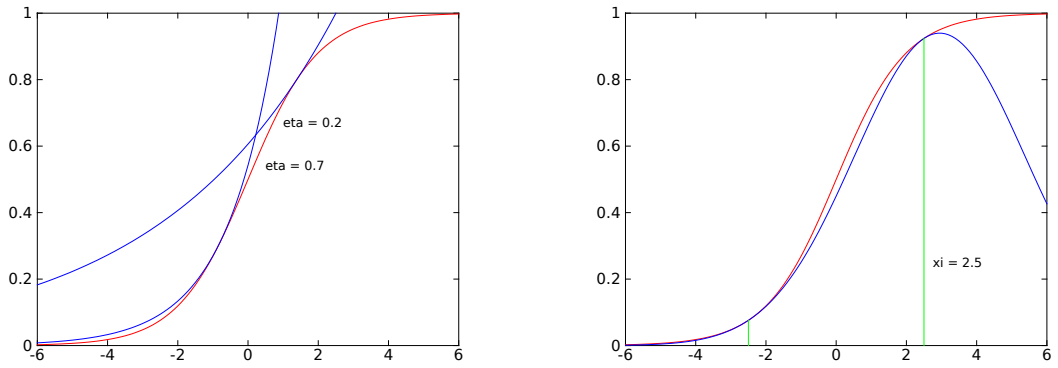
$$= x/2 - \log(e^{x/2} + e^{-x/2}) \tag{6.106}$$

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

$$f^\dagger(\eta) = \max_{x^2} \eta x^2 - f(\sqrt{x^2}) \tag{6.107}$$

We have

$$0 = \eta - \frac{dx}{dx^2}\frac{d}{dx} f(x) = \eta + \frac{1}{4x}\tanh(\frac{x}{2}) \tag{6.108}$$



(a) *(b)*

Figure 6.7: Illustration of (a) exponental upper bound and (b) quadratic lower bound to the sigmoid function. Generated by sigmoid_upper_bounds.py and sigmoid_lower_bounds.py.

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

$$\eta = -\frac{1}{4\xi}\tanh(\frac{\xi}{2}) = -\frac{1}{2\xi}\left[\sigma(\xi) - \frac{1}{2}\right] = -\lambda(\xi) \tag{6.109}$$

The conjugate function can be rewritten as

$$f^\dagger(\lambda(\xi)) = -\lambda(\xi)\xi^2 - f(\xi) = \lambda(\xi)\xi^2 + \log(e^{\xi/2} + e^{-\xi/2}) \tag{6.110}$$

So the lower bound on f becomes

$$f(x) \geq -\lambda(\xi)x^2 - g(\lambda(\xi)) = -\lambda(\xi)x^2 + \lambda(\xi)\xi^2 - \log(e^{\xi/2} + e^{-\xi/2}) \tag{6.111}$$

and the lower bound on the sigmoid function becomes

$$\sigma(x) \geq \sigma(\xi)\exp\left[(x-\xi)/2 - \lambda(\xi)(x^2 - \xi^2)\right] \tag{6.112}$$

This is illustrated in Figure 6.7(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 14.1.1 for an example, where we use this bound for Bayesian logistic regression.

Part II

Inference

Chapter 7

Inference algorithms: an overview

Chapter 8

Message passing inference

8.1 Kalman filtering variants

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

8.1.0.1 Handling unknown observation noise

In the case of scalar observations (as often arises in time series forecasting), we can extend the Kalman filter to handle the common situation in which the observation noise variance $V = \sigma^2$ is unknown, as described in [WH97, Sec 4.6]. The model is defined as follows:

$$p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) = \mathcal{N}(\boldsymbol{z}_t|\mathbf{A}_t\boldsymbol{z}_{t-1}, V\mathbf{Q}_t^*) \tag{8.1}$$

$$p(y_t|\boldsymbol{z}_t) = \mathcal{N}(y_t|\boldsymbol{h}_t^\mathsf{T}\boldsymbol{z}_t, V) \tag{8.2}$$

where $\mathbf{Q}_t^*$ is the unscaled system noise, and we define $\mathbf{C}_t = \boldsymbol{h}_t^\mathsf{T}$ to be the vector that maps the hidden state vector to the scalar observation. Let $\lambda = 1/V$ be the observation precision. To start the algorithm, we use the following prior:

$$p_0(\lambda) = \mathrm{Ga}(\frac{\nu_0}{2}, \frac{\nu_0\tau_0}{2}) \tag{8.3}$$

$$p_0(\boldsymbol{z}|\lambda) = \mathcal{N}(\boldsymbol{\mu}_0, V\boldsymbol{\Sigma}_0^*) \tag{8.4}$$

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

$$\mathcal{N}(\boldsymbol{z}_{t-1}, \lambda|\mathcal{D}_{1:t-1}) = \mathcal{N}(\boldsymbol{z}_{t-1}|\boldsymbol{\mu}_{t-1}, V\boldsymbol{\Sigma}_{t-1}^*)\mathrm{Ga}(\lambda|\frac{\nu_{t-1}}{2}, \frac{\nu_{t-1}\tau_{t-1}}{2}) \tag{8.5}$$

The posterior is given by

$$\mathcal{N}(\boldsymbol{z}_t, \lambda|\mathcal{D}_{1:t}) = \mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_t, V\boldsymbol{\Sigma}_t^*)\mathrm{Ga}(\lambda|\frac{\nu_t}{2}, \frac{\nu_t\tau_t}{2}) \tag{8.6}$$

where

$$\boldsymbol{\mu}_{t|t-1} = \mathbf{A}_t \boldsymbol{\mu}_{t-1} \tag{8.7}$$
$$\boldsymbol{\Sigma}^*_{t|t-1} = \mathbf{A}_t \boldsymbol{\Sigma}^*_{t-1} \mathbf{A}_t + \mathbf{Q}^*_t \tag{8.8}$$
$$e_t = y_t - \boldsymbol{h}_t^\mathsf{T} \boldsymbol{\mu}_{t|t-1} \tag{8.9}$$
$$s^*_t = \boldsymbol{h}_t^\mathsf{T} \boldsymbol{\Sigma}^*_{t|t-1} \boldsymbol{h}_t + 1 \tag{8.10}$$
$$\boldsymbol{k}_t = \boldsymbol{\Sigma}^*_{t|t-1} \boldsymbol{h}_t / s^*_t \tag{8.11}$$
$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + \boldsymbol{k}_t e_t \tag{8.12}$$
$$\boldsymbol{\Sigma}^*_t = \boldsymbol{\Sigma}^*_{t|t-1} - \boldsymbol{k}_t \boldsymbol{k}_t^\mathsf{T} s^*_t \tag{8.13}$$
$$\nu_t = \nu_{t-1} + 1 \tag{8.14}$$
$$\nu_t \tau_t = \nu_{t-1} \tau_{t-1} + e_t^2 / s^*_t \tag{8.15}$$

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

$$p(\boldsymbol{z}_t | \mathcal{D}_{1:t}) = \mathcal{T}_{\nu_t}(\boldsymbol{z}_t | \boldsymbol{\mu}_t, \tau_t \boldsymbol{\Sigma}^*_t) \tag{8.16}$$

8.2 Inference based on local linearization

In this section, we extend the Kalman filter and smoother to the case where the system dynamics and/or the observation model are nonlinear. However, we continue to assume Gaussian noise distributions. Thus we assume the model has the following form:

$$\boldsymbol{z}_t = \boldsymbol{f}_t(\boldsymbol{z}_{t-1}) + \mathcal{N}(\mathbf{0}, \mathbf{Q}_t) \tag{8.17}$$
$$\boldsymbol{y}_t = \boldsymbol{h}_t(\boldsymbol{z}_t) + \mathcal{N}(\mathbf{0}, \mathbf{R}_t) \tag{8.18}$$

where $\boldsymbol{z}_t \in \mathbb{R}^{N_z}$ is the hidden state, $\boldsymbol{y}_t \in \mathbb{R}^{N_y}$ is the observation, $\boldsymbol{f}_t : \mathbb{R}^{N_z} \to \mathbb{R}^{N_z}$ is the dynamics model, and $\boldsymbol{h}_t : \mathbb{R}^{N_z} \to \mathbb{R}^{N_y}$ is the observation model.

Exact posterior inference in this model family is generally computationally intractable, so in this section, we create a locally linear approximation to the model, following the presentation of [Sar13, Ch. 5]. For other reviews of algorithms for approximate inference in nonlinear state space models, see e.g., [Fan+17; Li+17; Koy+10].

8.2.1 Taylor series expansion

Suppose $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{z})$, where $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$ is a differentiable and invertible function. The pdf for $\boldsymbol{y}$ is given by

$$p(\boldsymbol{y}) = |\det \mathbf{J}(\boldsymbol{y})| \mathcal{N}(\boldsymbol{f}^{-1}(\boldsymbol{y}) | \boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{8.19}$$

where $\mathbf{J}$ is the Jacobian of $\boldsymbol{f}^{-1}$ evaluated at $\boldsymbol{y}$:

$$[\mathbf{J}]_{jj'} = \frac{\partial f_j^{-1}(\boldsymbol{u})}{\partial u_{j'}} |_{\boldsymbol{u}=\boldsymbol{y}} \tag{8.20}$$

In general this is intractable to compute, so we seek an approximation.

Suppose $\boldsymbol{z} = \boldsymbol{\mu} + \delta \boldsymbol{z}$, where $\boldsymbol{\delta z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$. Then we can form a Taylor series expansion of the function $\boldsymbol{f}$ as follows:

$$\boldsymbol{f}(\boldsymbol{z}) = \boldsymbol{f}(\boldsymbol{\mu} + \delta \boldsymbol{z}) \approx \boldsymbol{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu}) \delta \boldsymbol{z} + \sum_i \frac{1}{2} \delta \boldsymbol{z}^\mathsf{T} \mathbf{F}^i(\boldsymbol{\mu}) \, \delta \boldsymbol{z} \, \boldsymbol{e}_i + \cdots \tag{8.21}$$

where $\boldsymbol{e}_i = (0, \cdots, 1, 0, \cdots, 0)$ is the i'th unit vector, $\mathbf{F}(\boldsymbol{\mu})$ is the Jacobian of $\boldsymbol{f}$,

$$[\mathbf{F}(\boldsymbol{\mu})]_{jj'} = \frac{\partial f_j(\boldsymbol{z})}{\partial x_{j'}} |_{\boldsymbol{z}=\boldsymbol{\mu}} \tag{8.22}$$

and $\mathbf{F}^i(\boldsymbol{\mu})$ is the Hessian of $f_i(\cdot)$ computed at $\boldsymbol{\mu}$:

$$[\mathbf{F}^i(\boldsymbol{\mu})]_{jj'} = \frac{\partial^2 f_i(\boldsymbol{z})}{\partial x_j \partial x_{j'}}|_{\boldsymbol{z}=\boldsymbol{\mu}} \tag{8.23}$$

We can create a linear approximation by just using the first two terms:

$$\hat{\boldsymbol{f}}(\boldsymbol{z}) = \boldsymbol{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\delta\boldsymbol{z} \tag{8.24}$$

We now derive the induced Gaussian approximation to $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{z})$. The mean is given by

$$\begin{aligned}
\mathbb{E}[\boldsymbol{y}] &\approx \mathbb{E}[\boldsymbol{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\delta\boldsymbol{z}] && (8.25)\\
&= \boldsymbol{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\mathbb{E}[\delta\boldsymbol{z}] && (8.26)\\
&= \boldsymbol{f}(\boldsymbol{\mu}) && (8.27)
\end{aligned}$$

The covariance is given by

$$\begin{aligned}
\mathrm{Cov}[\boldsymbol{y}] &= \mathbb{E}\left[(\boldsymbol{f}(\boldsymbol{z}) - \mathbb{E}[\boldsymbol{f}(\boldsymbol{z})])(\boldsymbol{f}(\boldsymbol{z}) - \mathbb{E}[\boldsymbol{f}(\boldsymbol{z})])^\mathsf{T}\right] && (8.28)\\
&\approx \mathbb{E}\left[(\boldsymbol{f}(\boldsymbol{z}) - \boldsymbol{f}(\boldsymbol{\mu}))(\boldsymbol{f}(\boldsymbol{z}) - \boldsymbol{f}(\boldsymbol{\mu}))^\mathsf{T}\right] && (8.29)\\
&\approx \mathbb{E}\left[(\boldsymbol{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\delta\boldsymbol{z} - \boldsymbol{f}(\boldsymbol{\mu}))(\boldsymbol{f}(\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\delta\boldsymbol{z} - \boldsymbol{f}(\boldsymbol{\mu}))^\mathsf{T}\right] && (8.30)\\
&= \mathbb{E}\left[(\mathbf{F}(\boldsymbol{\mu})\delta\boldsymbol{z})(\mathbf{F}(\boldsymbol{\mu})\delta\boldsymbol{z})^\mathsf{T}\right] && (8.31)\\
&= \mathbf{F}(\boldsymbol{\mu})\mathbb{E}\left[\delta\boldsymbol{z}\delta\boldsymbol{z}^\mathsf{T}\right]\mathbf{F}(\boldsymbol{\mu})^\mathsf{T} && (8.32)\\
&= \mathbf{F}(\boldsymbol{\mu})\,\boldsymbol{\Sigma}\,\mathbf{F}(\boldsymbol{\mu})^\mathsf{T} && (8.33)
\end{aligned}$$

Let us consider a 1d example. In Figure 8.1a, we show what happens when we pass a Gaussian distribution $p(x)$, shown on the bottom right, through a nonlinear function $y = f(x)$, shown on the top right. The resulting distribution (approximated by Monte Carlo) is shown in the shaded gray area in the top left corner. The best Gaussian approximation to this, computed from $\mathbb{E}[f(x)]$ and $\mathbb{V}[f(x)]$ by Monte Carlo, is shown by the solid black line. The EKF approximates this Gaussian as follows: it linearizes the f function at the current mode, μ, and then passes the Gaussian distribution $p(x)$ through this linearized function. In this example, the result is quite a good approximation to the first and second moments of $p(y)$, for much less cost than an MC approximation.

We often also need to compute a Gaussian approximation to the joint distribution $p(\boldsymbol{z}, \boldsymbol{y})$. We can compute this in the same way, by defining the augmented function $\tilde{\boldsymbol{f}}(\boldsymbol{z}) = [\boldsymbol{z}, \boldsymbol{f}(\boldsymbol{z})]$. This gives

$$\mathbb{E}\left[\tilde{\boldsymbol{f}}(\boldsymbol{z})\right] \approx \begin{pmatrix}\boldsymbol{\mu}\\ \boldsymbol{f}(\boldsymbol{\mu})\end{pmatrix} \tag{8.34}$$

$$\mathrm{Cov}\left[\tilde{\boldsymbol{f}}(\boldsymbol{z})\right] \approx \begin{pmatrix}\mathbf{I}\\ \mathbf{F}(\boldsymbol{\mu})\end{pmatrix}\boldsymbol{\Sigma}\begin{pmatrix}\mathbf{I}\\ \mathbf{F}(\boldsymbol{\mu})\end{pmatrix}^\mathsf{T} = \begin{pmatrix}\boldsymbol{\Sigma} & \boldsymbol{\Sigma}\mathbf{F}(\boldsymbol{\mu})^\mathsf{T}\\ \mathbf{F}(\boldsymbol{\mu})\boldsymbol{\Sigma} & \mathbf{F}(\boldsymbol{\mu})\boldsymbol{\Sigma}\mathbf{F}(\boldsymbol{\mu})^\mathsf{T}\end{pmatrix} \tag{8.35}$$

When deriving the EKF, we need the following slightly more general version:

$$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}),\ \boldsymbol{y} = \boldsymbol{f}(\boldsymbol{z}) + \boldsymbol{q},\ \boldsymbol{q} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}) \tag{8.36}$$

The resulting linear approximation to the joint is

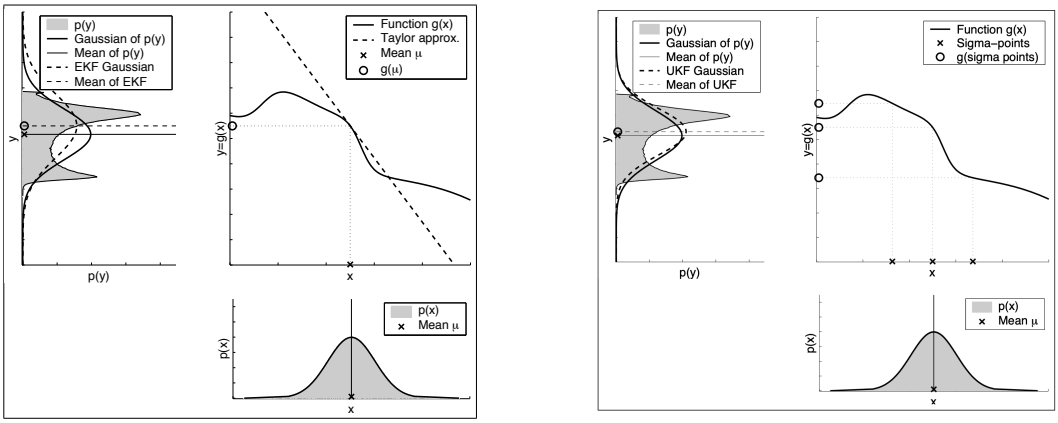
$$\begin{pmatrix}\boldsymbol{z}\\ \boldsymbol{y}\end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix}\boldsymbol{\mu}\\ \boldsymbol{\mu}_L\end{pmatrix}, \begin{pmatrix}\boldsymbol{\Sigma} & \mathbf{C}_L\\ \mathbf{C}_L^\mathsf{T} & \mathbf{S}_L\end{pmatrix}\right) \tag{8.37}$$

$$\boldsymbol{\mu}_L = \boldsymbol{f}(\boldsymbol{\mu}) \tag{8.38}$$

$$\mathbf{S}_L = \mathbf{F}(\boldsymbol{\mu})\boldsymbol{\Sigma}\mathbf{F}(\boldsymbol{\mu})^\mathsf{T} + \mathbf{Q} \tag{8.39}$$

$$\mathbf{C}_L = \boldsymbol{\Sigma}\mathbf{F}(\boldsymbol{\mu})^\mathsf{T} \tag{8.40}$$

It is also possible to derive an approximation for the case of non-additive Gaussian noise, where $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{z}, \boldsymbol{q})$. See [Sar13, Sec 5.1] for details.



(a) *(b)*

Figure 8.1: Nonlinear transformation of a Gaussian random variable. The prior $p(x)$ is shown on the bottom right. The function $y = f(x)$ is shown on the top right. The transformed distribution $p(y)$ is shown in the top left. A linear function induces a Gaussian distribution, but a non-linear function induces a complex distribution. (a) The solid line is the best Gaussian approximation to this. The dotted line is the EKF approximation to this. From Figure 3.4 of [TBF06]. (b) The dotted line is the UKF approximation to this. From Figure 3.7 of [TBF06]. Used with kind permission of Sebastian Thrun.

8.2.2 The extended Kalman filter (EKF)

In this section, we discuss the **extended Kalman filter** or **EKF**, which can compute a Gaussian approximation to the posterior even when the model has nonlinear dynamics and/or observations. The basic idea is to linearize the dynamics and observation models about the previous state estimate using a first order Taylor series expansion, as in Section 8.2.1, and then to apply the standard Kalman filter equations from **??**. Thus we approximate a stationary non-linear dynamical system with a non-stationary linear dynamical system. (However, the noise variance terms, $\mathbf{Q}$ and $\mathbf{R}$, are not changed, i.e., the additional error due to linearization is not modeled.)

8.2.2.1 Algorithm

The EKF linearizes the model at each step by computing the following Jacobian matrices:

$$\mathbf{F}_t = \frac{\partial \boldsymbol{f}_t(\boldsymbol{z})}{\partial \boldsymbol{z}}|_{\boldsymbol{\mu}_{t-1}} \tag{8.41}$$

$$\mathbf{H}_t = \frac{\partial \boldsymbol{h}_t(\boldsymbol{z})}{\partial \boldsymbol{z}}|_{\boldsymbol{\mu}_{t|t-1}} \tag{8.42}$$

The updates then become

$$\boldsymbol{\mu}_{t|t-1} = \boldsymbol{f}(\boldsymbol{\mu}_{t-1}) \tag{8.43}$$

$$\boldsymbol{\Sigma}_{t|t-1} = \mathbf{F}_t \boldsymbol{\Sigma}_{t-1} \mathbf{F}_t + \mathbf{Q}_t \tag{8.44}$$

$$\boldsymbol{e}_t = \boldsymbol{y}_t - \boldsymbol{h}(\boldsymbol{\mu}_{t|t-1}) \tag{8.45}$$

$$\mathbf{S}_t = \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^{\mathsf{T}} + \mathbf{R}_t \tag{8.46}$$

$$\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t \mathbf{S}_t^{-1} \tag{8.47}$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t \boldsymbol{e}_t \tag{8.48}$$

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} = \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t \mathbf{S}_t \mathbf{K}_t^{\mathsf{T}} \tag{8.49}$$

8.2.2.2 Derivation

The derivation of the EKF is similar to the derivation of the Kalman filter (**??**), except we also need to apply the linear approximation from Section 8.2.1.

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

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

$$\boldsymbol{m}' = \begin{pmatrix} \boldsymbol{\mu}_{t-1} \\ \boldsymbol{f}(\boldsymbol{\mu}_{t-1}) \end{pmatrix} \tag{8.51}$$

$$\boldsymbol{\Sigma}' = \begin{pmatrix} \boldsymbol{\Sigma}_{t-1} & \boldsymbol{\Sigma}_{t-1}\mathbf{F}_t^\mathsf{T} \\ \mathbf{F}_t\boldsymbol{\Sigma}_{t-1} & \mathbf{F}_t\boldsymbol{\Sigma}_{t-1}\mathbf{F}_t^\mathsf{T} + \mathbf{Q}_{t-1} \end{pmatrix} \tag{8.52}$$

where $\mathbf{F}_t$ is the Jacobian of $\boldsymbol{f}$ evaluated at $\boldsymbol{\mu}_{t-1}$. From this we can derive the marginal $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1})$, which gives us the predict step.

For the update step, we first consider a Gaussian approximation to $p(\boldsymbol{z}_t, \boldsymbol{y}_t|\boldsymbol{y}_{1:t-1})$ as follows:

$$p(\boldsymbol{z}_t, \boldsymbol{y}_t | \boldsymbol{y}_{1:t-1}) \approx \mathcal{N}(\begin{pmatrix} \boldsymbol{z}_t \\ \boldsymbol{y}_t \end{pmatrix} | \boldsymbol{m}'', \boldsymbol{\Sigma}'') \tag{8.53}$$

$$\boldsymbol{m}'' = \begin{pmatrix} \boldsymbol{\mu}_{t|t-1} \\ \boldsymbol{h}(\boldsymbol{\mu}_{t|t-1}) \end{pmatrix} \tag{8.54}$$

$$\boldsymbol{\Sigma}'' = \begin{pmatrix} \boldsymbol{\Sigma}_{t|t-1} & \boldsymbol{\Sigma}_{t|t-1}\mathbf{H}_t^\mathsf{T} \\ \mathbf{H}_t\boldsymbol{\Sigma}_{t|t-1} & \mathbf{H}_t\boldsymbol{\Sigma}_{t|t-1}\mathbf{H}_t^\mathsf{T} + \mathbf{R}_{t-1} \end{pmatrix} \tag{8.55}$$

where $\mathbf{H}_t$ is the Jacobian of $\boldsymbol{h}$ evaluated at $\boldsymbol{\mu}_{t|t-1}$.

Finally, we use Equation (2.52) to get the posterior

$$p(\boldsymbol{z}_t | \boldsymbol{y}_t, \boldsymbol{y}_{1:t-1}) \approx \mathcal{N}(\boldsymbol{z}_t | \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t) \tag{8.56}$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \boldsymbol{\Sigma}_{t|t-1}\mathbf{H}_t^\mathsf{T}(\mathbf{H}_t\boldsymbol{\Sigma}_{t|t-1}\mathbf{H}^\mathsf{T} + \mathbf{R}_t)^{-1}[\boldsymbol{y}_t - \boldsymbol{h}(\boldsymbol{\mu}_{t|t-1})] \tag{8.57}$$

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t|t-1} - \boldsymbol{\Sigma}_{t|t-1}\mathbf{H}_t^\mathsf{T}(\mathbf{H}_t\boldsymbol{\Sigma}_{t|t-1}\mathbf{H}^\mathsf{T} + \mathbf{R}_t)^{-1}\mathbf{H}_t\boldsymbol{\Sigma}_{t|t-1} \tag{8.58}$$

This gives us the update step.

See **??** for an example.

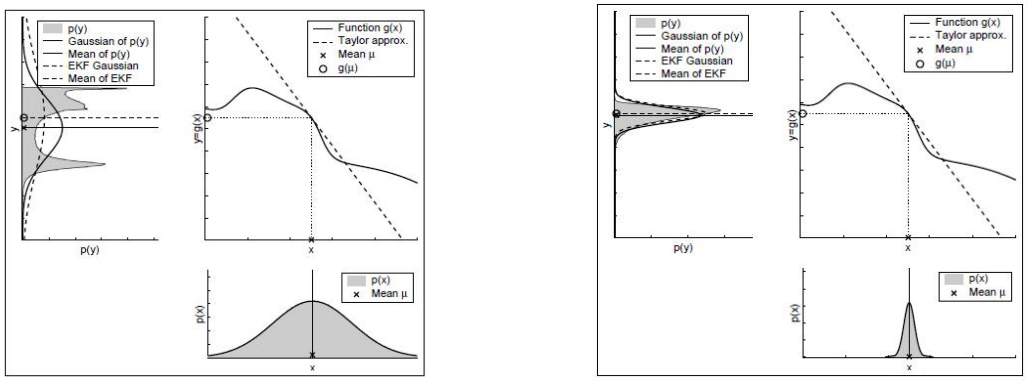
8.2.2.3 Accuracy

There are two cases when the EKF works poorly. The first is when the prior covariance is large. In this case, the prior distribution is broad, so we end up sending a lot of probability mass through different parts of the function that are far from $\boldsymbol{\mu}_{t-1|t-1}$, where the function has been linearized. See Figure 8.2 for an illustration. The other setting where the EKF works poorly is when the function is highly nonlinear near the current mean. See Figure 8.3 for an illustration.

In Section 8.3.2, we will discuss an algorithm called the UKF which works better than the EKF in both of these settings. An altenrative approach is to use a second-order Taylor series approximation. The resulting updates can still be computed in closed form (see [Sar13, Sec 5.2] for details). We can further improve performance by repeatedly re-linearizing the equations around $\boldsymbol{\mu}_t$ instead of $\boldsymbol{\mu}_{t|t-1}$; this is called the **iterated EKF**.

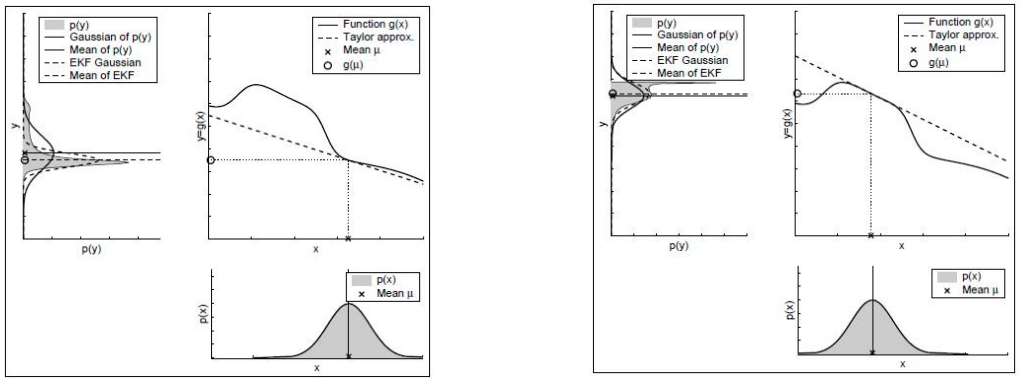
8.2.2.4 Diagonal approximation

The cost of the EKF is $O(N_y N_z^2)$, which can be prohibitive for large state spaces. In such cases, a natural approximation is to use a block diagonal approximation. Let us define the following Jacobian matrices for



(a) (b)

Figure 8.2: Illustration of the fact that a broad prior (a) may result in a more complex posterior than a narrow prior (b). Consequently, the EKF approximation may work poorly in situations of high uncertainty. From Figure 3.5 of [TBF06]. Used with kind permission of Dieter Fox.



(a) (b)

Figure 8.3: Illustration of the fact that if the function function is very nonlinear (a) at the current operating point, the posterior will be less well approximated by the EKF than if the function is locally linear (b). From Figure 3.6 of [TBF06]. Used with kind permission of Dieter Fox.

block i:

$$\mathbf{F}_t^i = \frac{\partial \boldsymbol{f}_t^i(\boldsymbol{z})}{\partial \boldsymbol{z}}|_{\boldsymbol{\mu}_{t-1}} \tag{8.59}$$

$$\mathbf{H}_t^i = \frac{\partial \boldsymbol{h}_t^i(\boldsymbol{z})}{\partial \boldsymbol{z}}|_{\boldsymbol{\mu}_{t|t-1}} \tag{8.60}$$

We then compute the following updates for each block:

$$\boldsymbol{\mu}_{t|t-1}^i = \boldsymbol{f}^i(\boldsymbol{\mu}_{t-1}) \tag{8.61}$$

$$\boldsymbol{\Sigma}_{t|t-1}^i = (\mathbf{F}_t^i)^\mathsf{T} \boldsymbol{\Sigma}_{t-1}^i \mathbf{F}_t^i + \mathbf{Q}_{t-1}^i \tag{8.62}$$

$$\mathbf{S}_t = \sum_i (\mathbf{H}_t^i)^\mathsf{T} \boldsymbol{\Sigma}_{t|t-1}^i \mathbf{H}_t^i + \mathbf{R}_t \tag{8.63}$$

$$\mathbf{K}_t^i = \boldsymbol{\Sigma}_{t|t-1}^i \mathbf{H}_t^i \mathbf{S}_t^{-1} \tag{8.64}$$

$$\boldsymbol{\mu}_t^i = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t^i \boldsymbol{e}_t \tag{8.65}$$

$$\boldsymbol{\Sigma}_t^i = \boldsymbol{\Sigma}_{t|t-1}^i - \mathbf{K}_t^i \mathbf{H}_t^i \boldsymbol{\Sigma}_{t|t-1}^i \tag{8.66}$$

8.2.3 The extended Kalman smoother

We can extend the EKF to the offline smoothing case as follows (see e.g., [Sar13, Sec 9.1] for the derivation):

$$p(\boldsymbol{z}_t|\boldsymbol{y}_{1:T}) = \mathcal{N}(\boldsymbol{\mu}_{t|T}, \boldsymbol{\Sigma}_{t|T}) \tag{8.67}$$

$$\boldsymbol{\mu}_{t|T} = \boldsymbol{\mu}_{t|t} + \mathbf{G}_t(\boldsymbol{\mu}_{t+1|T} - \boldsymbol{\mu}_{t+1|t}) \tag{8.68}$$

$$\boldsymbol{\Sigma}_{t|T} = \boldsymbol{\Sigma}_{t|t} + \mathbf{G}_t(\boldsymbol{\Sigma}_{t+1|T} - \boldsymbol{\Sigma}_{t+1|t})\mathbf{G}_t^\mathsf{T} \tag{8.69}$$

$$\mathbf{G}_t = \boldsymbol{\Sigma}_{t|t}\mathbf{F}(\boldsymbol{\mu}_t)^\mathsf{T} \boldsymbol{\Sigma}_{t+1|t}^{-1} \tag{8.70}$$

The only difference from the RTS smoother in **??** is that we replace the fixed $\mathbf{F}$ matrix with the Jacobian $\mathbf{F}(\boldsymbol{\mu}_t)$.

8.2.4 Exponential-family EKF

In this section, we present an extension of the EKF to the case where the observation model is in the exponential family, as proposed in [Oll18]. We call this the **Exponential family EKF** or **EEKF**. This allows us to apply the EKF for online parameter estimation of classification models, as we illustrate in Section 8.2.4.3.

8.2.4.1 Modeling assumptions

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

$$\boldsymbol{z}_t = f(\boldsymbol{z}_{t-1}, \boldsymbol{u}_t) + \mathcal{N}(\mathbf{0}, \mathbf{Q}_t) \tag{8.71}$$

We assume the observation model is

$$p(\boldsymbol{y}_t|\boldsymbol{z}_t) = \text{Expfam}(\boldsymbol{y}_t|\hat{\boldsymbol{y}}_t) \tag{8.72}$$

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

$$\hat{\boldsymbol{y}}_t = h(\boldsymbol{z}_t, \boldsymbol{u}_t) \tag{8.73}$$

The standard EKF corresponds to the special case of a Gaussian output with fixed observation covariance $\mathbf{R}_t$, with $\hat{\boldsymbol{y}}_t$ being the mean.

8.2.4.2 Algorithm

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

$$\boldsymbol{\mu}_{t|t-1} = f(\boldsymbol{\mu}_{t-1}, \boldsymbol{u}_t) \tag{8.74}$$

$$\mathbf{F}_t = \frac{\partial f}{\partial \boldsymbol{z}}|_{(\boldsymbol{\mu}_{t-1}, \boldsymbol{u}_t)} \tag{8.75}$$

$$\boldsymbol{\Sigma}_{t|t-1} = \mathbf{F}_t \boldsymbol{\Sigma}_{t-1} \mathbf{F}_t^\mathsf{T} + \mathbf{Q}_t \tag{8.76}$$

$$\hat{\boldsymbol{y}}_t = h(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{u}_t) \tag{8.77}$$

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

$$\boldsymbol{e}_t = \mathcal{T}(\boldsymbol{y}_t) - \hat{\boldsymbol{y}}_t \tag{8.78}$$

$$\mathbf{R}_t = \mathrm{Cov}\left[\mathcal{T}(\boldsymbol{y})|\hat{\boldsymbol{y}}_t\right] \tag{8.79}$$

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

Finally we perform the udpate:

$$\mathbf{H}_t = \frac{\partial h}{\partial \boldsymbol{z}}|_{(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{u}_t)} \tag{8.80}$$

$$\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\mathsf{T} (\mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^\mathsf{T} + \mathbf{R}_t)^{-1} \tag{8.81}$$

$$\boldsymbol{\Sigma}_t = (\mathbf{I} - \mathbf{K}_t \mathbf{H}_t) \boldsymbol{\Sigma}_{t|t-1} \tag{8.82}$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t \boldsymbol{e}_t \tag{8.83}$$

In [Oll18], they show that this is equivalent to an online version of natural gradient descent (**??**).

8.2.4.3 EEKF for training logistic regression

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

$$\mathcal{T}(\boldsymbol{y}) = \left[\mathbb{I}\left(y = 1\right), \ldots, \mathbb{I}\left(y = C - 1\right)\right] \tag{8.84}$$

be the $(C-1)$-dimensional vector of sufficient statistics, and let $\hat{\boldsymbol{y}} = [p_1, \ldots, p_{C-1}]$ 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

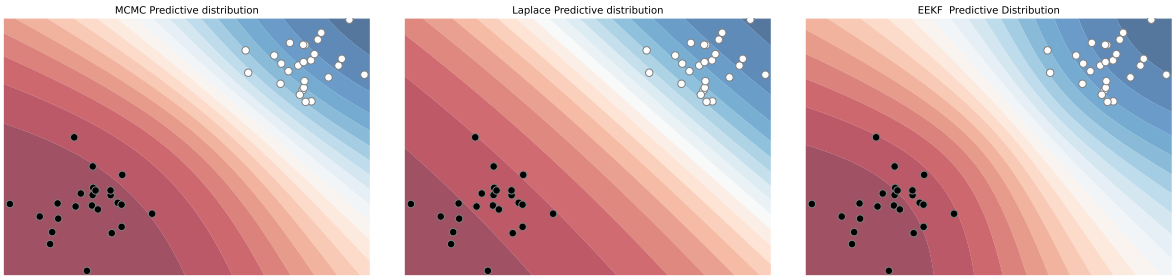
$$R_{ij} = \mathrm{diag}(p_i) - p_i p_j \tag{8.85}$$

Now consider the simpler case where we have two class labels, so $C = 2$. In this case, $\mathcal{T}(\boldsymbol{y}) = \mathbb{I}\left(\boldsymbol{y} = 1\right)$, and $\hat{\boldsymbol{y}} = p(\boldsymbol{y} = 1) = p$. 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:

$$p(y_t|\boldsymbol{z}_t, \boldsymbol{u}_t) = \mathrm{Ber}(y_t|\sigma(\boldsymbol{z}_t^\mathsf{T} \boldsymbol{u}_t)) \tag{8.86}$$

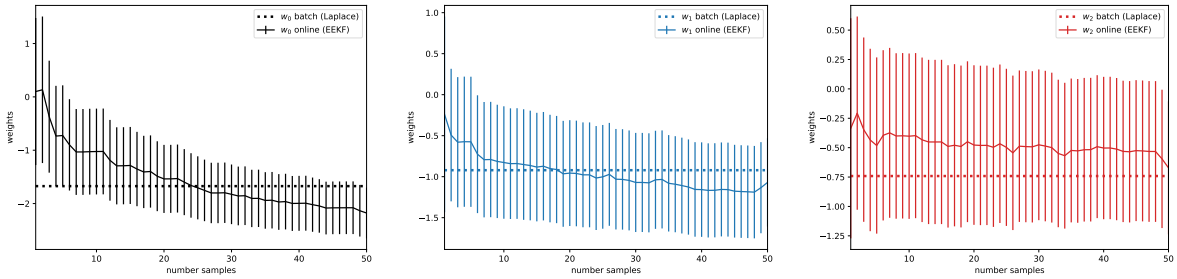
We can use the exponential family representation of the output discussed in Section 8.2.4.3.

We assume the parameters $\boldsymbol{z}_t$ are static, so $\mathbf{Q}_t = \mathbf{0}$. The 2d data is shown in Figure 8.4a. We sequentially compute the posterior using the EEKF, and compare to the offline estimate computed using a Laplace approximation (where the MAP estimate is computed using BFGS) and an MCMC approximation, which we take as "ground truth". In Figure 8.4c, we see that the resulting posterior predictive distributions are similar. Finally, in Figure 8.5, we visualize how the posterior marginals converge over time. (See also **??**, where we solve this same problem using ADF.)



(a) (b) (c)

Figure 8.4: Bayesian inference applied to a 2d binary logistic regression problem, $p(y = 1|\boldsymbol{x}) = \sigma(w_0 + w_1 x_1 + w_2 x_2)$. We show the training data and the posterior predictive produced by different methods. (a) Offline MCMC approximation. (b) Offline Laplace approximation. (c) Online EEKF approximation at the final step of inference. Generated by eekf_logistic_regression.py.



(a) (b) (c)

Figure 8.5: Marginal posteriors over time for the EEKF method. The horizontal line is the offline MAP estimate. Generated by eekf_logistic_regression.py.

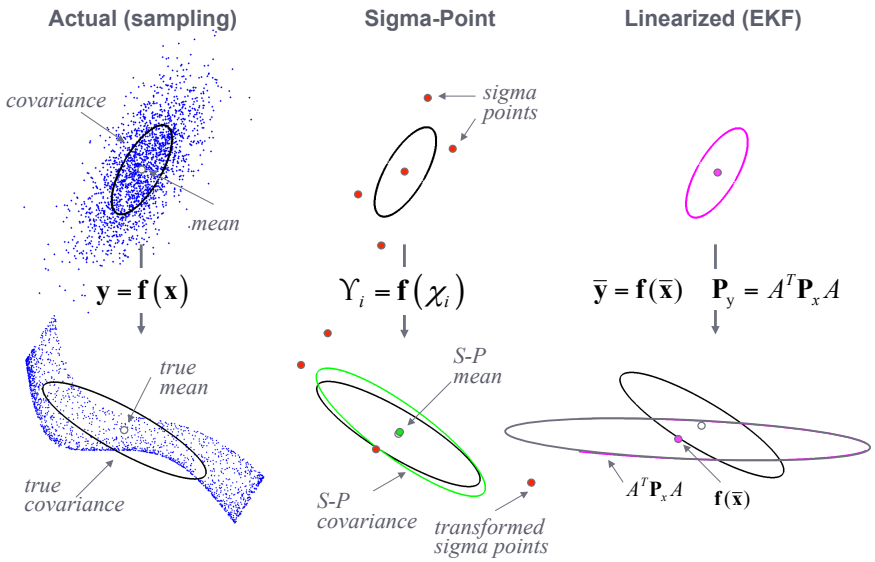


Figure 8.6: An example of the unscented transform in two dimensions. From [WM01]. Used with kind permission of Eric Wan.

8.3 Inference based on the unscented transform

In this section, we replace the local linearization of the model with a different approximation known as the **unscented transform**. When applied to Bayesian filtering, we get the **unscented Kalman filter** (**UKF**), since it is a version of the EKF that "doesn't stink" [JU97]. The key intuition is this: it is easier to compute a Gaussian approximation to a distribution than to approximate a function. So instead of computing a linear approximation to the function and then passing a Gaussian through it, we instead pass a deterministically chosen set of points, known as **sigma points**, through the function, and fit a Gaussian to the resulting transformed points. See Section 8.3.1 for details.

The UKF has the advantage over EKF of not needing to compute Jacobians of the observation and dynamics model, but has the disadvantage that can be slower, since it requires d evaluations of the dynamics and observation models. In addition, it has 3 hyper-parameters that need to be set. We give more details below. For even more information, see e.g., [RHG16]. For connections to the EKF, see [GH12]. For an application to distance estimation from bluetooth sensors on mobile phones see [Lov+20]. (This latter application was part of the UK COVID-19 contact risk score estimation and contact tracing app; see [BCH20; MKS21] for details.)

8.3.1 The unscented transform

Suppose we have two random variables $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{z})$, where $\boldsymbol{z} \in \mathbb{R}^d$ and $\boldsymbol{y} \in \mathbb{R}^d$. The unscented transform forms a Gaussian approximation to $p(\boldsymbol{y})$ as follows.

1. For a set of $2d + 1$ sigma points as follows:

$$\boldsymbol{y}_0 = \boldsymbol{\mu} \tag{8.87}$$

$$\boldsymbol{y}_i = \boldsymbol{\mu} + \sqrt{d + \lambda}\, [\sqrt{\boldsymbol{\Sigma}}]_{:i} \tag{8.88}$$

$$\boldsymbol{y}_{i+n} = \boldsymbol{\mu} - \sqrt{d + \lambda}\, [\sqrt{\boldsymbol{\Sigma}}]_{:i} \tag{8.89}$$

where notation $\mathbf{M}_{:i}$ means the i'th column of matrix $\mathbf{M}$, $\sqrt{\boldsymbol{\Sigma}}$ is the matrix square root, and λ is a scaling parameter given by

$$\lambda = \alpha^2(d + \kappa) - d \tag{8.90}$$

2. Propagate the sigma points through the nonlinear function to get the following $2d+1$ outputs:

$$\boldsymbol{y}_i = \boldsymbol{f}(\boldsymbol{y}_i) \tag{8.91}$$

3. Estimate the mean and covariance of the resulting bag of points:

$$\mathbb{E}\left[\boldsymbol{f}(\boldsymbol{z})\right] \approx \boldsymbol{\mu}_U = \sum_{i=0}^{2d} w_i^m \boldsymbol{y}_i \tag{8.92}$$

$$\mathrm{Cov}\left[\boldsymbol{f}(\boldsymbol{z})\right] \approx \mathbf{S}_U = \sum_{i=0}^{2d} w_i^c (\boldsymbol{y}_i - \boldsymbol{\mu}_U)(\boldsymbol{y}_i - \boldsymbol{\mu}_U)^\mathsf{T} \tag{8.93}$$

where the w's are weighting terms, given by the following:

$$w_0^m = \frac{\lambda}{d+\lambda} \tag{8.94}$$

$$w_0^c = \frac{\lambda}{d+\lambda} + (1 - \alpha^2 + \beta) \tag{8.95}$$

$$w_i^m = w_i^c = \frac{1}{2(d+\lambda)} \tag{8.96}$$

In general, the optimal values of α, β and κ are problem dependent. A typical recommendation is $\alpha = 10^{-3}$, $\kappa = 1$, $\beta = 2$ [Bit16]. See Figure 8.1b for a 1d illustration and Figure 8.6 for a 2d illustration.

Now suppose we want to approximate the joint distribution $p(\boldsymbol{z}, \boldsymbol{y})$, where $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{z}) + \boldsymbol{q}$, and $\boldsymbol{q} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$. We have

$$\begin{pmatrix} \boldsymbol{z} \\ \boldsymbol{y} \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_U \end{pmatrix}, \begin{pmatrix} \boldsymbol{\Sigma} & \mathbf{C}_U \\ \mathbf{C}_U^\mathsf{T} & \mathbf{S}_u \end{pmatrix} \right) \tag{8.97}$$

where

$$\boldsymbol{\mu}_U = \sum_{i=0}^{2d} w_i^m \boldsymbol{y}_i \tag{8.98}$$

$$\mathbf{S}_U = \sum_{i=0}^{2d} w_i^c (\boldsymbol{y}_i - \boldsymbol{\mu}_U)(\boldsymbol{y}_i - \boldsymbol{\mu}_U)^\mathsf{T} + \mathbf{Q} \tag{8.99}$$

$$\mathbf{C}_U = \sum_{i=0}^{2d} w_i^c (\boldsymbol{y}_i - \boldsymbol{\mu})(\boldsymbol{y}_i - \boldsymbol{\mu}_U)^\mathsf{T} \tag{8.100}$$

The unscented transform is a third-order method in the sense that the mean of $\boldsymbol{y}$ is exact for polynomials up to order 3. However the covariance is only exact for linear functions.

8.3.2 The unscented Kalman filter (UKF)

The UKF uses the unscented transform twice, once to approximate passing through the system model $\boldsymbol{f}$, and once to approximate passing through the measurement model $\boldsymbol{h}$. The derivation is analogous to that of the EKF. The resulting algorithm is as follows.

8.3.2.1 Prediction step

We perform these steps.

1. Form the sigma points

$$\boldsymbol{z}_{t-1,0} = \boldsymbol{\mu}_{t-1} \tag{8.101}$$
$$\boldsymbol{z}_{t-1,i} = \boldsymbol{\mu}_{t-1} + \sqrt{d+\lambda}\,[\sqrt{\boldsymbol{\Sigma}_{t-1}}]_{:i} \tag{8.102}$$
$$\boldsymbol{z}_{t-1,i+n} = \boldsymbol{\mu}_{t-1} - \sqrt{d+\lambda}\,[\sqrt{\boldsymbol{\Sigma}_{t-1}}]_{:i} \tag{8.103}$$

2. Propagate these points through the dynamics model:

$$\hat{\boldsymbol{z}}_{t,i} = \boldsymbol{f}(\boldsymbol{z}_{t-1,i}) \tag{8.104}$$

3. Compute the predicted mean and covariance

$$\boldsymbol{\mu}_{t|t-1} = \sum_{i=0}^{2d} w_i^m \hat{\boldsymbol{z}}_{t,i} \tag{8.105}$$
$$\boldsymbol{\Sigma}_{t|t-1} = \sum_{i=0}^{2d} w_i^c (\hat{\boldsymbol{z}}_{t,i} - \boldsymbol{\mu}_{t|t-1})(\hat{\boldsymbol{z}}_{t,i} - \boldsymbol{\mu}_{t|t-1})^\mathsf{T} + \mathbf{Q}_t \tag{8.106}$$

8.3.2.2 Update step

We perform these steps.

1. Form the sigma points

$$\boldsymbol{z}_{t,0} = \boldsymbol{\mu}_{t|t-1} \tag{8.107}$$
$$\boldsymbol{z}_{t,i} = \boldsymbol{\mu}_{t|t-1} + \sqrt{d+\lambda}\,[\sqrt{\boldsymbol{\Sigma}_{t|t-1}}]_{:i} \tag{8.108}$$
$$\boldsymbol{z}_{t,i+n} = \boldsymbol{\mu}_{t|t-1} - \sqrt{d+\lambda}\,[\sqrt{\boldsymbol{\Sigma}_{t|t-1}}]_{:i} \tag{8.109}$$

2. Propagate these points through the measurement model:

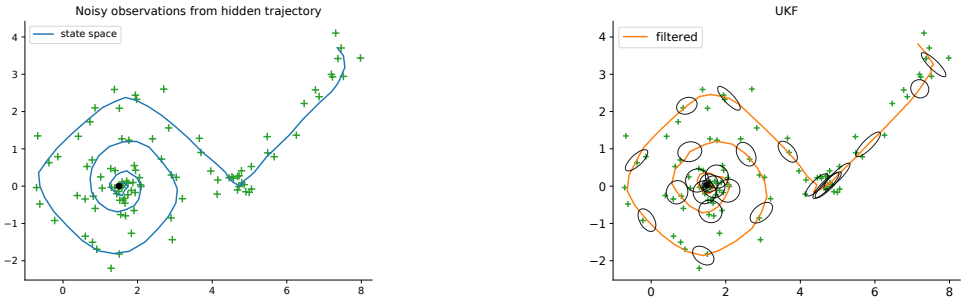
$$\hat{\boldsymbol{z}}_{t,i} = \boldsymbol{f}(\boldsymbol{z}_{t,i}) \tag{8.110}$$

3. Compute the predicted mean and covariance of $p(\boldsymbol{y}_t, \boldsymbol{z}_t|\boldsymbol{y}_{1:t-1})$:

$$\boldsymbol{m}_t = \sum_{i=0}^{2d} w_i^m \hat{\boldsymbol{z}}_{t,i} \tag{8.111}$$
$$\mathbf{S}_t = \sum_{i=0}^{2d} w_i^c (\hat{\boldsymbol{z}}_{t,i} - \boldsymbol{m}_t)(\hat{\boldsymbol{z}}_{t,i} - \boldsymbol{m}_t)^\mathsf{T} + \mathbf{R}_t \tag{8.112}$$
$$\mathbf{C}_t = \sum_{i=0}^{2d} w_i^c (\hat{\boldsymbol{z}}_{t,i} - \boldsymbol{\mu}_{t|t-1})(\hat{\boldsymbol{z}}_{t,i} - \boldsymbol{m}_t)^\mathsf{T} + \mathbf{R}_t \tag{8.113}$$

4. Apply Bayes rule for Gaussians to get the posterior:

$$\mathbf{K}_t = \mathbf{C}_t \mathbf{S}_t^{-1} \tag{8.114}$$
$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t(\boldsymbol{y}_t - \boldsymbol{m}_t) \tag{8.115}$$
$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t \mathbf{S}_t \mathbf{K}_t^\mathsf{T} \tag{8.116}$$



(a) (b)

Figure 8.7: Illustration of UKF applied to a 2d nonlinear dynamical system. (a) True underlying state and observed data. (b) UKF estimate. Generated by ekf_vs_ukf.py.

8.3.2.3 Example: noisy 2d tracking problem

Let us revisit the 2d nonlinear tracking problem from **??**. In Figure 8.7b, we see that the UKF algorithm (with $\alpha = 1$, $\beta = 0$, $\kappa = 2$) works well on this problem.

8.3.3 The unscented Kalman smoother

The unscented Kalman smoother is a simple modification of the usual Kalman smoothing step, and is given by the following:

$$p(\boldsymbol{z}_t|\boldsymbol{y}_{1:T}) = \mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_{t|T}, \boldsymbol{\Sigma}_{t|T}) \tag{8.117}$$

$$\boldsymbol{\mu}_{t|T} = \boldsymbol{\mu}_{t|t} + \mathbf{G}_t(\boldsymbol{\mu}_{t+1|T} - \boldsymbol{\mu}_{t+1|t}) \tag{8.118}$$

$$\boldsymbol{\Sigma}_{t|T} = \boldsymbol{\Sigma}_t + \mathbf{G}_t(\boldsymbol{\Sigma}_{t+1|T} - \boldsymbol{\Sigma}_{t+1|t})\mathbf{G}_t^{\mathsf{T}} \tag{8.119}$$

$$\mathbf{G}_t = \mathbf{D}_{t+1}\boldsymbol{\Sigma}_{t+1|t}^{-1} \tag{8.120}$$

$$\mathbf{D}_{t+1} = \sum_{i=0}^{2d} w_i^c(\boldsymbol{z}_{t,i} - \boldsymbol{\mu}_t)(\boldsymbol{z}_{t+1,i} - \boldsymbol{\mu}_{t+1|t})^{\mathsf{T}} \tag{8.121}$$

See e.g., [Sar13, Sec 9.3] for the derivation.

8.4 Other variants of the Kalman filter

In this section, we briefly mention some other variants of Kalman filtering. For a more extensive review, see [Li+17].

8.4.1 Ensemble Kalman filter

The **ensemble Kalman filter** (**EnKF**) is a technique developed in the geoscience (meteorology) community to perform approximate online inference in large nonlinear systems. The canonical reference is [Eve09], but a more accessible tutorial (using the same Bayesian signal processing approach we adopt in this chapter) is in [Rot+17].

EnKF is mostly used for problems where the hidden state represents an unknown physical quantity (e.g., temperature and pressure) at each point on a spatial grid, and the measurements are sparse and spatially localized. Combining this information over space and time is called **data assimilation**. However, the technique can be applied to other nonlinear state estimation problems. For example, it was recently used in

[Li+20] to model the spread of the SARS-CoV2 virus, using an ODE dynamics model, based on the EnKF method described in [And01]. We briefly explain the method below, following [Rot+17].

The key idea is to represent the belief state $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t})$ by a finite number of samples $\boldsymbol{z}_{t|t} = \{\boldsymbol{z}_{t|t}^s : s = 1 : S\}$, where each $\boldsymbol{z}_{t|t}^s \in N_z$. In contrast to particle filtering (**??**), the samples are updated in a manner that closely resembles the Kalman filter, so there is no importance sampling or resampling step. The downside is that the posterior does not converge to the true Bayesian posterior even as $S \to \infty$ [LGMT11], except in the linear-Gaussian case. However, sometimes the performance of EnKF can be better for small number of samples (although this depends of course on the PF proposal distribution).

The posterior mean and covariance can be derived from the ensemble of samples as follows:

$$\tilde{\boldsymbol{\mu}}_{t|t} = \frac{1}{S}\sum_{s=1}^{S} \boldsymbol{z}_{t|t}^s = \frac{1}{S}\boldsymbol{z}_{t|t}\mathbf{1} \tag{8.122}$$

$$\tilde{\boldsymbol{\Sigma}}_{t|t} = \frac{1}{S-1}\sum_{s=1}^{S}(\boldsymbol{z}_t^s - \tilde{\boldsymbol{\mu}}_{t|t})(\boldsymbol{z}_t^s - \tilde{\boldsymbol{\mu}}_{t|t})^{\mathsf{T}} = \frac{1}{S-1}\tilde{\mathbf{Z}}_{t|t}\tilde{\mathbf{Z}}_{t|t}^{\mathsf{T}} \tag{8.123}$$

where $\tilde{\mathbf{Z}}_{t|t} = \boldsymbol{z}_{t|t} - \tilde{\boldsymbol{\mu}}_{t|t}\mathbf{1}^{\mathsf{T}}$.

We update the samples as follows. For the time update, we first draw S system noise variables $\boldsymbol{v}_t^s \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$, and then we pass these, and the previous state estimate, through the dynamics model to get the one-step-ahead state predictions, $\boldsymbol{z}_{t|t-1} = f_z(\boldsymbol{z}_{t-1|t-1}, \mathbf{V}_t)$. This is the analog of **??**.

Next we draw S observation noise variables $\boldsymbol{e}_t^s \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$, and use them to compute the one-step-ahead observation predictions, $\boldsymbol{y}_{t|t-1} = f_x(\boldsymbol{z}_{t|t-1}, \mathbf{E}_t)$. This is the analog of **??**.

Finally we compute the measurement update using

$$\boldsymbol{z}_{t|t} = \boldsymbol{z}_{t|t-1} + \tilde{\mathbf{K}}_t(\boldsymbol{y}_t\mathbf{1}^{\mathsf{T}} - \boldsymbol{y}_{t|t-1}) \tag{8.124}$$

which is the analog of **??**.

We now discuss how to compute $\tilde{\mathbf{K}}_t$, which is the analog of the Kalman gain matrix in **??**. First note that we can write the exact Kalman gain matrix (in the linear-Gaussian case) as $\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1}\mathbf{C}^{\mathsf{T}}\mathbf{S}_t^{-1} = \mathbf{M}_t\mathbf{S}_t^{-1}$, where $\mathbf{S}_t$ is the covariance of the measurements, and $\mathbf{M}_t$ is the cross-covariance between the state and output predictions. In the EnKF, we approximate $\mathbf{S}_t$ and $\mathbf{M}_t$ empirically as follows. First we compute the deviations from predictions:

$$\tilde{\mathbf{Z}}_{t|t-1} = \boldsymbol{z}_{t|t-1}(\mathbf{I} - \frac{1}{N}\mathbf{1}\mathbf{1}^{\mathsf{T}}), \quad \tilde{\mathbf{Y}}_{t|t-1} = \boldsymbol{y}_{t|t-1}(\mathbf{I} - \frac{1}{N}\mathbf{1}\mathbf{1}^{\mathsf{T}}) \tag{8.125}$$

Then we compute the sample covariance matrices

$$\tilde{\mathbf{S}}_t = \frac{1}{S-1}\tilde{\mathbf{Y}}_{t|t-1}\tilde{\mathbf{Y}}_{t|t-1}^{\mathsf{T}}, \quad \tilde{\mathbf{M}}_t = \frac{1}{S-1}\tilde{\mathbf{Z}}_{t|t-1}\tilde{\mathbf{Y}}_{t|t-1}^{\mathsf{T}} \tag{8.126}$$

Finally we compute

$$\tilde{\mathbf{K}}_t = \tilde{\mathbf{M}}_t\tilde{\mathbf{S}}_t^{-1} \tag{8.127}$$

In practice, we should not perform this matrix inversion, but instead solve the linear system

$$\tilde{\mathbf{K}}_t\tilde{\mathbf{Y}}_{t|t-1}\tilde{\mathbf{Y}}_{t|t-1}^{\mathsf{T}} = \tilde{\mathbf{Z}}_{t|t-1}\tilde{\mathbf{Y}}_{t|t-1}^{\mathsf{T}} \tag{8.128}$$

We now compare the computational complexity to the KF algorithm. We will assume $N_z > S > N_y$, as occurs in most geospatial problems. The EnKF time update takes $O(N_z^2 S)$ operations, and the measurement update takes $O(N_z N_y S)$, where N_z is the number of latent dimensions and N_y is the number of observed dimensions. By contrast, in the KF, the time update takes $O(N_z^3)$ operations, and the measurement update takes $O(N_z^2 N_y)$. So we see that the EnKF is faster for high dimensional state-spaces, because it uses a low-rank approximation to the posterior covariance.

Unfortunately, if S is too small, the EnKF can be become overconfident, and the filter can diverge. Various heuristics (e.g., covariance inflation) have been proposed to fix this. However, most of these methods are ad-hoc. A variety of more well-principled solutions have also been proposed, see e.g., [FK13; Rei13].

8.4.2 Robust Kalman filters

In practice we often have noise that is non-Gaussian. A common example is when we have clutter, or outliers, in the observation model, or sudden changes in the process model. In this case, we might use the Laplace distribution [Ara+09] or the Student-t distribution [Ara10; RÖG13; Ara+17] as noise models.

[Hua+17] proposes a variational Bayes (**??**) approach, that allows the dynamical prior and the observation model to both be (linear) Student distributions, but where the posterior is approximated at each step using a Gaussian, conditional on the noise scale matrix, which is modeled using an inverse Wishart distribution. An extension of this, to handle mixture distributions, can be found in [Hua+19].

8.4.3 Gaussian filtering

In this section, we discuss a simple unified framework, known as the **Gaussian filter** [IX00; Wu+06], which includes EKF, UKF, and various other algorithms. Our presentation is based on [Sar13, Ch. 6].

8.4.3.1 The Gaussian approximation

To explain the approach, we temporarily drop the time indices, and the conditioning on past information, and consider a single time step of inference. Furthermore, we will use the shorthand

$$\int_x f(x) = \int_{-\infty}^{\infty} f(x)dx \tag{8.129}$$

Let $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z)$ and $p(\boldsymbol{y}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{g}(\boldsymbol{z}), \mathbf{Q})$ for some function $\boldsymbol{g}$. Let $p(\boldsymbol{z}, \boldsymbol{y}) = p(\boldsymbol{z})p(\boldsymbol{y}|\boldsymbol{z})$ be the exact joint distribution. The best Gaussian approximation to the joint can be obtained by **moment matching**, i.e.,

$$q(\boldsymbol{z}, \boldsymbol{y}) = \mathcal{N}\left(\begin{pmatrix}\boldsymbol{z}\\ \boldsymbol{y}\end{pmatrix} | \begin{pmatrix}\boldsymbol{\mu}_z\\ \boldsymbol{\mu}_y\end{pmatrix}, \begin{pmatrix}\boldsymbol{\Sigma}_z & \boldsymbol{\Sigma}_{zy}\\ \boldsymbol{\Sigma}_{zy}^{\mathsf{T}} & \boldsymbol{\Sigma}_y\end{pmatrix}\right) \tag{8.130}$$

where

$$\boldsymbol{\mu}_y = \int_{\boldsymbol{z}} \boldsymbol{g}(\boldsymbol{z})\mathcal{N}(\boldsymbol{z}|\boldsymbol{m}, \boldsymbol{\Sigma}) \tag{8.131}$$

$$\boldsymbol{\Sigma}_y = \int_{\boldsymbol{z}} (\boldsymbol{g}(\boldsymbol{z}) - \boldsymbol{\mu}_y)(\boldsymbol{g}(\boldsymbol{z}) - \boldsymbol{\mu}_y)^{\mathsf{T}}\mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z) + \mathbf{Q} \tag{8.132}$$

$$\boldsymbol{\Sigma}_{zy} = \int_{\boldsymbol{z}} (\boldsymbol{z} - \boldsymbol{\mu}_z)(\boldsymbol{g}(\boldsymbol{z}) - \boldsymbol{\mu}_y)^{\mathsf{T}}\mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z) \tag{8.133}$$

We can either compute these integrals by linearizing $\boldsymbol{g}$ and using closed form expressions, or by using numerical integration, as we discuss in Section 8.4.3.3.

Once we have computed the joint $q(\boldsymbol{z}, \boldsymbol{y})$, we can compute the posterior conditional $q(\boldsymbol{z}|\boldsymbol{y})$ using the usual rules for conditioning a Gaussian:

$$q(\boldsymbol{z}|\boldsymbol{y}) = \mathcal{N}\left(\boldsymbol{z}| \underbrace{\boldsymbol{\mu}_z + \boldsymbol{\Sigma}_{zy}\boldsymbol{\Sigma}_y^{-1}(\boldsymbol{y} - \boldsymbol{\mu}_y)}_{\boldsymbol{\mu}_{z|y}}, \underbrace{\boldsymbol{\Sigma}_z - \boldsymbol{\Sigma}_{zy}\boldsymbol{\Sigma}_y^{-1}\boldsymbol{\Sigma}_{zy}^{\mathsf{T}}}_{\boldsymbol{\Sigma}_{zz|y}}\right) \tag{8.134}$$

In practice, $\boldsymbol{\Sigma}_y$ may be rank deficient, so we should avoid computing $\boldsymbol{\Sigma}_y^{-1}$. Fortunately we can compute $\boldsymbol{\mu}_{z|x}$ and $\boldsymbol{\Sigma}_{zz|x}$ in a numerically stable way by solving a linear system. In particular, the posterior mean is the solution to

$$\boldsymbol{\mu}_{z|y} = \boldsymbol{\mu}_z + \boldsymbol{\Sigma}_{zy}\boldsymbol{a} \tag{8.135}$$

$$\boldsymbol{\Sigma}_y\boldsymbol{a} = \boldsymbol{y} - \boldsymbol{\mu}_y \tag{8.136}$$

and the posterior covariance is the solution to

$$\boldsymbol{\Sigma}_{zz|y} = \boldsymbol{\Sigma}_z - \boldsymbol{\Sigma}_{zy}\mathbf{A} \tag{8.137}$$

$$\boldsymbol{\Sigma}_y\mathbf{A} = \boldsymbol{\Sigma}_{zy}^\mathsf{T} \tag{8.138}$$

These solutions are unique, even if $\boldsymbol{\Sigma}_y$ is degenerate, as shown in [Wüt+16, App. A].

8.4.3.2 Application to online filtering

Let us now apply this method in the filtering context. We assume the prior has the form $p(\boldsymbol{z}_{t-1}|\boldsymbol{y}_{1:t-1}) = \mathcal{N}(\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1})$. We then compute the Gaussian prediction step as follows:

$$\boldsymbol{\mu}_{t|t-1} = \int_{\boldsymbol{z}_{t-1}} \boldsymbol{f}(\boldsymbol{z}_{t-1})\mathcal{N}(\boldsymbol{z}_{t-1}|\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}) \tag{8.139}$$

$$\boldsymbol{\Sigma}_{t|t-1} = \int_{\boldsymbol{z}_{t-1}} (\boldsymbol{f}(\boldsymbol{z}_{t-1}) - \boldsymbol{\mu}_{t|t-1})(\boldsymbol{f}(\boldsymbol{z}_{t-1}) - \boldsymbol{\mu}_{t|t-1})^\mathsf{T}\mathcal{N}(\boldsymbol{z}_{t-1}|\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}) + \mathbf{Q}_{t-1} \tag{8.140}$$

The update step becomes

$$\hat{\boldsymbol{y}}_t = \int_{\boldsymbol{z}_t} \boldsymbol{h}(\boldsymbol{z}_t)\mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}) \tag{8.141}$$

$$\mathbf{S}_t = \int_{\boldsymbol{z}_t} (\boldsymbol{h}(\boldsymbol{z}_t) - \hat{\boldsymbol{y}}_t)(\boldsymbol{h}(\boldsymbol{z}_t) - \hat{\boldsymbol{y}}_t)^\mathsf{T}\mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}) + \mathbf{R}_t \tag{8.142}$$

$$\mathbf{C}_t = \int_{\boldsymbol{z}_t} (\boldsymbol{z}_t - \boldsymbol{\mu}_{t|t-1})(\boldsymbol{h}(\boldsymbol{z}_t) - \hat{\boldsymbol{y}}_t)^\mathsf{T}\mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}) \tag{8.143}$$

$$\mathbf{K}_t = \mathbf{C}_t\mathbf{S}_t^{-1} \tag{8.144}$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t(\boldsymbol{y}_t - \hat{\boldsymbol{y}}_t) \tag{8.145}$$

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t\mathbf{S}_t\mathbf{K}_t^\mathsf{T} \tag{8.146}$$

8.4.3.3 Deriving EKF, UKF, and QKF

To implement the above integrals in practice, we usually need some approximations. Suppose we make the linear approximations

$$\boldsymbol{f}_t(\boldsymbol{z}) = \boldsymbol{f}(\boldsymbol{\mu}_{t-1}) + \mathbf{F}_{t-1}(\boldsymbol{z} - \boldsymbol{\mu}_{t|t-1}) \tag{8.147}$$

$$\boldsymbol{h}_t(\boldsymbol{z}) = \boldsymbol{h}(\boldsymbol{\mu}_{t|t-1}) + \mathbf{H}_t(\boldsymbol{y}_t - \boldsymbol{\mu}_{t|t-1}) \tag{8.148}$$

where $\mathbf{F}_{t-1}$ is the Jacobian of $\boldsymbol{f}$ at $\boldsymbol{\mu}_{t-1}$ and $\mathbf{H}_t$ is the Jacobian of $\boldsymbol{h}$ at $\boldsymbol{\mu}_{t|t-1}$. Plugging this into the above equations will give us the EKF.

Alternatively, we can use numerical integration methods, such as **spherical cubature integration**, which gives rise to the **cubature Kalman filter** [AH09]. This turns out (see [Sar13, p110]) to be a special case of the UKF, with $2n + 1$ sigma points, and fixed hyper-parameters of $\alpha = 1$ and $\beta = 0$, with κ left free.

A more accurate approximation uses **Gauss-Hermite integration**, which allows the user to select more sigma points (see [Sar13, Sec 6.3]). This gives rise the **quadrature Kalman filter** or **QKF** [AHE07].

We can also approximate the integrals with Monte Carlo. Note, however, that this is not the same as particle filtering (**??**), which approximates the conditional $p(\boldsymbol{z}|\boldsymbol{y})$ rather than the joint $p(\boldsymbol{z}, \boldsymbol{y})$, as explained in **??**.

8.5 Assumed density filtering for SLDS models

In this section, we discuss the application of the **assumed density filtering** or **ADF** [May79] algorithm to switching linear dynamical systems (SLDS). The resulting method is known as the **Gaussian sum filter**.

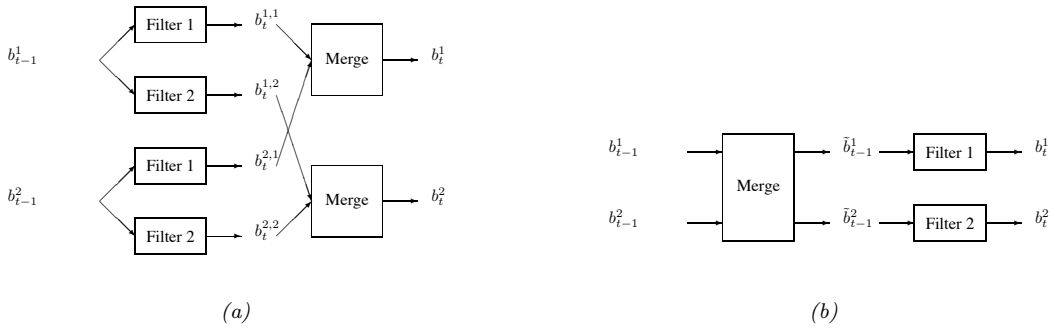


Figure 8.8: ADF for a switching linear dynamical system with 2 discrete states. (a) GPB2 method. (b) IMM method.

A Gaussian sum filter approximates the belief state at each step by a mixture of K Gaussians. This can be implemented by running K Kalman filters in parallel. This is particularly well suited to switching SSMs. We now describe one version of this algorithm, known as the "second order **generalized pseudo Bayes filter**" (GPB2) [BSF88]. We assume that the prior belief state b_{t-1} is a mixture of K Gaussians, one per discrete state:

$$b_{t-1}^i \triangleq p(\boldsymbol{z}_{t-1}, c_{t-1} = i|\boldsymbol{y}_{1:t-1}) = \pi_{t-1,i}\mathcal{N}(\boldsymbol{z}_{t-1}|\boldsymbol{\mu}_{t-1,i}, \boldsymbol{\Sigma}_{t-1,i}) \tag{8.149}$$

where $i \in \{1, \ldots, K\}$. We then pass this through the K different linear models to get

$$b_t^{ij} \triangleq p(\boldsymbol{z}_t, c_{t-1} = i, c_t = j|\boldsymbol{y}_{1:t}) = \pi_{tij}\mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_{t,ij}, \boldsymbol{\Sigma}_{t,ij}) \tag{8.150}$$

where $\pi_{tij} = \pi_{t-1,i}p(c_t = j|c_{t-1} = i)$. Finally, for each value of j, we collapse the K Gaussian mixtures down to a single mixture to give

$$b_t^j \triangleq p(\boldsymbol{z}_t, c_t = j|\boldsymbol{y}_{1:t}) = \pi_{tj}\mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_{t,j}, \boldsymbol{\Sigma}_{t,j}) \tag{8.151}$$

See Figure 8.8a for a sketch.

The optimal way to approximate a mixture of Gaussians with a single Gaussian is given by $q = \arg\min_q D_{\mathbb{KL}}(q \parallel p)$, where $p(\boldsymbol{z}) = \sum_k \pi_k \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ and $q(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. This can be solved by moment matching, that is,

$$\boldsymbol{\mu} = \mathbb{E}[\boldsymbol{z}] = \sum_k \pi_k \boldsymbol{\mu}_k \tag{8.152}$$

$$\boldsymbol{\Sigma} = \mathrm{Cov}[\boldsymbol{z}] = \sum_k \pi_k \left(\boldsymbol{\Sigma}_k + (\boldsymbol{\mu}_k - \boldsymbol{\mu})(\boldsymbol{\mu}_k - \boldsymbol{\mu})^\mathsf{T}\right) \tag{8.153}$$

In the graphical model literature, this is called **weak marginalization** [Lau92], since it preserves the first two moments. Applying these equations to our model, we can go from b_t^{ij} to b_t^j as follows (where we drop the t subscript for brevity):

$$\pi_j = \sum_i \pi_{ij} \tag{8.154}$$

$$\pi_{j|i} = \frac{\pi_{ij}}{\sum_{j'} \pi_{ij'}} \tag{8.155}$$

$$\boldsymbol{\mu}_j = \sum_i \pi_{j|i} \boldsymbol{\mu}_{ij} \tag{8.156}$$

$$\boldsymbol{\Sigma}_j = \sum_i \pi_{j|i} \left(\boldsymbol{\Sigma}_{ij} + (\boldsymbol{\mu}_{ij} - \boldsymbol{\mu}_j)(\boldsymbol{\mu}_{ij} - \boldsymbol{\mu}_j)^\mathsf{T}\right) \tag{8.157}$$

This algorithm requires running K^2 filters at each step. A cheaper alternative, known as **interactive multiple models** or **IMM** [BSF88], can be obtained by first collapsing the prior to a single Gaussian (by moment matching), and then updating it using K different Kalman filters, one per value of c_t. See Figure 8.8b for a sketch.

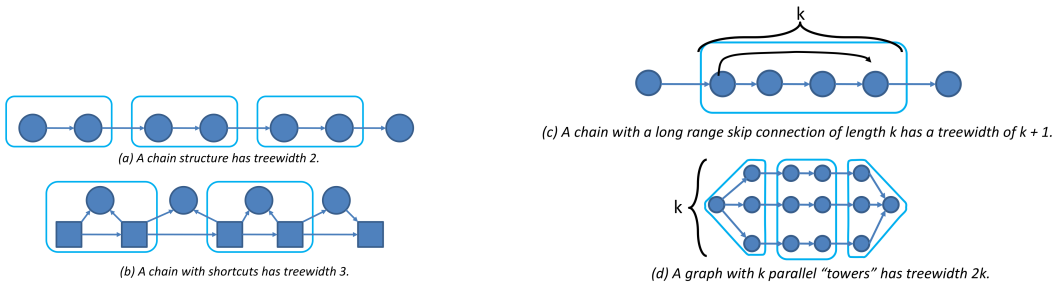


Figure 8.9: Examples of optimal tree decompositions for some common graph structures. Adapted from https://bit.ly/2m5vauG.

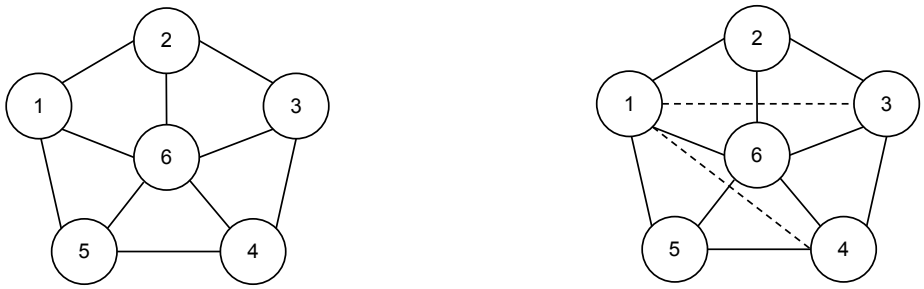


Figure 8.10: A triangulated graph is not just composed of little triangles. Left: this graph is not triangulated, despite appearances, since it contains a chordless 5-cycle 1-2-3-4-5-1. Right: one possible triangulation, by adding the 1-3 and 1-4 fill-in edges. Adapted from [Arm05, p46]

8.6 More junction trees

In this section, we discuss more topics related to junction trees.

8.6.1 Tree decompositions of some common graph structures

In **??**, 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., [GW08; BMR97; ZP00; Che+16]. For example, for a linear chain, we can reduce the memory from $O(L)$ to $O(\log L)$, if we are willing to increase the runtime from $O(L)$ to $O(L \log L)$.

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 [LT79] for the formal proof.)

Note that a graph may look like it is triangulated, even though it is not. For example, Figure 8.10(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 8.10(b), in which we add two fill-in edges.

8.6.2 Junction tree algorithm 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 [SHJ97], 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 8.11. We

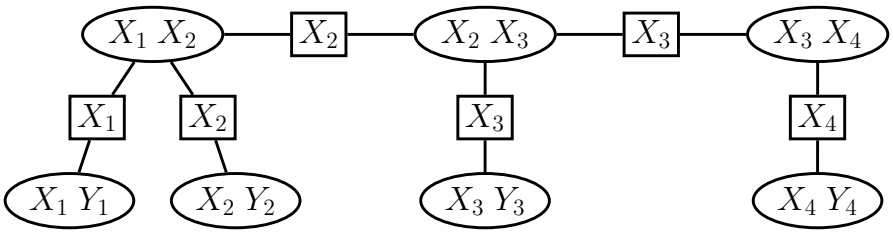


Figure 8.11: The junction tree derived from an HMM of length $T = 4$.

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

$$\psi^*_{ij}(X_{t-1}) = \sum_{X_{t-2}} \psi_i(X_{t-2}, X_{t-1}) = p(X_{t-1}|\boldsymbol{y}_{1:t-1}) = \alpha_{t-1})(X_t) \tag{8.158}$$

$$\psi^*_{i'j}(X_t) = \sum_{Y_t} \psi_i(X_t, Y_t) \propto p(Y_t|X_t) = \lambda_t(X_t) \tag{8.159}$$

We then absorb messages from these separator potentials to compute the new clique potential:

$$\psi^*_i(X_{t-1}, X_t) \propto \psi_i(X_{t-1}, X_t) \frac{\psi^*_{ij}(X_{t-1})}{\psi_{ij}(X_{t-1})} \frac{\psi^*_{i'j}(X_t)}{\psi_{i'j}(X_t)} \tag{8.160}$$

$$= A(X_{t-1}, X_t) \frac{\alpha_{t-1}(X_{t-1})}{1} \frac{\lambda_t(X_t)}{1} \propto p(X_{t-1}, X_t|\boldsymbol{y}_{1:t}) \tag{8.161}$$

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:

$$\psi^{**}_j(X_{t-1}, X_t) \propto \psi^*_j(X_{t-1}, X_t) \frac{\psi^{**}_{k,j}(X_t)}{\psi^*_{k,j}(X_t)} \tag{8.162}$$

$$= \left[A(X_{t-1}, X_t)\alpha_{t-1}(X_{t-1})\lambda_t(X_t)\right] \frac{\gamma_t(X_t)}{\alpha_t(X_t)} \tag{8.163}$$

$$\propto p(X_{t-1}, X_t|\boldsymbol{y}_{1:T}) \tag{8.164}$$

where $\alpha_t(X_t) = p(X_t|\boldsymbol{y}_{1:t})$ and $\gamma_t(X_t) = p(X_t|\boldsymbol{y}_{1:T})$ are the separator potentials for S_{jk} on the forwards and backwards passes. This matches the two slice smothed marginal in **??**.

8.6.3 JTA for temporal graphical models

In this section, we discuss how to perform exact inference in temporal graphical models, which includes dynamic Bayes nets (**??**) 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** x_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

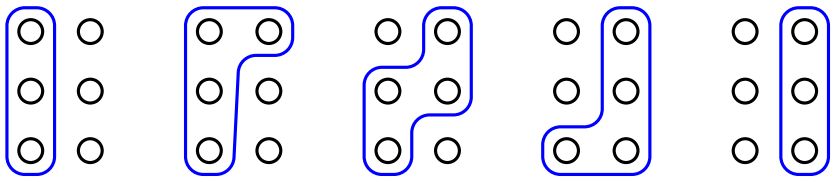


Figure 8.12: The cliques in the junction tree decomposition as we advance the frontier from one time-slice to the next in a 3-chain factorial HMM model.

chains, with transition matrices given by

$$\mathbf{A}_1 = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \ \mathbf{A}_2 = \begin{pmatrix} e & f \\ g & h \end{pmatrix} \tag{8.165}$$

Then the transition matrix of the flattened model has the following Kronecker product form:

$$\mathbf{A} = \mathbf{A}_1 \otimes \mathbf{A}_2 = \begin{pmatrix} ae & af & be & bf \\ ag & ah & bg & bh \\ ce & cf & de & df \\ cg & ch & dg & dh \end{pmatrix} \tag{8.166}$$

For example, the probability of going from state (1,2) to (2,1) is $p(1 \to 2) \times p(2 \to 1) = b \times g$. We note that this is not a sparse matrix, oeven though the chains are completely independent.

One can use this expanded matrix inside the forwards-backwards algorithm to compute $p(x_{1,t}, x_{2,t}|\boldsymbol{y}_{1:T}, \boldsymbol{\theta})$, from which the marginals of each chain, $p(x_{i,t}|\boldsymbol{y}_{1:T}, \boldsymbol{\theta})$, 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 **??**(a). 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 $p(x_{1:M,t}|\boldsymbol{y}_{1:t}, \boldsymbol{\theta})$ 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 8.12. This takes $O(TMK^{M+1})$ time (see [GJ97, App. B] for the details of the algorithm). This method has been called the **frontier algorithm** [ZM96], 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 [Bil10].

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 **??**(b). 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 **??**(b), this is perhaps not so suprising, 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 28.3.4.1, we derive a structured mean field approximation for FHMMs, which

exploits the parallel chain structure. This only takes $O(TMK^2I)$ time, where I is the number of iterations of the inference algorithm (typically $I \sim 10$ suffices for good performance). See also **??** for an approach based on assumed density filtering, and **??** for an approach based on particle 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".

8.7 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 $\boldsymbol{x}^* = \arg\max_{\boldsymbol{x} \in \mathcal{X}^V} p(\boldsymbol{x})$, where $\mathcal{X} = \{1, \ldots, K\}$ 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 (**??**) with pairwise cliques, one per edge:

$$p(\boldsymbol{x}) = \frac{1}{Z} \exp\left\{ \sum_{s \in \mathcal{V}} \theta_s(x_s) + \sum_{(s,t) \in \mathcal{E}} \theta_{st}(x_s, x_t) \right\} \tag{8.167}$$

Here $\mathcal{V} = \{x_1, \ldots, x_V\}$ 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:

$$Z = \sum_{\boldsymbol{x}} \exp\left\{ \sum_{s \in \mathcal{V}} \theta_s(x_s) + \sum_{(s,t) \in \mathcal{E}} \theta_{st}(x_s, x_t) \right\} \tag{8.168}$$

Since we just want the MAP configuration, we can ignore Z, and just compute

$$\boldsymbol{x}^* = \operatorname*{argmax}_{\boldsymbol{x}} \sum_{s \in \mathcal{V}} \theta_s(x_s) + \sum_{(s,t) \in \mathcal{E}} \theta_{st}(x_s, x_t) \tag{8.169}$$

We can compute this exactly using dynamic programming as we explain in **??**; 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 [WJ08; KF09].

8.7.1 Notation

To simplify the presentation, we write the distribution in the following form:

$$p(\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{\theta})} \exp(-\mathcal{E}(\boldsymbol{x}))\mathcal{E}(\boldsymbol{x}) \triangleq -\boldsymbol{\theta}^{\mathsf{T}} \mathcal{T}(\boldsymbol{x}) \tag{8.170}$$

where $\boldsymbol{\theta} = (\{\theta_{s;j}\}, \{\theta_{s,t;j,k}\})$ are all the node and edge parameters (the canonical parameters), and $\mathcal{T}(\boldsymbol{x}) = (\{\mathbb{I}(x_s = j)\}, \{\mathbb{I}(x_s = j, x_t = k)\})$ 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

$$\boldsymbol{\mu} = \mathbb{E}[\mathcal{T}(\boldsymbol{x})] = (\{p(x_s = j)\}_s, \{p(x_s = j, x_t = k)\}_{s \neq t}) = (\{\mu_{s;j}\}_s, \{\mu_{st;jk}\}_{s \neq t}) \tag{8.171}$$

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 $p(\boldsymbol{x})$, so we sometimes treat $\boldsymbol{\mu}$ as a distribution itself.

Equation (8.171) 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

$$p(\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left\{ \sum_{s \in \mathcal{V}} \theta_s x_s + \sum_{(s,t) \in \mathcal{E}} \theta_{st} x_s x_t \right\} \tag{8.172}$$

Hence we can use the following minimal parameterization

$$\mathcal{T}(\boldsymbol{x}) = (x_s, s \in V; x_s x_t, (s,t) \in \mathcal{E}) \in \mathbb{R}^d \tag{8.173}$$

where $d = V + E$. The corresponding mean parameters are $\mu_s = p(x_s = 1)$ and $\mu_{st} = p(x_s = 1, x_t = 1)$.

8.7.2 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:

$$\mathbb{M}(G) \triangleq \{\boldsymbol{\mu} \in \mathbb{R}^d : \exists p \text{ s.t. } \boldsymbol{\mu} = \sum_{\boldsymbol{x}} \mathcal{T}(\boldsymbol{x}) p(\boldsymbol{x}) \text{ for some } p(\boldsymbol{x}) \geq 0, \sum_{\boldsymbol{x}} p(\boldsymbol{x}) = 1\} \tag{8.174}$$

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

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \\ -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_{12} \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \end{pmatrix} \tag{8.175}$$

These four constraints define a series of half-planes, whose intersection defines a polytope, as shown in Figure 8.13(a).

Since $\mathbb{M}(G)$ is obtained by taking a convex combination of the $\mathcal{T}(\boldsymbol{x})$ vectors, it can also be written as the convex hull of these vectors:

$$\mathbb{M}(G) = \text{conv}\{\mathcal{T}_1(\boldsymbol{x}), \ldots, \mathcal{T}_d(\boldsymbol{x})\} \tag{8.176}$$

For example, for a 2 node MRF $X_1 - X_2$ with binary states, we have

$$\mathbb{M}(G) = \text{conv}\{(0,0,0), (1,0,0), (0,1,0), (1,1,1)\} \tag{8.177}$$

These are the four black dots in Figure 8.13(a). We see that the convex hull defines the same volume as the intersection of half-spaces.

8.7.3 Linear programming relaxation

We can write the MAP estimation problem as follows:

$$\max_{\boldsymbol{x} \in \mathcal{X}^V} \boldsymbol{\theta}^\mathsf{T} \mathcal{T}(\boldsymbol{x}) = \max_{\boldsymbol{\mu} \in \mathbb{M}(G)} \boldsymbol{\theta}^\mathsf{T} \boldsymbol{\mu} \tag{8.178}$$

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 assigment 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.

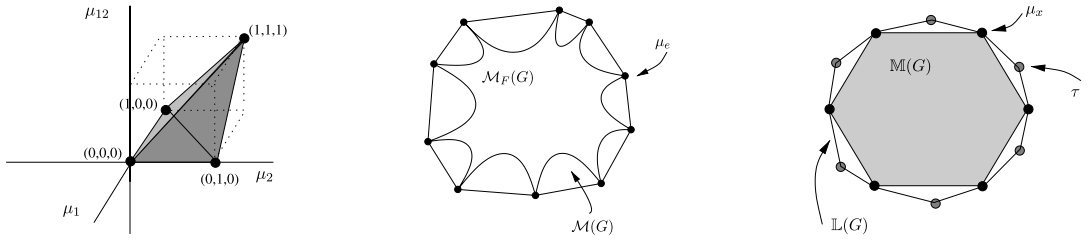


Figure 8.13: (a) Illustration of the marginal polytope for an Ising model with two variables. (b) Cartoon illustration of the set $\mathbb{M}_F(G)$, which is a nonconvex inner bound on the marginal polytope $\mathbb{M}(G)$. $\mathbb{M}_F(G)$ is used by mean field. (c) Cartoon illustration of the relationship 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 [WJ08]. Used with kind permission of Martin Wainwright.

It seems like we have an easy problem to solve, since the objective in Equation (8.178) 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 8.7.3.1. Thus we try to maximize the following upper bound on the original objective:

$$\boldsymbol{\tau}^* = \underset{\boldsymbol{\tau} \in \mathbb{L}(G)}{\operatorname{argmax}} \, \boldsymbol{\theta}^\mathsf{T} \boldsymbol{\tau} \tag{8.179}$$

This is called a **linear programming relaxation** of the problem. If the solution $\boldsymbol{\tau}^*$ is integral, it corresponds to the exact MAP estimate; this will be the case when the graph is a tree. In general, $\boldsymbol{\tau}^*$ will be fractional; we can derive an approximate MAP estimate by rounding (see [Wer07] for details).

8.7.3.1 A convex outer approximation to the marginal polytope

Consider a set of probability vectors $\boldsymbol{\tau}$ that satisfy the following **local consistency** constraints:

$$\sum_{x_s} \tau_s(x_s) = 1 \tag{8.180}$$

$$\sum_{x_t} \tau_{st}(x_s, x_t) = \tau_s(x_s) \tag{8.181}$$

The first constraint is called the normalization constraint, and the second is called the marginalization constraint. We then define the set

$$\mathbb{L}(G) \triangleq \{\boldsymbol{\tau} \geq 0 : (\textit{Equation } (8.180)) \text{ holds } \forall s \in \mathcal{V}, (\textit{Equation } (8.181)) \text{ holds } \forall (s,t) \in \mathcal{E}\} \tag{8.182}$$

The set $\mathbb{L}(G)$ is also a polytope, but it only has $O(|V|+|E|)$ constraints. It is a convex **outer approximation** on $\mathbb{M}(G)$, as shown in Figure 8.13(c). (By contrast, the mean field approximation, which we discuss in **??**, is a non-convex inner approximation, as we discuss in **??**.)

We call the terms $\tau_s, \tau_{st} \in \mathbb{L}(G)$ **pseudo marginals**, since they may not correspond to marginals of any valid probability distribution. As an example of this, consider Figure 8.14(a). The picture shows a set of pseudo node and edge marginals, which satisfy the local consistency requirements. However, they are not

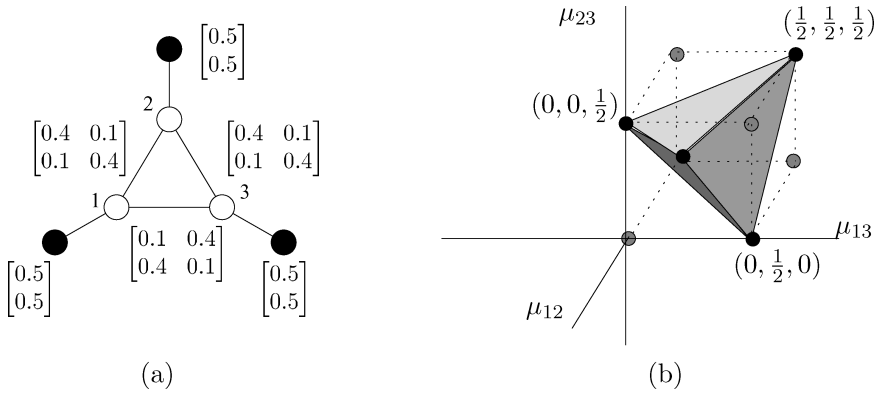


Figure 8.14: (a) Illustration of pairwise UGM on binary nodes, together with a set of pseudo marginals that are not globally consistent. (b) A slice of the marginal polytope illustrating the set of feasible edge marginals, assuming the node marginals are clamped at $\mu_1 = \mu_2 = \mu_3 = 0.5$. From Figure 4.1 of [WJ08]. Used with kind permission of Martin Wainwright.

globally consistent. To see why, note that τ_{12} implies $p(X_1 = X_2) = 0.8$, τ_{23} implies $p(X_2 = X_3) = 0.8$, but τ_{13} implies $p(X_1 = X_3) = 0.2$, which is not possible (see [WJ08, p81] for a formal proof). Indeed, Figure 8.14(b) shows that $\mathbb{L}(G)$ contains points that are not in $\mathbb{M}(G)$.

We claim that $\mathbb{M}(G) \subseteq \mathbb{L}(G)$, with equality iff G is a tree. To see this, first consider an element $\boldsymbol{\mu} \in \mathbb{M}(G)$. Any such vector must satisfy the normalization and marginalization constraints, hence $\mathbb{M}(G) \subseteq \mathbb{L}(G)$.

Now consider the converse. Suppose T is a tree, and let $\boldsymbol{\mu} \in \mathbb{L}(T)$. By definition, this satisfies the normalization and marginalization constraints. However, any tree can be represented in the form

$$p_{\boldsymbol{\mu}}(\boldsymbol{x}) = \prod_{s \in V} \mu_s(x_s) \prod_{(s,t) \in E} \frac{\mu_{st}(x_s, x_t)}{\mu_s(x_s)\mu_t(x_t)} \tag{8.183}$$

Hence satsifying normalization and local consistency is enough to define a valid distribution for any tree. Hence $\boldsymbol{\mu} \in \mathbb{M}(T)$ as well.

In contrast, if the graph has loops, we have that $\mathbb{M}(G) \neq \mathbb{L}(G)$. See Figure 8.14(b) for an example of this fact. The importance of this observation will become clear in Section 9.1.3.

8.7.3.2 Algorithms

Our task is to solve Equation (8.179), 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 $\boldsymbol{\tau}^*$. In particular, the **tree reweighted belief propagation** algorithm can be used; see Section 9.1.5.3 for details.

8.7.3.3 Application to stereo depth estimation

Belief propagation is often applied to low-level computer vision problems (see e.g., [Sze10; BKR11; Pri12]). For example, Figure 8.15 illustrates its application to the problem of **stereo depth estimation** given a pair of monocular images (only one is shown). The value x_i is the distance of pixel i from the camera (quantized to a certain number of values). The goal is to infer these values from noisy measurements. We quantize the state space, rather than using a Gaussian model, in order to avoid oversmoothing at discontinuities, which occur at object boundaries, as illustrated in Figure 8.15. (We can also use a hybrid discrete-continuous state space, as discussed in [Yam+12], but we can no longer apply BP.)

Not surprisingly, people have recently applied deep learning to this problem. For example, [XAH19] describes a differentiable version of message passing (**??**), which is fast and can be trained end-to-end.

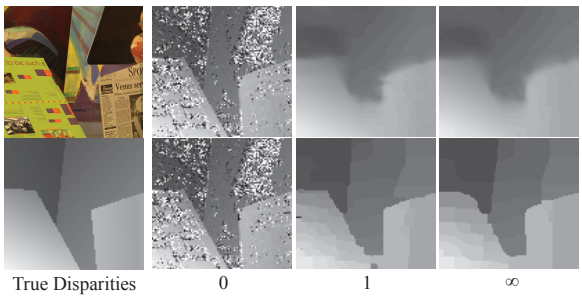


Figure 8.15: Illustration of belief propagation for stereo depth estimation applied to the Venus image from the Middlebury stereo benchmark dataset [SS02]. Left column: image and true disparities. Remaining columns: initial estimate, estimate after 1 iteration, and estimate at convergence. Top row: Gaussian edge potentials using a continuous state space. Bottom row: robust edge potentials using a quantized state space. From Figure 4 of [SF08]. Used with kind permission of Erik Sudderth.

However, it requires labeled data for training, i.e., pixel-wise ground truth depth values. For this particular problem, such data can be collected from depth cameras, but for other problems, BP on "unsupervised" MRFs may be needed.

8.7.4 Graphcuts

In this section, we show how to find MAP state estimates, or equivalently, minimum energy configurations, by using the **maxflow** / **mincut** algorithm for graphs. This class of methods is known as **graphcuts** and is very widely used, especially in computer vision applications (see e.g., [BK04]).

We will start by considering the case of MRFs with binary nodes and a restricted class of potentials; in this case, graphcuts will find the exact global optimum. We then consider the case of multiple states per node; we can approximately solve this case by solving a series of binary subproblems, as we will see.

8.7.4.1 Graphcuts for the Ising model

Let us start by considering a binary MRF where the edge energies have the following form:

$$\mathcal{E}_{uv}(x_u, x_v) = \begin{cases} 0 & \text{if } x_u = x_v \\ \lambda_{uv} & \text{if } x_u \neq x_v \end{cases} \tag{8.184}$$

where $\lambda_{st} \geq 0$ is the edge cost. This encourages neighboring nodes to have the same value (since we are trying to minimize energy). Since we are free to add any constant we like to the overall energy without affecting the MAP state estimate, let us rescale the local energy terms such that either $\mathcal{E}_u(1) = 0$ or $\mathcal{E}_u(0) = 0$.

Now let us construct a graph which has the same set of nodes as the MRF, plus two distinguished nodes: the source s and the sink t. If $\mathcal{E}_u(1) = 0$, we add the edge $x_u \to t$ with cost $\mathcal{E}_u(0)$. Similarly, If $\mathcal{E}_u(0) = 0$, we add the edge $s \to x_u$ with cost $\mathcal{E}_u(1)$. Finally, for every pair of variables that are connected in the MRF, we add edges $x_u \to x_v$ and $x_v \to x_u$, both with cost $\lambda_{u,v} \geq 0$. Figure 8.16 illustrates this construction for an MRF with 4 nodes and the following parameters:

$$\mathcal{E}_1(0) = 7, \mathcal{E}_2(1) = 2, \mathcal{E}_3(1) = 1, \mathcal{E}_4(1) = 6 \lambda_{1,2} = 6, \lambda_{2,3} = 6, \lambda_{3,4} = 2, \lambda_{1,4} = 1 \tag{8.185}$$

Having constructed the graph, we compute a minimal $s-t$ cut. This is a partition of the nodes into 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 sum of the cost of the edges between nodes on different sides of the partition:

$$\text{cost}(\mathcal{X}_s, \mathcal{X}_t) = \sum_{x_u \in \mathcal{X}_s, x_v \in \mathcal{X}_t} \text{cost}(x_u, x_v) \tag{8.186}$$

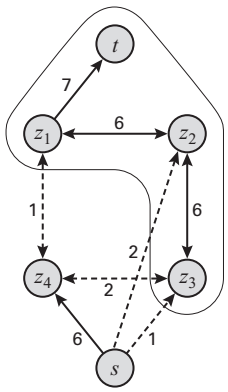


Figure 8.16: Illustration of graphcuts applied to an MRF with 4 nodes. Dashed lines are ones which contribute to the cost of the cut (for bidirected edges, we only count one of the costs). Here the min cut has cost 6. From Figure 13.5 from [KF09]. Used with kind permission of Daphne Koller.

In Figure 8.16, 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 8.16, 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]).

8.7.4.2 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:

$$\mathcal{E}_{uv}(1,1) + \mathcal{E}_{uv}(0,0) \leq \mathcal{E}_{uv}(1,0) + \mathcal{E}_{uv}(0,1) \tag{8.187}$$

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 (**??**). 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 [GPS89].

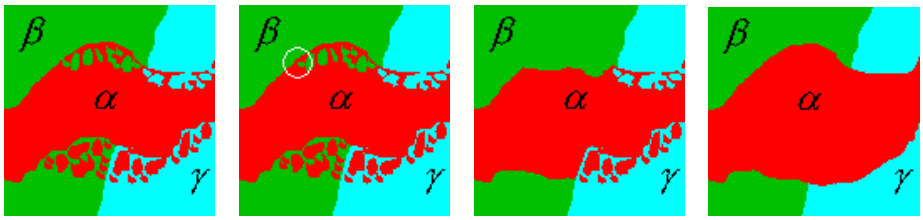
8.7.4.3 Graphcuts for nonbinary metric MRFs

We now discuss how to use graphcuts for approximate MAP estimation in MRFs where each node can have multiple states [BVZ01].

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 8.17(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 8.17(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



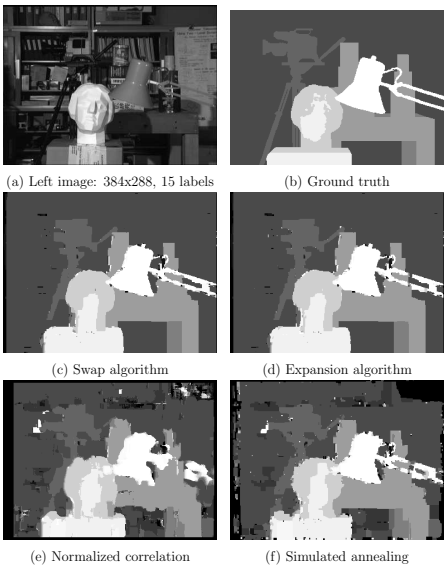
(a) initial labeling (b) standard move (c) α-β-swap (d) α-expansion

Figure 8.17: (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 [BVZ01]. Used with kind permission of Ramin Zabih.

neighbors to have similar labels, but never "punishes" them by more than δ. (This δ term prevents over-smoothing, which we illustrate in Figure 8.15.)

8.7.4.4 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 8.7.3.3. Figure 8.18 compares graphcuts (both swap and expansion version) to two other algorithms (simulated annealated, 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 [Sze+08; TF03]. It also tends to outperform belief propagation (results not shown) in terms of speed and accuracy on stereo problems [Sze+08; TF03], as well as other problems such as CRF labeling of LIDAR point cloud data [LMW17].



(a) Left image: 384x288, 15 labels (b) Ground truth

(c) Swap algorithm (d) Expansion algorithm

(e) Normalized correlation (f) Simulated annealing

Figure 8.18: 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 [BVZ01]. Used with kind permission of Ramin Zabih.*

Chapter 9

Variational inference

9.0.1 Exploiting partial conjugacy

If the full conditionals of the joint model are conjugate distributions, we can use the VMP approach of **??** to approximate the posterior one term at a time, similar to Gibbs sampling (**??**). However, in many models, some parts of the joint distribution are conjugate, and some are non-conjugate. In [KL17] they proposed the **conjugate-computation variational inference** or **CVI** method to tackle models of this form. They exploit the partial conjugacy to perform some updates in closed form, and perform the remaining updates using stochastic approximations.

To explain the method in more detail, let us assume the joint distribution has the form

$$p(\boldsymbol{y}, \boldsymbol{z}) \propto \tilde{p}_{nc}(\boldsymbol{y}, \boldsymbol{z}) \tilde{p}_c(\boldsymbol{y}, \boldsymbol{z}) \tag{9.1}$$

where $\boldsymbol{z}$ are all the latents (global or local), $\boldsymbol{y}$ are all the observabels (data)[1], 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 $\boldsymbol{z}$. More precisely, we assume the conjugate part is an exponential family model of the following form:

$$\tilde{p}_c(\boldsymbol{y}, \boldsymbol{z}) = h(\boldsymbol{z}) \exp[\mathcal{T}(\boldsymbol{z})^\mathsf{T} \boldsymbol{\eta} - A_c(\boldsymbol{\eta})] \tag{9.2}$$

where $\boldsymbol{\eta}$ is a *known* vector of natural parameters. (Any unknown model parameters should be included in the latent state $\boldsymbol{z}$, as we illustrate below.) We also assume that the variational posterior is an exponential family model with the same sufficient statistics, but different parameters:

$$q(\boldsymbol{z}|\boldsymbol{\lambda}) = h(\boldsymbol{z}) \exp[\mathcal{T}(\boldsymbol{z})^\mathsf{T} \boldsymbol{\eta} - A(\boldsymbol{\lambda})] \tag{9.3}$$

The mean parameters are given by $\boldsymbol{\mu} = \mathbb{E}_q[\mathcal{T}(\boldsymbol{z})]$. We assume the sufficient statistics are minimal, so that there is a unique 1:1 mapping between $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$: using

$$\boldsymbol{\mu} = \nabla_\lambda A(\boldsymbol{\lambda}) \tag{9.4}$$

$$\boldsymbol{\lambda} = \nabla_{\boldsymbol{\mu}} A^*(\boldsymbol{\mu}) \tag{9.5}$$

where A^* is the conjugate of A (see **??**). The ELBO is given by

$$Ł(\boldsymbol{\lambda}) = \mathbb{E}_q[\log p(\boldsymbol{y}, \boldsymbol{z}) - \log q(\boldsymbol{z}|\boldsymbol{\lambda})] \tag{9.6}$$

$$Ł(\boldsymbol{\mu}) = Ł(\boldsymbol{\lambda}(\boldsymbol{\mu})) \tag{9.7}$$

The simplest way to fit this variational posterior is to perform SGD on the ELBO wrt the natural parameters:

$$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t + \eta_t \nabla_{\boldsymbol{\lambda}} Ł(\boldsymbol{\lambda}_t) \tag{9.8}$$

[1] We denote observables by $\boldsymbol{y}$ since the examples we consider later on are conditional models, where $\boldsymbol{x}$ denote the inputs.

(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:

$$\boldsymbol{\lambda}_{t+1} = \underset{\boldsymbol{\lambda}\in\Omega}{\operatorname{argmin}} \underbrace{(\nabla_{\boldsymbol{\lambda}} \text{Ł}(\boldsymbol{\lambda}_t))^\intercal \boldsymbol{\lambda} - \frac{1}{2\eta_t}||\boldsymbol{\lambda} - \boldsymbol{\lambda}_t||_2^2}_{J(\boldsymbol{\lambda})} \tag{9.9}$$

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

$$\nabla_{\boldsymbol{\lambda}} J(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}_t) - \frac{1}{2\eta_t}(2\boldsymbol{\lambda} - 2\boldsymbol{\lambda}_t) = \mathbf{0} \tag{9.10}$$

from which we get Equation (9.8).

We can replace the Euclidean distance with a more general proximity function, such as the Bregman divergence between the distributions (see Section 6.2.1). This gives rise to the **mirror descent** algorithm (Section 6.2). We can also perform updates in the mean parameter space. In [RM15], they show that this is equivalent to performing natural gradient updates in the natural parameter space. Thus this method is sometimes called **natural gradient VI** or **NGVI**. Combining these two steps gives the following update equation:

$$\boldsymbol{\mu}_{t+1} = \underset{\boldsymbol{\mu}\in\mathcal{M}}{\operatorname{argmin}} (\nabla_{\boldsymbol{\mu}} \text{Ł}(\boldsymbol{\mu}_t))^\intercal \boldsymbol{\mu} - \frac{1}{\eta_t} B_{A*}(\boldsymbol{\mu}||\boldsymbol{\mu}_t) \tag{9.11}$$

where $\mathcal{M}$ is the space of valud mean parameters and $\eta_t > 0$ is a stepsize.

In [KL17], they show that the above update is equivalent to performing exact Bayesian inference in the following conjugate model:

$$q(\boldsymbol{z}|\boldsymbol{\lambda}_{t+1}) \propto e^{\mathcal{T}(\boldsymbol{z})^\intercal \tilde{\boldsymbol{\lambda}}_t} \tilde{p}_c(\boldsymbol{y}, \boldsymbol{z}) \tag{9.12}$$

We can think of the first term as an exponential family approximation to the non-conjugate part of the model, using local variational natural parameters $\tilde{\boldsymbol{\lambda}}_t$. (These are similar to the site parameters used in expectation propagation **??**.) These can be computed usng the following recursive update:

$$\tilde{\boldsymbol{\lambda}}_t = (1-\eta_t)\tilde{\boldsymbol{\lambda}}_{t-1} + \eta_t \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{\mu}_t)} \left[\log \tilde{p}_{nc}(\boldsymbol{y}, \boldsymbol{z})\right] \tag{9.13}$$

where $\tilde{\boldsymbol{\lambda}}_0 = \mathbf{0}$ and $\tilde{\boldsymbol{\lambda}}_1 = \boldsymbol{\eta}$. (Details on how to compute this derivative are given in **??**.) Once we can have "conjugated" the non-conjugate part, the natural parameter of the new variational posterior is obtained by

$$\boldsymbol{\lambda}_{t+1} = \tilde{\boldsymbol{\lambda}}_t + \boldsymbol{\eta} \tag{9.14}$$

This corresponds to a multiplicative update of the form

$$q_{t+1}(\boldsymbol{z}) \propto q_t(\boldsymbol{z})^{1-\eta_t} \left[\exp(\tilde{\boldsymbol{\lambda}}_t^\intercal \mathcal{T}(\boldsymbol{z}))\right]^{\eta_t} \tag{9.15}$$

We give some examples of this below.

9.0.1.1 Example: Gaussian process with non-Gaussian likelihoods

In **??**, we discuss Gaussian processes, which are a popular model for non-parametric regression. Given a set of N inputs $\boldsymbol{x}_n \in \mathcal{X}$ and outputs $y_n \in \mathbb{R}$, we define the following joint Gaussian distribution:

$$p(\boldsymbol{y}_{1:N}, \boldsymbol{z}_{1:N}|\mathbf{X}) = \left[\prod_{n=1}^N \mathcal{N}(y_n|z_n, \sigma^2)\right] \mathcal{N}(\boldsymbol{z}|\mathbf{0}, \mathbf{K}) \tag{9.16}$$

where $\mathbf{K}$ is the kernel matrix computed using $K_{ij} = \mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and $z_n = f(\boldsymbol{x}_n)$ is the unknown function value for input n. Since this model is jointly Gaussian, we can easily compute the exact posterior $p(\boldsymbol{z}|\boldsymbol{y})$ in $O(N^3)$ time. (Faster approximations are also possible, see **??**.)

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 $q(\boldsymbol{z}|\boldsymbol{\lambda}) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{\lambda}^{(1)}, \boldsymbol{\lambda}^{(2)})$.

Since the likelihood term factorizes across data points $n = 1 : N$, we will only need to compute marginals of this variational posterior. From **??** we know that the sufficient statistics and natural parameters of a univariate Gaussian are given by

$$\mathcal{T}(z_n) = [z_n, z_n^2] \tag{9.17}$$

$$\boldsymbol{\lambda}_n = [\frac{m_n}{v_n}, -\frac{1}{2v_n}] \tag{9.18}$$

The corresponding moment parameters are

$$\boldsymbol{\mu}_n = [m_n, m_n^2 + v_n] \tag{9.19}$$

$$m_n = v_n \lambda_n^{(1)} \tag{9.20}$$

$$v_n = \frac{1}{2\lambda_n^{(2)}} \tag{9.21}$$

We need to compute the gradient terms $\nabla_{\boldsymbol{\mu}_n} \mathbb{E}_{\mathcal{N}(z_n|\boldsymbol{\mu}_n)} [\log p(y_n|z_n)]$. We can do this by sampling z_n from the local Gaussian posterior, and then pushing gradients inside, using the results from **??**. 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:

$$\tilde{\lambda}_{n,t}^{(i)} = (1 - \eta_t)\tilde{\lambda}_{n,t-1}^{(i)} + \eta_t \hat{g}_{n,t}^{(i)} \tag{9.22}$$

We can also perform a "doubly stochastic" approximation (as in **??**) by just updating a random subset of these terms. Once we have updated the likelihood, we can update the posterior using

$$q(\boldsymbol{z}|\boldsymbol{\lambda}_{t+1}) \propto \left[\prod_{n=1}^{N} e^{z_n \tilde{\lambda}_{n,t}^{(1)} + z_n^2 \tilde{\lambda}_{n,t}^{(2)}}\right] \mathcal{N}(\boldsymbol{z}|\mathbf{0}, \mathbf{K}) \tag{9.23}$$

$$\propto \left[\prod_{n=1}^{N} \mathcal{N}_c(z_n|\tilde{\lambda}_{n,t}^{(1)}, \tilde{\lambda}_{n,t}^{(2)})\right] \mathcal{N}(\boldsymbol{z}|\mathbf{0}, \mathbf{K}) \tag{9.24}$$

$$= \left[\prod_{n=1}^{N} \mathcal{N}(z_n|\tilde{m}_{n,t}, \tilde{v}_{n,t})\right] \mathcal{N}(\boldsymbol{z}|\mathbf{0}, \mathbf{K}) \tag{9.25}$$

$$= \left[\prod_{n=1}^{N} \mathcal{N}(\tilde{m}_{n,t}|z_n, \tilde{v}_{n,t})\right] \mathcal{N}(\boldsymbol{z}|\mathbf{0}, \mathbf{K}) \tag{9.26}$$

where $\tilde{m}_{n,t}$ and $\tilde{v}_{n,t}$ are derived from $\tilde{\boldsymbol{\lambda}}_{n,t}$. We can think of this as **Gaussianizing the likelihood** at each step, where we replace the observations y_n by **pseudo-observations** $\tilde{m}_{n,t}$ and use a variational variance $\tilde{v}_{n,t}$. This lets us use exact GP regression updates in the inner loop. See [SKZ19; Cha+20] for details.

9.0.1.2 Example: Bayesian logistic regression

In this section, we discuss how to compute a Gaussian approximation to $p(\boldsymbol{w}|\mathcal{D})$ for a binary logistic regression model with a Gaussian prior on the weights. We will use CVI in which we "Gaussianize" the likelihoods, and then perform closed form Bayesian linear regression in the inner loop. This is similar to the approach used in Section 14.1, where we derive a quadratic lower bound to the log likelihood. However, such "local VI" methods are not guaranteed to converge to a local maximum of the ELBO [Kha12], unlike the CVI method.

The joint distribution has the form

$$p(\boldsymbol{y}_{1:N}, \boldsymbol{w}|\mathbf{X}) = \left[\prod_{n=1}^{N} p(y_n|z_n)\right] \mathcal{N}(\boldsymbol{w}|\mathbf{0}, \delta\mathbf{I}) \tag{9.27}$$

where $z_n = \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n$ is the local latent, and $\delta > 0$ is the prior variance (analogous to an ℓ_2 regularizer). We compute the local Gaussian likelihood terms $\tilde{\boldsymbol{\lambda}}_n$ as in in Section 9.0.1.1. We then have the following variational joint:

$$q(\boldsymbol{w}|\boldsymbol{\lambda}_{t+1}) \propto \left[\prod_{n=1}^{N} \mathcal{N}(\tilde{m}_{n,t}|\boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n, \tilde{v}_{n,t})\right] \mathcal{N}(\boldsymbol{w}|\boldsymbol{0}, \delta\mathbf{I}) \tag{9.28}$$

This corresponds to a Bayesian linear regression problem with pseudo-observations $\tilde{m}_{n,t}$ and variational variance $\tilde{v}_{n,t}$.

9.0.1.3 Example: Kalman smoothing with GLM likelihoods

We can extend the above examples to perform posterior inference in a linear-Gaussian state-space model (**??**) with generalized linear model (GLM) likelihoods: we alternate between Gaussianizing the likelihoods and running the Kalman smoother (**??**).

9.1 Exact and approximate inference for PGMs

In this section, we discuss exact and approximate inference for discrete PGMs from a variational perspective, following [WJ08].

Similar to Section 8.7, we will assume a pairwise MRF of the form

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) = \frac{1}{Z}\exp\left\{\sum_{s\in\mathcal{V}}\theta_s(z_s) + \sum_{(s,t)\in\mathcal{E}}\theta_{st}(z_s, z_t)\right\} \tag{9.29}$$

We can write this as an exponential family model, $p(\boldsymbol{z}|\boldsymbol{x}) = \tilde{p}(\boldsymbol{z})/Z$, where $Z = \log p(\boldsymbol{x})$, $\tilde{p}(\boldsymbol{z}) = \mathcal{T}(\boldsymbol{z})^\mathsf{T}\boldsymbol{\theta}$, $\boldsymbol{\theta} = (\{\theta_{s;j}\}, \{\theta_{s,t;j,k}\})$ are all the node and edge parameters (the canonical parameters), and $\mathcal{T}(\boldsymbol{z}) = (\{\mathbb{I}(z_s = j)\}, \{\mathbb{I}(z_s = j, z_t = k)\})$ 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.

9.1.1 Exact inference as VI

We know that the ELBO is a lower bound on the log marginal likelihood:

$$\mathcal{L}(q) = \mathbb{E}_{q(\boldsymbol{z})}[\log\tilde{p}(\boldsymbol{z})] + \mathbb{H}(q) \leq \log Z \tag{9.30}$$

Let $\boldsymbol{\mu} = \mathbb{E}_q[\mathcal{T}(\boldsymbol{z})]$ be the mean parameters of the variational distribution. Then we can rewrite this as

$$\mathcal{L}(\boldsymbol{\mu}) = \boldsymbol{\theta}^\mathsf{T}\boldsymbol{\mu} + \mathbb{H}(\boldsymbol{\mu}) \leq \log Z \tag{9.31}$$

The set of all valid (unrestricted) mean parameters $\boldsymbol{\mu}$ is the **marginal polytope** corresponding to the graph, $\mathbb{M}(G)$, as explained in Section 8.7.2. Optimizing over this set recovers $q = p$, and hence

$$\max_{\boldsymbol{\mu}\in\mathbb{M}(G)} \boldsymbol{\theta}^T\boldsymbol{\mu} + \mathbb{H}(\boldsymbol{\mu}) = \log Z \tag{9.32}$$

Equation (9.32) seems easy to optimize: the objective is concave, since it is the sum of a linear function and a concave function (see Figure **??** 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 9.1 for a high level summary of the methods we discuss.

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 9.1.1
Mean field	$\max_{\boldsymbol{\mu} \in \mathbb{M}_F(G)} \boldsymbol{\theta}^T \boldsymbol{\mu} + \mathbb{H}_{\mathrm{MF}}(\boldsymbol{\mu}) \leq \log Z$	Concave	Nonconvex inner approx.	Section 9.1.2
Loopy BP	$\max_{\boldsymbol{\tau} \in \mathbb{L}(G)} \boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{\mathrm{Bethe}}(\boldsymbol{\tau}) \approx \log Z$	Non-concave	Convex outer approx.	Section 9.1.3
TRBP	$\max_{\boldsymbol{\tau} \in \mathbb{L}(G)} \boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{\mathrm{TRBP}}(\boldsymbol{\tau}) \geq \log Z$	Concave	Convex outer approx.	Section 9.1.5

Table 9.1: Summary of some variational inference methods for graphical models. TRBP is tree-reweighted belief propagation.

9.1.2 Mean field VI

The mean field approximation to the entropy is simply

$$\mathbb{H}_{\mathrm{MF}}(\boldsymbol{\mu}) = \sum_s \mathbb{H}(\boldsymbol{\mu}_s) \tag{9.33}$$

which follows from the factorization assumption. Thus the mean field objective is

$$\mathcal{L}_{\mathrm{MF}}(\boldsymbol{\mu}) = \boldsymbol{\theta}^{\mathsf{T}} \boldsymbol{\mu} + \mathbb{H}_{\mathrm{MF}}(\boldsymbol{\mu}) \leq \log Z \tag{9.34}$$

This is a concave lower bound on $\log Z$. We will maximize this over a 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:

$$\Omega(F) \triangleq \{\boldsymbol{\theta} \in \Omega : \boldsymbol{\theta}_\alpha = 0 \; \forall \alpha \in \mathcal{I} \setminus \mathcal{I}(F)\} \tag{9.35}$$

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

$$\Omega(F_0) \triangleq \{\boldsymbol{\theta} \in \Omega : \boldsymbol{\theta}_{st} = 0 \; \forall (s,t) \in E\} \tag{9.36}$$

In the case of structured mean field (Section **??**), 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:

$$\mathbb{M}_F(G) \triangleq \{\boldsymbol{\mu} \in \mathbb{R}^d : \boldsymbol{\mu} = \mathbb{E}_{\boldsymbol{\theta}}[\mathcal{T}(\boldsymbol{z})] \quad \text{for some } \boldsymbol{\theta} \in \Omega(F)\} \tag{9.37}$$

This is called an **inner approximation** to the marginal polytope, since $\mathbb{M}_F(G) \subseteq \mathbb{M}(G)$. See Figure 8.13(b) for a sketch. Note that $\mathbb{M}_F(G)$ is a non-convex polytope, which results in multiple local optima.

Thus the mean field problem becomes

$$\max_{\boldsymbol{\mu} \in \mathbb{M}_F(G)} \boldsymbol{\theta}^{\mathsf{T}} \boldsymbol{\mu} + \mathbb{H}_{\mathrm{MF}}(\boldsymbol{\mu}) \tag{9.38}$$

This requires maximizing a concave objective over a non-convex set. It is typically optimized using coordinate ascent, since it is easy to optimize a scalar concave function over the marginal distribution for each node.

9.1.3 Loopy belief propagation as VI

Recall from Section 9.1.1 that exact inference can be posed as solving the following optimization problem: $\max_{\boldsymbol{\mu} \in \mathbb{M}(G)} \boldsymbol{\theta}^T \boldsymbol{\mu} + \mathbb{H}(\boldsymbol{\mu})$, where $\mathbb{M}(G)$ is the marginal polytope corresponding to the graph (see Section 8.7.2 for details). Since this set has exponentially many facets, it is intractable to optimize over.

In Section 9.1.2, we discussed the mean field approximation, which uses a nonconvex inner approximation, $\mathbb{M}_F(G)$, obtained by dropping some edges from the graphical model, thus enforcing a factorization of the posterior. We also approximated the entropy by using the entropy of each marginal.

In this section, we will consider a convex outer approximation, $\mathbb{L}(G)$, based on pseudo marginals, as in Section 8.7.3.1. We also need to approximate the entropy (which was not needed when performing MAP estimation, discussed in Section 8.7.3). We discuss this entropy approximation in Section 9.1.3.1, and then show how we can use this to approximate $\log Z$. Finally we show that loopy belief propagation attempts to optimize this approximation.

9.1.3.1 Bethe free energy

From Equation (8.183), we know that a joint distribution over a tree-structured graphical model can be represented exactly by the following:

$$p_{\boldsymbol{\mu}}(\boldsymbol{x}) = \prod_{s \in V} \mu_s(x_s) \prod_{(s,t) \in E} \frac{\mu_{st}(x_s, x_t)}{\mu_s(x_s)\mu_t(x_t)} \tag{9.39}$$

This satisfies the normalization and pairwise marginalization constraints of the outer approximation by construction.

From Equation 9.39, we can write the exact entropy of any tree structured distribution $\boldsymbol{\mu} \in \mathbb{M}(T)$ as follows:

$$\mathbb{H}(\boldsymbol{\mu}) = \sum_{s \in V} H_s(\mu_s) - \sum_{(s,t) \in E} I_{st}(\mu_{st}) \tag{9.40}$$

$$H_s(\mu_s) = -\sum_{x_s \in \mathcal{X}_s} \mu_s(x_s) \log \mu_s(x_s) \tag{9.41}$$

$$I_{st}(\mu_{st}) = \sum_{(x_s, x_t) \in \mathcal{X}_s \times \mathcal{X}_t} \mu_{st}(x_s, x_t) \log \frac{\mu_{st}(x_s, x_t)}{\mu_s(x_s)\mu_t(x_t)} \tag{9.42}$$

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:

$$\mathbb{H}(\boldsymbol{\mu}) = -\sum_{s \in V} (d_s - 1) H_s(\mu_s) + \sum_{(s,t) \in E} H_{st}(\mu_{st}) \tag{9.43}$$

where d_s is the degree (number of neighbors) for node s.

The **Bethe**[2] approximation to the entropy is simply the use of Equation 9.40 even when we don't have a tree:

$$\mathbb{H}_{\text{Bethe}}(\boldsymbol{\tau}) = \sum_{s \in V} H_s(\tau_s) - \sum_{(s,t) \in E} I_{st}(\tau_{st}) \tag{9.44}$$

We define the **Bethe free energy** as the expected energy minus approximate entropy:

$$\mathcal{F}_{\text{Bethe}}(\boldsymbol{\tau}) \triangleq -\left[\boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{\text{Bethe}}(\boldsymbol{\tau})\right] \approx -\log Z \tag{9.45}$$

Thus our final objective becomes

$$\max_{\boldsymbol{\tau} \in \mathbb{L}(G)} \boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{\text{Bethe}}(\boldsymbol{\tau}) \tag{9.46}$$

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 [SWW08]. In Section 9.1.5, we discuss how to modify the algorithm so it always minimizes an upper bound for any model.

[2] Hans Bethe was a German-American physicist, 1906–2005.

9.1.3.2 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 $t \to s$. We can now write the Lagrangian as

$$\begin{aligned}\mathcal{L}(\boldsymbol{\tau}, \boldsymbol{\lambda}; \boldsymbol{\theta}) \triangleq \boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{\text{Bethe}}(\boldsymbol{\tau}) + \sum_s \lambda_{ss} C_{ss}(\boldsymbol{\tau}) \\ + \sum_{s,t} \left[\sum_{x_s} \lambda_{ts}(x_s) C_{ts}(x_s; \boldsymbol{\tau}) + \sum_{x_t} \lambda_{st}(x_t) C_{st}(x_t; \boldsymbol{\tau}) \right] \end{aligned} \tag{9.47}$$

(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

$$\log \tau_s(x_s) = \lambda_{ss} + \theta_s(x_s) + \sum_{t \in \text{nbr}(s)} \lambda_{ts}(x_s) \tag{9.48}$$

$$\log \frac{\tau_{st}(x_s, x_t)}{\tilde{\tau}_s(x_s) \tilde{\tau}_t(x_t)} = \theta_{st}(x_s, x_t) - \lambda_{ts}(x_s) - \lambda_{st}(x_t) \tag{9.49}$$

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 implies $\tilde{\tau}_s(x_s) = \tau_s(x_s)$, we get

$$\begin{aligned}\log \tau_{st}(x_s, x_t) = \lambda_{ss} + \lambda_{tt} + \theta_{st}(x_s, x_t) + \theta_s(x_s) + \theta_t(x_t) \\ + \sum_{u \in \text{nbr}(s) \setminus t} \lambda_{us}(x_s) + \sum_{u \in \text{nbr}(t) \setminus s} \lambda_{ut}(x_t) \end{aligned} \tag{9.50}$$

To make the connection to message passing, define $m_{t \to s}(x_s) = \exp(\lambda_{ts}(x_s))$. With this notation, we can rewrite the above equations (after taking exponents of both sides) as follows:

$$\tau_s(x_s) \propto \exp(\theta_s(x_s)) \prod_{t \in \text{nbr}(s)} m_{t \to s}(x_s) \tag{9.51}$$

$$\begin{aligned}\tau_{st}(x_s, x_t) \propto \exp\left(\theta_{st}(x_s, x_t) + \theta_s(x_s) + \theta_t(x_t)\right) \\ \times \prod_{u \in \text{nbr}(s) \setminus t} m_{u \to s}(x_s) \prod_{u \in \text{nbr}(t) \setminus s} m_{u \to t}(x_t) \end{aligned} \tag{9.52}$$

where the λ terms and irrelevant constants are absorbed into the constant of proportionality. We see that this is equivalent to the usual expression for the node and edge marginals in LBP.

To derive an equation for the messages in terms of other messages (rather than in terms of λ_{ts}), we enforce the marginalization condition $\sum_{x_t} \tau_{st}(x_s, x_t) = \tau_s(x_s)$. Then one can show that

$$m_{t \to s}(x_s) \propto \sum_{x_t} \left[\exp\left\{\theta_{st}(x_s, x_t) + \theta_t(x_t)\right\} \prod_{u \in \text{nbr}(t) \setminus s} m_{u \to t}(x_t) \right] \tag{9.53}$$

We see that this is equivalent to the usual expression for the messages in LBP.

9.1.3.3 Kikuchi free energy

We have shown that LBP minimizes the Bethe free energy. In this section, we show that generalized BP (**??**) minimizes the **Kikuchi free energy**; we define this below, but the key idea is that it is a tighter approximation to $\log Z$.

In more detail, define $\mathbb{L}_t(G)$ to be the set of all pseudo-marginals such that normalization and marginalization constraints hold on a hyper-graph whose largest hyper-edge is of size $t+1$. For example, in Figure **??**, we impose constraints of the form

$$\sum_{x_1,x_2} \tau_{1245}(x_1,x_2,x_4,x_5) = \tau_{45}(x_4,x_5), \quad \sum_{x_6} \tau_{56}(x_5,x_6) = \tau_5(x_5), \ldots \tag{9.54}$$

Furthermore, we approximate the entropy as follows:

$$\mathbb{H}_{\text{Kikuchi}}(\boldsymbol{\tau}) \triangleq \sum_{g \in E} c(g) H_g(\tau_g) \tag{9.55}$$

where $H_g(\tau_g)$ is the entropy of the joint (pseudo) distribution on the vertices in set g, and $c(g)$ is called the **overcounting number** of set g. These are related to **Mobius numbers** in set theory. Rather than giving a precise definition, we just give a simple example. For the graph in Figure **??**, we have

$$\mathbb{H}_{\text{Kikuchi}}(\boldsymbol{\tau}) = -[H_{1245} + H_{2356} + H_{4578} + H_{5689}] - [H_{25} + H_{45} + H_{56} + H_{58}] + H_5 \tag{9.56}$$

Putting these two approximations together, we can define the **Kikuchi free energy**[3] as follows:

$$\mathcal{F}_{\text{Kikuchi}}(\boldsymbol{\tau}) \triangleq -\left[\boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{\text{Kikuchi}}(\boldsymbol{\tau})\right] \approx -\log Z \tag{9.57}$$

Our variational problem becomes

$$\max_{\boldsymbol{\tau} \in \mathbb{L}(G)} \boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{\text{Kikuchi}}(\boldsymbol{\tau}) \tag{9.58}$$

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., [WJ08, Sec 4.2] or [KF09, Sec 11.3.2].

9.1.4 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 (9.32) is a concave objective maximized over a convex set, it is natural to try to come up with an appproximation 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:

$$\mathbb{H}(\boldsymbol{\mu}(G)) \leq \sum_{F \in \mathcal{F}} \rho(F) \mathbb{H}(\boldsymbol{\mu}(F)) \triangleq \mathbb{H}(\boldsymbol{\mu}, \rho) \tag{9.59}$$

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:

$$\mathbb{L}(G; \mathcal{F}) \triangleq \{\boldsymbol{\tau} \in \mathbb{R}^d : \boldsymbol{\tau}(F) \in \mathbb{M}(F) \; \forall F \in \mathcal{F}\} \tag{9.60}$$

This is a convex set since each $\mathbb{M}(F)$ is a projection of a convex set. Hence we define our problem as

$$\max_{\boldsymbol{\tau} \in \mathbb{L}(G;\mathcal{F})} \boldsymbol{\tau}^T \boldsymbol{\theta} + \mathbb{H}(\boldsymbol{\tau}, \rho) \tag{9.61}$$

[3] Ryoichi Kikuchi is a Japanese physicist.

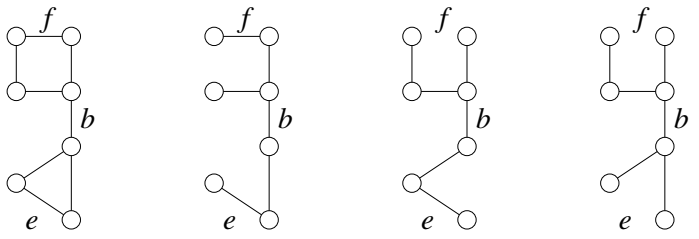


Figure 9.1: (a) A graph. (b-d) Some of its spanning trees. From Figure 7.1 of [WJ08]. Used with kind permission of Martin Wainwright.

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.

9.1.5 Tree-reweighted belief propagation

In this section, we discuss **tree reweighted BP** [WJW05b; Kol06], which is a form of convex BP which uses spanning trees as the set of tractable models $\mathcal{F}$, as we describe below.

9.1.5.1 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 9.40. 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:

$$\mathbb{H}(\boldsymbol{\mu}) \leq \sum_{s \in V} H_s(\mu_s) - \sum_{(s,t) \in E} \rho_{st} I_{st}(\mu_{st}) \triangleq \mathbb{H}_{\text{TRBP}}(\boldsymbol{\mu}) \tag{9.62}$$

This is called the **tree reweighted BP** approximation [WJW05b; Kol06]. 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 9.1 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 [WJW05a]. 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:

$$\max_{\boldsymbol{\tau} \in \mathbb{L}(G)} \boldsymbol{\tau}^T \boldsymbol{\theta} + \mathbb{H}_{\text{TRBP}}(\boldsymbol{\tau}) \geq \log Z \tag{9.63}$$

9.1.5.2 Message passing implementation

The simplest way to minimize Equation (9.63) 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 [WJ08, Sec 7.2.1]:

$$m_{t\to s}(x_s) \propto \sum_{x_t} \exp\left(\frac{1}{\rho_{st}}\theta_{st}(x_s,x_t) + \theta_t(x_t)\right) \frac{\prod_{v\in \text{nbr}(t)\setminus s}[m_{v\to t}(x_t)]^{\rho_{vt}}}{[m_{s\to t}(x_t)]^{1-\rho_{ts}}} \tag{9.64}$$

At convergence, the node and edge pseudo marginals are given by

$$\tau_s(x_s) \propto \exp(\theta_s(x_s)) \prod_{v\in \text{nbr}(s)} [m_{v\to s}(x_s)]^{\rho_{vs}} \tag{9.65}$$

$$\tau_{st}(x_s,x_t) \propto \varphi_{st}(x_s,x_t) \frac{\prod_{v\in \text{nbr}(s)\setminus t}[m_{v\to s}(x_s)]^{\rho_{vs}}}{[m_{t\to s}(x_s)]^{1-\rho_{st}}} \frac{\prod_{v\in \text{nbr}(t)\setminus s}[m_{v\to t}(x_t)]^{\rho_{vt}}}{[m_{s\to t}(x_t)]^{1-\rho_{ts}}} \tag{9.66}$$

$$\varphi_{st}(x_s,x_t) \triangleq \exp\left(\frac{1}{\rho_{st}}\theta_{st}(x_s,x_t) + \theta_s(x_s) + \theta_t(x_t)\right) \tag{9.67}$$

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 [HS08], but in practice, using damped updates as in Equation **??** is often sufficient to ensure convergence.

9.1.5.3 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 (9.64) (see [WJ08, Sec 8.4.3] for details). This is guaranteed to converge to the LP relaxation discussed in Section 8.7.3 under a suitable scheduling known as **sequential tree-reweighted message passing** [Kol06].

9.1.6 Other tractable versions of convex BP

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 $p(\boldsymbol{x}) \propto \exp(\sum_{(s,t)\in E} \theta_{st} x_s x_t)$. 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 [GJ07] for details, and [Sch10] for a related *exact* method for planar Ising models.

Chapter 10

Monte Carlo Inference

Chapter 11

Markov Chain Monte Carlo (MCMC) inference

Chapter 12

Sequential Monte Carlo (SMC) inference

Part III

Prediction

Chapter 13

Predictive models: an overview

Chapter 14

Generalized linear models

14.1 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, $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}_0, \mathbf{V}_0)$. We will create a "Gaussian-like" lower bound to the likelihood, which becomes conjugate to this prior. We then iteratively improve this lower bound.

14.1.1 Binary logistic regression

In this section, we discuss VI for binary logistic regression. Our presentation follows [Bis06, Sec 10.6].

Let us first rewrite the likelihood for a single observation as follows:

$$p(y_n|\boldsymbol{x}_n, \boldsymbol{w}) = \sigma(\eta_n)^{y_n}(1-\sigma(\eta_n))^{1-\eta_n} \tag{14.1}$$

$$= \left(\frac{1}{1+e^{-\eta_n}}\right)^{y_n}\left(1-\frac{1}{1+e^{-\eta_n}}\right)^{1-y_n} \tag{14.2}$$

$$= e^{-\eta_n y_n}\frac{e^{-\eta_n}}{1+e^{-\eta_n}} = e^{-\eta_n y_n}\sigma(-\eta_n) \tag{14.3}$$

where $\eta_n = \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n$ 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 [JJ96; JJ00]:

$$\sigma(\eta_n) \geq \sigma(\boldsymbol{\psi}_n)\exp\left[(\eta_n - \boldsymbol{\psi}_n)/2 - \lambda(\boldsymbol{\psi}_n)(\eta_n^2 - \boldsymbol{\psi}_n^2)\right] \tag{14.4}$$

where $\boldsymbol{\psi}_n$ is the variational parameter for datapoint n, and

$$\lambda(\boldsymbol{\psi}) \triangleq \frac{1}{4\boldsymbol{\psi}}\tanh(\boldsymbol{\psi}/2) = \frac{1}{2\boldsymbol{\psi}}\left[\sigma(\boldsymbol{\psi}) - \frac{1}{2}\right] \tag{14.5}$$

We shall refer to this as the **JJ bound**, after its inventors, Jaakkola and Jordan. See Figure 14.1(a) for a plot, and see Section 6.4.4.2 for a derivation.

Using this bound, we can write

$$p(y_n|\boldsymbol{x}_n, \boldsymbol{w}) = e^{-\eta_n y_n}\sigma(-\eta_n) \geq e^{-\eta_n y_n}\sigma(\boldsymbol{\psi}_n)\exp\left[(-\eta_n + \boldsymbol{\psi}_n)/2 - \lambda(\boldsymbol{\psi}_n)(\eta_n^2 - \boldsymbol{\psi}_n^2)\right] \tag{14.6}$$

We can now lower bound the log joint as follows:

$$\log p(\boldsymbol{y}|\mathbf{X}, \boldsymbol{w}) + \log p(\boldsymbol{w}) \geq -\frac{1}{2}(\boldsymbol{w}-\boldsymbol{\mu}_0)^\mathsf{T}\mathbf{V}_0^{-1}(\boldsymbol{w}-\boldsymbol{\mu}_0) \tag{14.7}$$

$$+ \sum_{n=1}^{N}\left[\eta_n(y_n - 1/2) - \lambda(\boldsymbol{\psi}_n)\boldsymbol{w}^\mathsf{T}(\boldsymbol{x}_n\boldsymbol{x}_n^\mathsf{T})\boldsymbol{w}\right] \tag{14.8}$$

(a) (b)

Figure 14.1: 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 14.1.2.2). This is tight at $\psi = 2.5$. Generated by sigmoid_lower_bounds.py.

Since this is a quadratic function of $\boldsymbol{w}$, we can derive a Gaussian posterior approximation as follows:

$$q(\boldsymbol{w}|\boldsymbol{\psi}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}_N, \mathbf{V}_N) \tag{14.9}$$

$$\boldsymbol{\mu}_N = \mathbf{V}_N \left(\mathbf{V}_0^{-1}\boldsymbol{\mu}_0 + \sum_{n=1}^{N} (y_n - 1/2)\boldsymbol{x}_n \right) \tag{14.10}$$

$$\mathbf{V}_N^{-1} = \mathbf{V}_0^{-1} + 2\sum_{n=1}^{N} \lambda(\boldsymbol{\psi}_n)\boldsymbol{x}_n\boldsymbol{x}_n^{\mathsf{T}} \tag{14.11}$$

This is more flexible than a Laplace approximation, since the variational parameters $\boldsymbol{\psi}$ can be used to optimize the curvature of the posterior covariance. To find the optimal $\boldsymbol{\psi}$, we can maximize the ELBO, which is given by

$$\log p(\boldsymbol{y}|\mathbf{X}) = \log \int p(\boldsymbol{y}|\mathbf{X}, \boldsymbol{w})p(\boldsymbol{w})d\boldsymbol{w} \geq \log \int h(\boldsymbol{w}, \boldsymbol{\psi})p(\boldsymbol{w})d\boldsymbol{w} = Ł(\boldsymbol{\psi}) \tag{14.12}$$

where

$$h(\boldsymbol{w}, \boldsymbol{\psi}) = \prod_{n=1}^{N} \sigma(\boldsymbol{\psi}_n) \exp\left[\eta_n y_n - (\eta_n + \boldsymbol{\psi}_n)/2 - \lambda(\boldsymbol{\psi}_n)(\eta_n^2 - \boldsymbol{\psi}_n^2)\right] \tag{14.13}$$

We can evaluate the lower bound analytically to get

$$Ł(\boldsymbol{\psi}) = \frac{1}{2}\log\frac{|\mathbf{V}_N|}{|\mathbf{V}_0|} + \frac{1}{2}\boldsymbol{\mu}_N^{\mathsf{T}}\mathbf{V}_N^{-1}\boldsymbol{\mu}_N - \frac{1}{2}\boldsymbol{\mu}_0^{\mathsf{T}}\mathbf{V}_0^{-1}\boldsymbol{\mu}_0 + \sum_{n=1}^{N}\left[\log\sigma(\boldsymbol{\psi}_n) - \frac{1}{2}\boldsymbol{\psi}_n + \lambda(\boldsymbol{\psi}_n)\boldsymbol{\psi}_n^2\right] \tag{14.14}$$

If we solve for $\nabla_{\boldsymbol{\psi}} Ł(\boldsymbol{\psi}) = \mathbf{0}$, we get the following iterative update equation for each variational parameter:

$$(\boldsymbol{\psi}_n^{\text{new}})^2 = \boldsymbol{x}_n \mathbb{E}\left[\boldsymbol{w}\boldsymbol{w}^{\mathsf{T}}\right]\boldsymbol{x}_n = \boldsymbol{x}_n\left(\mathbf{V}_N + \boldsymbol{\mu}_N\boldsymbol{\mu}_N^{\mathsf{T}}\right)\boldsymbol{x}_n \tag{14.15}$$

One we have estimated $\boldsymbol{\psi}_n$, we can plug it into the above Gaussian approximation $q(\boldsymbol{w}|\boldsymbol{\psi})$.

14.1.2 Multinomial logistic regression

In this section we discuss how to approximate the posterior $p(\boldsymbol{w}|\mathcal{D})$ for multinomial logistic regression using variational inference, extending the approach of Section 14.1 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 [Boh92]. We can then compute the variational posterior in closed form. This will let us deterministically optimize the ELBO.

Let $\boldsymbol{y}_i \in \{0, 1\}^C$ be a one-hot label vector, and define the logits for example i to be

$$\boldsymbol{\eta}_i = [\boldsymbol{x}_i^{\mathsf{T}}\boldsymbol{w}_1, \ldots, \boldsymbol{x}_i^{\mathsf{T}}\boldsymbol{w}_C] \tag{14.16}$$

If we define $\mathbf{X}_i = \mathbf{I} \otimes \boldsymbol{x}_i$, where $\otimes$ is the kronecker product, and $\mathbf{I}$ is $C \times C$ identity matrix, then we can write the logits as $\boldsymbol{\eta}_i = \mathbf{X}_i \boldsymbol{w}$. (For example, if $C = 2$ and $\boldsymbol{x}_i = [1, 2, 3]$, we have $\mathbf{X}_i = [1, 2, 3, 0, 0, 0; 0, 0, 0, 1, 2, 3]$.) Then the likelihood is given by

$$p(\boldsymbol{y}|\mathbf{X}, \boldsymbol{w}) = \prod_{i=1}^{N} \exp[\boldsymbol{y}_i^\mathsf{T} \boldsymbol{\eta}_i - \text{lse}(\boldsymbol{\eta}_i)] \tag{14.17}$$

where lse() is the log-sum-exp function

$$\text{lse}(\boldsymbol{\eta}_i) \triangleq \log \left(\sum_{c=1}^{C} \exp(\eta_{ic}) \right) \tag{14.18}$$

For identifiability, we can set $\boldsymbol{w}_C = \mathbf{0}$, so

$$\text{lse}(\boldsymbol{\eta}_i) = \log \left(1 + \sum_{m=1}^{M} \exp(\eta_{im}) \right) \tag{14.19}$$

where $M = C - 1$. (We subtract 1 so that in the binary case, $M = 1$.)

14.1.2.1 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 $\boldsymbol{\psi}_i \in \mathbb{R}^M$:

$$\text{lse}(\boldsymbol{\eta}_i) = \text{lse}(\boldsymbol{\psi}_i) + (\boldsymbol{\eta}_i - \boldsymbol{\psi}_i)^\mathsf{T} \boldsymbol{g}(\boldsymbol{\psi}_i) + \frac{1}{2} (\boldsymbol{\eta}_i - \boldsymbol{\psi}_i)^\mathsf{T} \mathbf{H}(\boldsymbol{\psi}_i)(\boldsymbol{\eta}_i - \boldsymbol{\psi}_i) \tag{14.20}$$

$$\boldsymbol{g}(\boldsymbol{\psi}_i) = \exp[\boldsymbol{\psi}_i - \text{lse}(\boldsymbol{\psi}_i)] = \mathcal{S}(\boldsymbol{\psi}_i) \tag{14.21}$$

$$\mathbf{H}(\boldsymbol{\psi}_i) = \text{diag}(\boldsymbol{g}(\boldsymbol{\psi}_i)) - \boldsymbol{g}(\boldsymbol{\psi}_i) \boldsymbol{g}(\boldsymbol{\psi}_i)^\mathsf{T} \tag{14.22}$$

where $\boldsymbol{g}$ and $\mathbf{H}$ are the gradient and Hessian of lse, and $\boldsymbol{\psi}_i \in \mathbb{R}^M$, where $M = C - 1$ is the number of classes minus 1. An upper bound to lse can be found by replacing the Hessian matrix $\mathbf{H}(\boldsymbol{\psi}_i)$ with a matrix $\mathbf{A}_i$ such that $\mathbf{A}_i \succeq \mathbf{H}(\boldsymbol{\psi}_i)$ for all ψ_i. [Boh92] showed that this can be achieved if we use the matrix $\mathbf{A}_i = \frac{1}{2} \left[\mathbf{I}_M - \frac{1}{M+1} \mathbf{1}_M \mathbf{1}_M^\mathsf{T} \right]$. 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 $\boldsymbol{\psi}_i$; 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

$$\text{lse}(\boldsymbol{\eta}_i) \leq \frac{1}{2} \boldsymbol{\eta}_i^\mathsf{T} \mathbf{A}_i \boldsymbol{\eta}_i - \boldsymbol{b}_i^\mathsf{T} \boldsymbol{\eta}_i + c_i \tag{14.23}$$

$$\mathbf{A}_i = \frac{1}{2} \left[\mathbf{I}_M - \frac{1}{M+1} \mathbf{1}_M \mathbf{1}_M^\mathsf{T} \right] \tag{14.24}$$

$$\boldsymbol{b}_i = \mathbf{A}_i \boldsymbol{\psi}_i - \boldsymbol{g}(\boldsymbol{\psi}_i) \tag{14.25}$$

$$c_i = \frac{1}{2} \boldsymbol{\psi}_i^\mathsf{T} \mathbf{A}_i \boldsymbol{\psi}_i - \boldsymbol{g}(\boldsymbol{\psi}_i)^\mathsf{T} \boldsymbol{\psi}_i + \text{lse}(\boldsymbol{\psi}_i) \tag{14.26}$$

where $\boldsymbol{\psi}_i \in \mathbb{R}^M$ is a vector of variational parameters.

We can use the above result to get the following lower bound on the softmax likelihood:

$$\log p(y_i = c|\boldsymbol{x}_i, \boldsymbol{w}) \geq \left[\boldsymbol{y}_i^\mathsf{T} \mathbf{X}_i \boldsymbol{w} - 4 \frac{1}{2} \boldsymbol{w}^\mathsf{T} \mathbf{X}_i \mathbf{A}_i \mathbf{X}_i \boldsymbol{w} + \boldsymbol{b}_i^\mathsf{T} \mathbf{X}_i \boldsymbol{w} - c_i \right]_c \tag{14.27}$$

To simplify notation, define the pseudo-measurement

$$\tilde{\boldsymbol{y}}_i \triangleq \mathbf{A}_i^{-1} (\boldsymbol{b}_i + \boldsymbol{y}_i) \tag{14.28}$$

Then we can get a "Gaussianized" version of the observation model:

$$p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{w}) \geq f(\boldsymbol{x}_i, \boldsymbol{\psi}_i)\ \mathcal{N}(\tilde{\boldsymbol{y}}_i|\mathbf{X}_i\boldsymbol{w}, \mathbf{A}_i^{-1}) \tag{14.29}$$

where $f(\boldsymbol{x}_i, \boldsymbol{\psi}_i)$ is some function that does not depend on $\boldsymbol{w}$. Given this, it is easy to compute the posterior $q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{m}_N, \mathbf{V}_N)$, using Bayes rule for Gaussians.

Given the posterior, we can write the ELBO as follows:

$$\text{Ł}(\boldsymbol{\psi}) \triangleq -D_{\mathbb{KL}}\left(q(\boldsymbol{w}) \parallel p(\boldsymbol{w})\right) + \mathbb{E}_q\left[\sum_{i=1}^N \log p(y_i|\boldsymbol{x}_i, \boldsymbol{w})\right] \tag{14.30}$$

$$= -D_{\mathbb{KL}}\left(q(\boldsymbol{w}) \parallel p(\boldsymbol{w})\right) + \mathbb{E}_q\left[\sum_{i=1}^N \boldsymbol{y}_i^\mathsf{T}\boldsymbol{\eta}_i - \text{lse}(\boldsymbol{\eta}_i)\right] \tag{14.31}$$

$$= -D_{\mathbb{KL}}\left(q(\boldsymbol{w}) \parallel p(\boldsymbol{w})\right) + \sum_{i=1}^N \boldsymbol{y}_i^\mathsf{T}\mathbb{E}_q\left[\boldsymbol{\eta}_i\right] - \sum_{i=1}^N \mathbb{E}_q\left[\text{lse}(\boldsymbol{\eta}_i)\right] \tag{14.32}$$

where $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_0, \mathbf{V}_0)$ is the prior and $q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_N, \mathbf{V}_N)$ is the approximate posterior. The first term is just the KL divergence between two Gaussians, which is given by

$$-D_{\mathbb{KL}}\left(\mathcal{N}(\boldsymbol{m}_N, \mathbf{V}_N) \parallel \mathcal{N}(\boldsymbol{m}_0, \mathbf{V}_0)\right) = -\frac{1}{2}\left[\text{tr}(\mathbf{V}_N\mathbf{V}_0^{-1}) - \log|\mathbf{V}_N\mathbf{V}_0^{-1}|\right.$$
$$\left.+(\boldsymbol{m}_N - \boldsymbol{m}_0)^\mathsf{T}\mathbf{V}_0^{-1}(\boldsymbol{m}_N - \boldsymbol{m}_0) - DM\right] \tag{14.33}$$

where DM is the dimensionality of the Gaussian, and we assume a prior of the form $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{m}_0, \mathbf{V}_0)$, where typically $\boldsymbol{\mu}_0 = \mathbf{0}_{DM}$, and $\mathbf{V}_0$ is block diagonal. The second term is simply

$$\sum_{i=1}^N \boldsymbol{y}_i^\mathsf{T}\mathbb{E}_q\left[\boldsymbol{\eta}_i\right] = \sum_{i=1}^N \boldsymbol{y}_i^\mathsf{T}\tilde{\boldsymbol{m}}_i \tag{14.34}$$

where $\tilde{\boldsymbol{m}}_i \triangleq \mathbf{X}_i\boldsymbol{m}_N$. The final term can be lower bounded by taking expectations of our quadratic upper bound on lse as follows:

$$-\sum_{i=1}^N \mathbb{E}_q\left[\text{lse}(\boldsymbol{\eta}_i)\right] \geq -\frac{1}{2}\text{tr}(\mathbf{A}_i\tilde{\mathbf{V}}_i) - \frac{1}{2}\tilde{\boldsymbol{m}}_i\mathbf{A}_i\tilde{\boldsymbol{m}}_i + \boldsymbol{b}_i^\mathsf{T}\tilde{\boldsymbol{m}}_i - c_i \tag{14.35}$$

where $\tilde{\mathbf{V}}_i \triangleq \mathbf{X}_i\mathbf{V}_N\mathbf{X}_i^\mathsf{T}$. Hence we have

$$\text{Ł}(\boldsymbol{\psi}) \geq -\frac{1}{2}\left[\text{tr}(\mathbf{V}_N\mathbf{V}_0^{-1}) - \log|\mathbf{V}_N\mathbf{V}_0^{-1}| + (\boldsymbol{m}_N - \boldsymbol{m}_0)^\mathsf{T}\mathbf{V}_0^{-1}(\boldsymbol{m}_N - \boldsymbol{m}_0)\right]$$
$$-\frac{1}{2}DM + \sum_{i=1}^N \boldsymbol{y}_i^\mathsf{T}\tilde{\boldsymbol{m}}_i - \frac{1}{2}\text{tr}(\mathbf{A}_i\tilde{\mathbf{V}}_i) - \frac{1}{2}\tilde{\boldsymbol{m}}_i\mathbf{A}_i\tilde{\boldsymbol{m}}_i + \boldsymbol{b}_i^\mathsf{T}\tilde{\boldsymbol{m}}_i - c_i \tag{14.36}$$

We will use coordinate ascent to optimize this lower bound. That is, we update the variational posterior parameters $\mathbf{V}_N$ and $\boldsymbol{m}_N$, and then the variational likelihood parameters $\boldsymbol{\psi}_i$. We leave the detailed derivation as an exercise, and just state the results. We have

$$\mathbf{V}_N = \left(\mathbf{V}_0 + \sum_{i=1}^N \mathbf{X}_i^\mathsf{T}\mathbf{A}_i\mathbf{X}_i\right)^{-1} \tag{14.37}$$

$$\boldsymbol{m}_N = \mathbf{V}_N\left(\mathbf{V}_0^{-1}\boldsymbol{m}_0 + \sum_{i=1}^N \mathbf{X}_i^\mathsf{T}(\boldsymbol{y}_i + \boldsymbol{b}_i)\right) \tag{14.38}$$

$$\boldsymbol{\psi}_i = \tilde{\boldsymbol{m}}_i = \mathbf{X}_i\boldsymbol{m}_N \tag{14.39}$$

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:

$$\mathbf{V}_N = \left(\mathbf{V}_0 + \mathbf{A} \otimes \sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^\mathsf{T} \right)^{-1} \tag{14.40}$$

$$\boldsymbol{m}_N = \mathbf{V}_N \left(\mathbf{V}_0^{-1} \boldsymbol{m}_0 + \sum_{i=1}^{N} (\boldsymbol{y}_i + \boldsymbol{b}_i) \otimes \boldsymbol{x}_i \right) \tag{14.41}$$

where $\otimes$ denotes the kronecker product.

14.1.2.2 Bohning's bound in the binary case

If we have binary data, then $y_i \in \{0, 1\}$, $M = 1$ and $\eta_i = \boldsymbol{w}^\mathsf{T} \boldsymbol{x}_i$ where $\boldsymbol{w} \in \mathbb{R}^D$ is a weight vector (not matrix). In this case, the Bohning bound becomes

$$\log(1 + e^\eta) \leq \frac{1}{2} a \eta^2 - b\eta + c \tag{14.42}$$

$$a = \frac{1}{4} \tag{14.43}$$

$$b = a\psi - (1 + e^{-\psi})^{-1} \tag{14.44}$$

$$c = \frac{1}{2} a \psi^2 - (1 + e^{-\psi})^{-1} \psi + \log(1 + e^\psi) \tag{14.45}$$

It is possible to derive an alternative quadratic bound for this case. as shown in Section 6.4.4.2. This has the following form

$$\log(1 + e^\eta) \leq \lambda(\boldsymbol{\psi})(\eta^2 - \boldsymbol{\psi}^2) + \frac{1}{2}(\eta - \boldsymbol{\psi}) + \log(1 + e^{\boldsymbol{\psi}}) \tag{14.46}$$

$$\lambda(\boldsymbol{\psi}) \triangleq \frac{1}{4\boldsymbol{\psi}} \tanh(\boldsymbol{\psi}/2) = \frac{1}{2\boldsymbol{\psi}} \left[\sigma(\boldsymbol{\psi}) - \frac{1}{2} \right] \tag{14.47}$$

To facilitate comparison with Bohning's bound, let us rewrite the JJ bound as a quadratic form as follows

$$\log(1 + e^\eta) \leq \frac{1}{2} a(\boldsymbol{\psi}) \eta^2 - b(\boldsymbol{\psi})\eta + c(\boldsymbol{\psi}) \tag{14.48}$$

$$a(\boldsymbol{\psi}) = 2\lambda(\boldsymbol{\psi}) \tag{14.49}$$

$$b(\boldsymbol{\psi}) = -\frac{1}{2} \tag{14.50}$$

$$c(\boldsymbol{\psi}) = -\lambda(\boldsymbol{\psi}) \boldsymbol{\psi}^2 - \frac{1}{2}\boldsymbol{\psi} + \log(1 + e^{\boldsymbol{\psi}}) \tag{14.51}$$

The JJ bound has an adaptive curvature term, since a depends on $\boldsymbol{\psi}$. In addition, it is tight at two points, as is evident from Figure 14.1(a). By contrast, the Bohning bound is a constant curvature bound, and is only tight at one point, as is evident from Figure 14.1(b). Nevertheless, the Bohning bound is simpler, and somewhat faster to compute, since $\mathbf{V}_N$ is a constant, independent of the variational parameters $\boldsymbol{\Psi}$.

14.1.2.3 Other bounds

It is possible to devise bounds that are even more accurate than the JJ bound, and which work for the multiclass case, by using a piecewise quadratic upper bound to lse, as described in [MKM11]. By increasing the number of pieces, the bound can be made arbitrarily tight.

It is also possible to come up with approximations that are not bounds. For example, [SF19] gives a simple approximation for the output of a softmax layer when applied to a stochastic input (characterized in terms of its first two moments).

14.2 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 [Tad15].

To see why this relationship is true, we follow the presentation of [McE20, Sec 11.3.3]. 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

$$p(y_1, y_2|m, \mu_1, \mu_2) = \frac{m!}{y_1!y_2!}\mu_1^{y_1}\mu_2^{y_2} \tag{14.52}$$

Now consider a product of two Poisson likelihoods, for each set of counts:

$$p(y_1, y_2|\lambda_1, \lambda_2) = p(y_1|\lambda_1)p(y_2|\lambda_2) = \frac{e^{-\lambda_1}\lambda_1^{y_1}}{y_1!}\frac{e^{-\lambda_2}\lambda_2^{y_2}}{y_2!} \tag{14.53}$$

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

$$p(y_1, y_2|m, \mu_1, \mu_2) = \frac{m!}{y_1!y_2!}\left(\frac{\lambda_1}{\Lambda}\right)^{y_1}\left(\frac{\lambda_2}{\Lambda}\right)^{y_2} = \frac{m!}{\Lambda^{y_1}\Lambda^{y_2}}\frac{\lambda_1^{y_1}}{y_1!}\frac{\lambda_2^{y_2}}{y_2!} \tag{14.54}$$

$$= \frac{m!}{\Lambda^m}\frac{e^{-\lambda_1}}{e^{-\lambda_1}}\frac{\lambda_1^{y_1}}{y_1!}\frac{e^{-\lambda_2}}{e^{-\lambda_2}}\frac{\lambda_2^{y_2}}{y_2!} \tag{14.55}$$

$$= \underbrace{\frac{m!}{e^{-\Lambda}\Lambda^m}}_{p(m)^{-1}}\underbrace{\frac{e^{-\lambda_1}\lambda_1^{y_1}}{y_1!}}_{p(y_1)}\underbrace{\frac{e^{-\lambda_2}\lambda_2^{y_2}}{y_2!}}_{p(y_2)} \tag{14.56}$$

The final expression says that $p(y_1, y_2|m) = p(y_1)p(y_2)/p(m)$, which makes sense.

14.3 Case study: is Berkeley admissions biased against women?

In this section, we consider a simple but interesting example of logistic regression from [McE20, Sec 11.1.4]. The question of interest is whether admission to graduate school at UC Berkeley is biased against women. The dataset comes from a famous paper [BHO75], which collected statistics for 6 departments for men and women. The data table only has 12 rows, shown in Table 14.1, although the total sample size (number of observations) is 4526.

We conduct a regression analysis to try to determine if gender **causes** imbalanced admissions rates. (This is a simple example of a **fairness** analysis.)

14.3.1 Binomial logistic regression

An obvious way to attempt to answer the question of interest is to fit a binomial logistic regression model, in which the outcome is the admissions rate for each row, and the input is the gender id of the corresponding group. One way to write this model is as follows:

$$A_i \sim \text{Bin}(N_i, \mu_i) \tag{14.57}$$

$$\text{logit}(\mu_i) = \alpha + \beta\text{MALE}[i] \tag{14.58}$$

$$\alpha \sim \mathcal{N}(0, 10) \tag{14.59}$$

$$\beta \sim \mathcal{N}(0, 1.5) \tag{14.60}$$

dept	gender	admit	reject	applications
A	male	512	313	825
A	female	89	19	108
B	male	353	207	560
B	female	17	8	25
C	male	120	205	325
C	female	202	391	593
D	male	138	279	417
D	female	131	244	375
E	male	53	138	191
E	female	94	299	393
F	male	22	351	373
F	female	24	317	341

Table 14.1: Admissions data for UC Berkeley from [BHO75].

Here $\text{MALE}[i] = 1$ iff case i refers to male admissions data. So the log odds is α for female cases, and $\alpha + \beta$ for male candidates. (The choice of prior for these parameters is discussed in **??**.)

The above formulation is asymmetric in the genders. In particular, the log odds for males has two random variables associated with it, and hence is a priori is more uncertain. It is often better to rewrite the model in the following symmetric way:

$$A_i \sim \text{Bin}(N_i, \mu_i) \tag{14.61}$$
$$\text{logit}(\mu_i) = \alpha_{\text{GENDER}[i]} \tag{14.62}$$
$$\alpha_j \sim \mathcal{N}(0, 1.5),\ j \in \{1, 2\} \tag{14.63}$$

Here $\text{GENDER}[i]$ is the gender (1 for male, 2 for female), so the log odds is α_1 for males and α_2 for females.

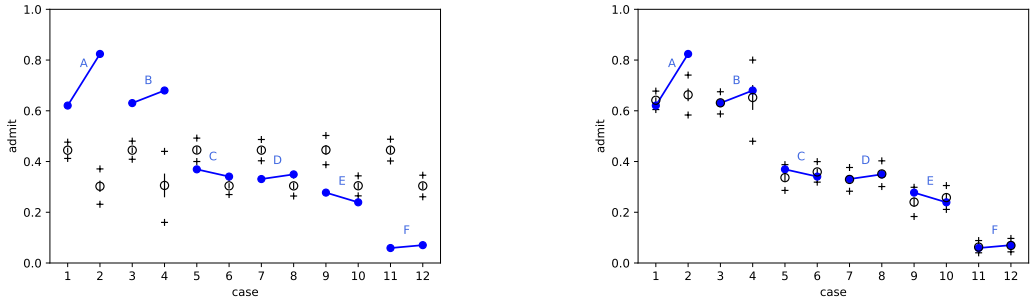
We can perform posterior inference using a variety of methods (see **??**). Here we use HMC (**??**). We find the 89% credible interval for α_1 is $[-0.29, 0.16]$ and for α_2 is $[-0.91, 0.75]$.[1] The corresponding distribution for the difference in probability, $\sigma(\alpha_1) - \sigma(\alpha_2)$, is $[0.12, 0.16]$, with a mean of 0.14. So it seems that Berkeley is biased in favor of men.

However, before jumping to conclusions, we should check if the model is any good. In Figure 14.2a, we plot the posterior predictive distribution, along with the original data. We see the model is a very bad fit to the data (the blue data dots are often outside the black predictive intervals). In particular, we see that the empirical admissions rate for women is actually higher in all the departments except for C and E, yet the model says that women should have a 14% lower chance of admission.

The trouble is that men and women did not apply to the same departments in equal amounts. Women tended not to apply to departments, like A and B, with high admissions rates, but instead applied more to departments, like F, with low admissions rates. So even though less women were accepted *overall*, within in each department, women tended to be accepted at about the same rate.

We can get a better understanding if we consider the DAG in Figure 14.3a. This is intended to be a causal model of the relevant factors. We discuss causality in more detail in **??**, but the basic idea should be clear from this picture. In particular, we see that there is an indirect causal path $G \to D \to A$ from gender to acceptance, so to infer the direct affect $G \to A$, we need to condition on D and close the indirect path.

[1] McElreath uses 89% interval instead of 95% to emphasize the arbitrary nature of these values. The difference is insignificant.



(a) *(b)*

Figure 14.2: Blue dots are admission rates for each of the 6 departments (A-F) for males (left half of each dyad) and females (right half). The circle is the posterior mean of μ_i, the small vertical black lines indicate 1 standard deviation of μ_i. The + marks indicate 95% predictive interval for A_i. (a) Basic model, only taking gender into account. (b) Augmented model, adding department specific offsets. Adapted from Figure 11.5 of [McE20]. Generated by logreg_ucb_admissions_numpyro.ipynb.

We can do this by adding department id as another feature:

$$A_i \sim \text{Bin}(N_i, \mu_i) \tag{14.64}$$
$$\text{logit}(\mu_i) = \alpha_{\text{GENDER}[i]} + \gamma_{\text{DEPT}[i]} \tag{14.65}$$
$$\alpha_j \sim \mathcal{N}(0, 1.5), j \in \{1, 2\} \tag{14.66}$$
$$\gamma_k \sim \mathcal{N}(0, 1.5), k \in \{1, \ldots, 6\} \tag{14.67}$$

Here $j \in \{1, 2\}$ (for gender) and $k \in \{1, \ldots, 6\}$ (for department). Note that there 12 parameters in this model, but each combination (slice of the data) has a fairly large sample size of data associated with it, as we see in Table 14.1.

In Figure 14.2b, we plot the posterior predictive distribution for this new model; we see the fit is now much better. We find the 89% credible interval for α_1 is $[-1.38, 0.35]$ and for α_2 is $[-1.31, 0.42]$. The corresponding distribution for the difference in probability, $\sigma(\alpha_1) - \sigma(\alpha_2)$, is $[-0.05, 0.01]$. So it seems that there is no bias after all.

However, the above conclusion is based on the correctness of the model in Figure 14.3a. What if there are **unobserved confounders** U, such as academic ability, influencing both admission rate and department choice? This hypothesis is shown in Figure 14.3b. In this case, conditioning on the collider D opens up a non-causal path between gender and admissions, $G \to D \leftarrow U \to A$. This invalidates any causal conclusions we may want to draw.

The point of this example is to serve as a cautionary tale to those trying to draw causal conclusions from predictive models. See **??** for more details.

14.3.2 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:

$$y_i \sim \text{BetaBinom}(m_i, \alpha_i, \beta_i) \tag{14.68}$$
$$\alpha_i = \pi_i \kappa \tag{14.69}$$
$$\beta_i = (1 - \pi_i)\kappa \tag{14.70}$$
$$\pi_i = \sigma(\boldsymbol{w}^\mathsf{T} \boldsymbol{x}_i) \tag{14.71}$$

(a) *(b)*

Figure 14.3: Some possible causal models of admissions rates. G is gender, D is department, A is acceptance rate. (a) No hidden confounders. (b) Hidden confounder (small dot) affects both D and A. Generated by logreg_ucb_admissions_numpyro.ipynb.

Note that we have parameterized the model in terms of its mean rate,

$$\pi_i = \frac{\alpha_i}{\alpha_i + \beta_i} \tag{14.72}$$

and shape,

$$\kappa_i = \alpha_i + \beta_i \tag{14.73}$$

We choose to make the mean depend on the inputs (covariates), but to treat the shape (which is like a precision term) as a shared constant.

As we discussed in **??**, the beta-binomial distribution as a continuous mixture distribution of the following form:

$$\text{BetaBinom}(y|m, \alpha, \beta) = \int \text{Bin}(y|m, \mu)\text{Beta}(\mu|\alpha, \beta)d\mu \tag{14.74}$$

In the regression context, we can interpret this as follows: rather than just predicting the mean directly, we predict the mean and variance. This allows for each individual example to have more variability than we might otherwise expect.

If the shape parameter κ is less than 2, then the distribution is an inverted U-shape which strongly favors probabilities of 0 or 1 (see Figure 2.4a). We generally want to avoid this, which we can do by ensuring $\kappa > 2$.

Following [McE20, p371], let us use this model to reanalyze the Berkeley admissions data from Section 14.3. We saw that there was a lot of variability in the outcomes, due to the different admissions rates of each department. Suppose we just regress on the gender, i.e., $\boldsymbol{x}_i = (\mathbb{I}(\text{GENDER}_i = 1), \mathbb{I}(\text{GENDER}_i = 2))$, and $\boldsymbol{w} = (\alpha_1, \alpha_2)$ are the corresponding logits. If we use a binomial regression model, we can be misled into thinking there is gender bias. But if we use the more robust beta-binomial model, we avoid this false conclusion, as we show below.

We fit the following model:

$$A_i \sim \text{BetaBinom}(N_i, \pi_i, \kappa) \tag{14.75}$$

$$\text{logit}(\pi_i) = \alpha_{\text{GENDER}[i]} \tag{14.76}$$

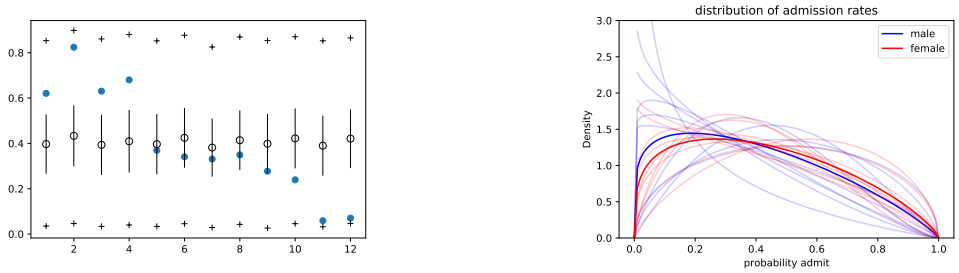
$$\alpha_j \sim \mathcal{N}(0, 1.5) \tag{14.77}$$

$$\kappa = \phi + 2 \tag{14.78}$$

$$\phi \sim \text{Expon}(1) \tag{14.79}$$

(To ensure that $\kappa > 2$, we use a trick and define it as $\kappa = \phi + 2$, where we put an exponential prior (which has a lower bound of 0) on ϕ.)

We fit this model (using HMC) and plot the results in Figure 14.4. In Figure 14.4a, we show the posterior predictive distribution; we see that is quite broad, so the model is no longer overconfident. In Figure 14.4b,



(a) *(b)*

Figure 14.4: 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 [McE20]. Generated by logreg_ucb_admissions_numpyro.ipynb.

we plot $p(\sigma(\alpha_j)|\mathcal{D})$, which is the posterior over the rate of admissions for men and women. We see that there is considerable uncertainty in these value, so now we avoid the false conclusion that one is significantly higher than the other. However, the model is so vague in its predictions as to be useless. In Section 14.3.4, we fix this problem by using a multi-level logistic regression model.

14.3.3 Poisson regression

Let us revisit the Berkeley admissions example from Section 14.3 using Poisson regression. We use a simplified form of the model, in which we just model the outcome counts without using any features, such as gender or department. That is, the model has the form

$$y_{j,n} \sim \text{Poi}(\lambda_j) \tag{14.80}$$
$$\lambda_j = e^{\alpha_j} \tag{14.81}$$
$$\alpha_j \sim \mathcal{N}(0, 1.5) \tag{14.82}$$

for $j = 1:2$ and $n = 1:12$. Let $\overline{\lambda}_i = \mathbb{E}[\lambda_i|\mathcal{D}_i]$, where $\mathcal{D}_1 = \boldsymbol{y}_{1,1:N}$ is the vector of admission counts, and $\mathcal{D}_2 = \boldsymbol{y}_{2,1:N}$ 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

$$\frac{\overline{\lambda}_1}{\overline{\lambda}_1 + \overline{\lambda}_2} = \frac{146.2}{146.2 + 230.9} = 0.38 \tag{14.83}$$

Let us compare this to a binomial regression model of the form

$$y_n \sim \text{Bin}(m_n, \mu) \tag{14.84}$$
$$\mu = \sigma(\alpha) \tag{14.85}$$
$$\alpha \sim \mathcal{N}(0, 1.5) \tag{14.86}$$

Let $\overline{\alpha} = \mathbb{E}[\alpha|\mathcal{D}]$, where $\mathcal{D} = (\boldsymbol{y}_{1,1:N}, \boldsymbol{m}_{1:N})$. The expected acceptance rate across the entire dataset is $\sigma(\overline{\alpha}) = 0.38$, which matches Equation (14.83). (See logreg_ucb_admissions_numpyro.ipynb for the code.)

14.3.4 GLMM (hierarchical Bayes) regression

Let us revisit the Berkeley admissions dataset from Section 14.3, where there are 12 examples, corresponding to male and female admissions to 6 departments. Thus the data is grouped both by gender and department.

Recall that A_i is the number of students admitted in example i, N_i is the number of applicants, μ_i is the expected rate of admissions (the variable of interest), and $\text{DEPT}[i]$ is the department (6 possible values). For pedagogical reasons, we replace the categorical variable $\text{GENDER}[i]$ with the binary indicator $\text{MALE}[i]$. We can create a model with **varying intercept** and **varying slope** as follows:

$$A_i \sim \text{Bin}(N_i, \mu_i) \tag{14.87}$$

$$\text{logit}(\mu_i) = \alpha_{\text{DEPT}[i]} + \beta_{\text{DEPT}[i]} \times \text{MALE}[i] \tag{14.88}$$

This has 12 parameters, as does the original formulation in Equation (14.65). However, these are not independent degrees of freedom. In particular, the intercept and slope are correlated, as we see in Figure 14.2 (higher admissions means steeper slope). We can capture this using the following prior:

$$(\alpha_j, \beta_j) \sim \mathcal{N}(\begin{pmatrix}\overline{\alpha}\\ \overline{\beta}\end{pmatrix}, \boldsymbol{\Sigma}) \tag{14.89}$$

$$\overline{\alpha} \sim \mathcal{N}(0, 4) \tag{14.90}$$

$$\overline{\beta} \sim \mathcal{N}(0, 1) \tag{14.91}$$

$$\boldsymbol{\Sigma} = \text{diag}(\boldsymbol{\sigma})\, \mathbf{R}\, \text{diag}(\boldsymbol{\sigma}) \tag{14.92}$$

$$\mathbf{R} \sim \text{LKJ}(2) \tag{14.93}$$

$$\boldsymbol{\sigma} \sim \prod_{d=1}^{2} \mathcal{N}_+(\sigma_d | 0, 1) \tag{14.94}$$

We can write this more compactly in the following way.[2]. We define $\boldsymbol{u} = (\overline{\alpha}, \overline{\beta})$, and $\boldsymbol{w}_j = (\alpha_j, \beta_j)$, and then use this model:

$$\log(\mu_i) = w_{\text{DEPT}[i]}[0] + w_{\text{DEPT}[i]}[1] \times \text{MALE}[i] \tag{14.95}$$

$$\boldsymbol{w}_j \sim \mathcal{N}(\boldsymbol{u}, \boldsymbol{\Sigma}) \tag{14.96}$$

$$\boldsymbol{u} \sim \mathcal{N}(\mathbf{0}, \text{diag}(4, 1)) \tag{14.97}$$

See Figure 14.5(a) for the graphical model.

Following the discussion in **??**, it is advisable to rewrite the model in a non-centered form. Thus we write

$$\boldsymbol{w}_j = \boldsymbol{u} + \sigma \mathbf{L} \boldsymbol{z}_j \tag{14.98}$$

where $\mathbf{L} = \text{chol}(\mathbf{R})$ is the Cholesy factor for the correlation matrix $\mathbf{R}$, and $\boldsymbol{z}_j \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_2)$. Thus the model becomes the following:[3].

$$\boldsymbol{z}_j \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_2) \tag{14.99}$$

$$\boldsymbol{v}_j = \text{diag}(\boldsymbol{\sigma}) \mathbf{L} \boldsymbol{z}_j \tag{14.100}$$

$$\boldsymbol{u} \sim \mathcal{N}(\mathbf{0}, \text{diag}(4, 1)) \tag{14.101}$$

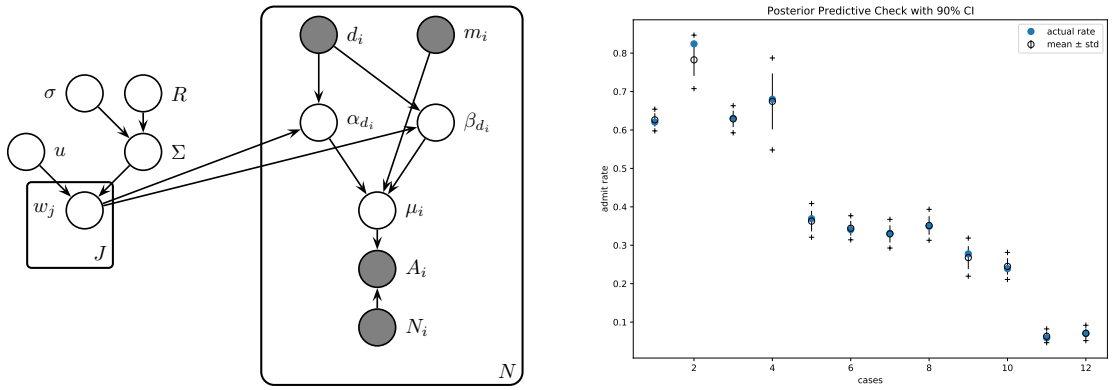
$$\log(\mu_i) = u[0] + v[\text{DEPT}[i], 0] + (u[1] + v[\text{DEPT}[i], 1]) \times \text{MALE}[i] \tag{14.102}$$

This is the version of the model that is implemented in the numypro code.

The results of fitting this model are shown in Figure 14.5(b). The fit is slightly better than in Figure 14.2b, especially for the second column (females in department 2), where the observed value is now inside the predictive interval.

[2] In https://bit.ly/3mP1QWH, this is referred to as glmm4. Note that we use $\boldsymbol{w}$ instead of $\boldsymbol{v}$, and we use $\boldsymbol{u}$ instead of $\boldsymbol{v}_\mu$.

[3] In https://bit.ly/3mP1QWH, this is referred to as glmm5.



(a) *(b)*

Figure 14.5: (a) Generalized linear mixed model for inputs d_i (department) and m_i (male), and output A_i (number of admissionss), given N_i (number of applicants). (b) Results of fitting this model to the UCB dataset. Generated by logreg_ucb_admissions_numpyro.ipynb.

Chapter 15

Deep neural networks

Chapter 16

Bayesian neural networks

Chapter 17

Gaussian processes

Chapter 18

Beyond the iid assumption

Part IV

Generation

Chapter 19

Generative models: an overview

Chapter 20

Variational autoencoders

Chapter 21

Auto-regressive models

Chapter 22

Normalizing flows

Chapter 23

Energy-based models

Chapter 24

Denoising diffusion models

Chapter 25

Generative adversarial networks

Part V

Discovery

Chapter 26

Discovery methods: an overview

Chapter 27

Latent factor models

27.1 Inference in topic models

In this section, we discuss some methods for performing inference in LDA (Latent Dirichlet Allocatiin) models.

27.1.1 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:

$$p(c_{il} = k|\cdot) \propto \exp[\log \pi_{ik} + \log w_{k,x_{il}}] \tag{27.1}$$

$$p(\boldsymbol{\pi}_i|\cdot) = \text{Dir}(\{\alpha_k + \sum_l \mathbb{I}(c_{il} = k)\}) \tag{27.2}$$

$$p(\boldsymbol{w}_k|\cdot) = \text{Dir}(\{\gamma_v + \sum_i \sum_l \mathbb{I}(x_{il} = v, c_{il} = k)\}) \tag{27.3}$$

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 c_{il}'s. This approach was first suggested in [GS04], and is an example of collapsed Gibbs sampling. Figure 27.1(b) shows that now all the c_{il} variables are fully correlated. However, we can sample them one at a time, as we explain below.

First, we need some notation. Let $N_{ivk} = \sum_{l=1}^{L_i} \mathbb{I}(c_{il} = k, x_{il} = v)$ 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

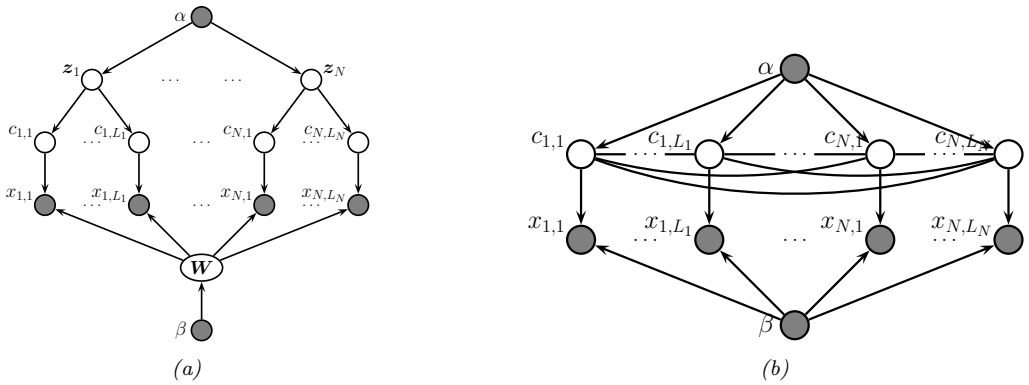
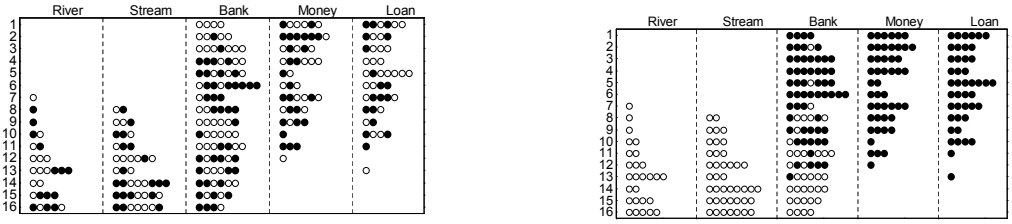


Figure 27.1: (a) LDA unrolled for N documents. (b) Collapsed LDA, where we integrate out the continuous latents $\boldsymbol{z}_n$ and the continuous topic parameters $\mathbf{W}$.



(a) *(b)*

Figure 27.2: 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 [SG07]. Used with kind permission of Tom Griffiths.

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 **??**, one can show that

$$p(\boldsymbol{c}|\alpha) = \prod_i \int \left[\prod_{l=1}^{L_i} \text{Cat}(c_{il}|\boldsymbol{\pi}_i)\right] \text{Dir}(\boldsymbol{\pi}_i|\boldsymbol{\alpha}\mathbf{1}_K) d\boldsymbol{\pi}_i \tag{27.4}$$

$$= \left(\frac{\Gamma(K\alpha)}{\Gamma(\alpha)^K}\right)^N \prod_{i=1}^N \frac{\prod_{k=1}^K \Gamma(N_{ik} + \alpha)}{\Gamma(L_i + K\alpha)} \tag{27.5}$$

By similar reasoning, one can show

$$p(\boldsymbol{x}|\boldsymbol{c}, \beta) = \prod_k \int \left[\prod_{il:c_{il}=k} \text{Cat}(x_{il}|\boldsymbol{w}_k)\right] \text{Dir}(\boldsymbol{w}_k|\beta\mathbf{1}_V) d\boldsymbol{w}_k \tag{27.6}$$

$$= \left(\frac{\Gamma(V\beta)}{\Gamma(\beta)^V}\right)^K \prod_{k=1}^K \frac{\prod_{v=1}^V \Gamma(N_{vk} + \beta)}{\Gamma(N_k + V\beta)} \tag{27.7}$$

From the above equations, and using the fact that $\Gamma(x+1)/\Gamma(x) = x$, we can derive the full conditional for $p(c_{il}|\boldsymbol{c}_{-i,l})$. Define N^-_{ivk} to be the same as N_{ivk} except it is computed by summing over all locations in document i except for c_{il}. Also, let $x_{il} = v$. Then

$$p(c_{i,l} = k|\boldsymbol{c}_{-i,l}, \boldsymbol{y}, \alpha, \beta) \propto \frac{N^-_{v,k} + \beta}{N^-_k + V\beta} \; \frac{N^-_{i,k} + \alpha}{L_i + K\alpha} \tag{27.8}$$

We see that a word in a document is assigned to a topic based both on how often that word is generated by the topic (first term), and also on how often that topic is used in that document (second term).

Given Equation (27.8), we can implement the collapsed Gibbs sampler as follows. We randomly assign a topic to each word, $c_{il} \in \{1, \ldots, K\}$. We can then sample a new topic as follows: for a given word in the corpus, decrement the relevant counts, based on the topic assigned to the current word; draw a new topic from Equation (27.8), update the count matrices; and repeat. This algorithm can be made efficient since the count matrices are very sparse [Li+14].

This process is illustrated in Figure 27.2 on a small example with two topics, and five words. The left part of the figure illustrates 16 documents that were sampled from the LDA model using $p(\text{money}|k = 1) = p(\text{loan}|k = 1) = p(\text{bank}|k = 1) = 1/3$ and $p(\text{river}|k = 2) = p(\text{stream}|k = 2) = p(\text{bank}|k = 2) = 1/3$. For example, we see that the first document contains the word "bank" 4 times (indicated by the four dots in

row 1 of the "bank" column), as well as various other financial terms. The right part of the figure shows the state of the Gibbs sampler after 64 iterations. The "correct" 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.

27.1.2 Variational inference for LDA

A faster alternative to MCMC is to use variational EM, which we discuss in general terms in **??**. There are several ways to apply this to LDA, which we discuss in the following sections.

27.1.2.1 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 [BNJ03], we will use a fully factorized (mean field) approximation of the form

$$q(\boldsymbol{z}_n, \boldsymbol{s}_n) = \text{Dir}(\boldsymbol{z}_n|\tilde{\boldsymbol{z}}_n) \prod_l \text{Cat}(s_{nl}|\tilde{\mathbf{N}}_{nl}) \tag{27.9}$$

where $\tilde{\boldsymbol{z}}_n$ are the variational parameters for the approximate posterior over $\boldsymbol{z}_n$, 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:

$$\tilde{N}_{nlk} \propto w_{d,k} \exp(\mathbb{E}[\log z_{nk}]) \tag{27.10}$$

where $d = x_{nl}$, and

$$\mathbb{E}[\log z_{nk}] = \psi_k(\tilde{\boldsymbol{z}}_n) \triangleq \Psi(\tilde{z}_{nk}) - \psi(\sum_{k'} \tilde{z}_{nk'}) \tag{27.11}$$

where ψ is the digamma function. The update for $q(\boldsymbol{z}_n)$ is obtained by adding up the expected counts:

$$\tilde{z}_{nk} = \alpha_k + \sum_l \tilde{N}_{nlk} \tag{27.12}$$

The M step is obtained by adding up the expected counts and normalizing:

$$\hat{w}_{dk} \propto \beta_d + \sum_{n=1}^{N} \sum_{l=1}^{L_n} \tilde{N}_{nlk} \mathbb{I}(x_{nl} = d) \tag{27.13}$$

27.1.2.2 Count version

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 to perform inference in the mPCA version of the model, which works with counts; these only take $O(N N_w N_z)$ space, which is a big savings if documents are long. (By contrast, the collapsed Gibbs sampler must work explicitly with the s_{nl} variables.)

Following the discussion in **??**, we will work with the variables $\boldsymbol{z}_n$ and $\mathbf{N}_n$, where $\mathbf{N}_n = [N_{ndk}]$ is the matrix of counts, which can be derived from $\boldsymbol{s}_{n,1:L_n}$. We will again use a fully factorized (mean field) approximation of the form

$$q(\boldsymbol{z}_n, \mathbf{N}_n) = \text{Dir}(\boldsymbol{z}_n|\tilde{\boldsymbol{z}}_n) \prod_d \mathcal{M}(\mathbf{N}_{nd}|x_{nd}, \tilde{N}_{nd}) \tag{27.14}$$

where $x_{nd} = \sum_{l=1}^{L_n} \mathbb{I}(x_{nl} = d)$ is the the total number of times token d occurs in document n.

The E step becomes

$$\tilde{z}_{nk} = \alpha_k + \sum_d x_{nd} \tilde{N}_{ndk} \tag{27.15}$$

$$\tilde{N}_{ndk} \propto w_{dk} \exp(\mathbb{E}[\log z_{nk}]) \tag{27.16}$$

The M step becomes

$$\hat{w}_{dk} \propto \beta_d + \sum_n x_{nd} \tilde{N}_{ndk} \tag{27.17}$$

27.1.2.3 Bayesian version

Algorithm 3: Batch VB for LDA

Input: $\{x_{nd}\}$, N_z, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$;
Estimate $\tilde{w}_{dk}$ using EM for multinomial mixtures;
while *not converged* **do**
  // E step ;
  $a_{dk} = 0$ // expected sufficient statistics;
  for *each document* $n = 1 : N$ **do**
    $(\tilde{\boldsymbol{z}}_n, \tilde{\mathbf{N}}_n) = \text{VB-Estep}(\boldsymbol{x}_n, \tilde{\mathbf{W}}, \boldsymbol{\alpha})$;
    $a_{dk} += x_{nd} \tilde{N}_{ndk}$;
  // M step ;
  for *each topic* $k = 1 : N_z$ **do**
    $\tilde{w}_{dk} = \beta_d + a_{dk}$;
function $(\tilde{\boldsymbol{z}}_n, \tilde{\mathbf{N}}_n) = \text{VB-Estep}(\boldsymbol{x}_n, \tilde{\mathbf{W}}, \boldsymbol{\alpha})$;
Initialize $\tilde{z}_{nk} = \alpha_k$;
repeat
  $\tilde{\boldsymbol{z}}_n^{old} = \tilde{\boldsymbol{z}}_n$, $\tilde{z}_{nk} = \alpha_k$;
  for *each word* $d = 1 : N_w$ **do**
    for *each topic* $k = 1 : N_z$ **do**
      $\tilde{N}_{ndk} = \exp\left(\psi_k(\tilde{\boldsymbol{w}}_d) + \psi_k(\tilde{\boldsymbol{z}}_n^{old})\right)$;
    $\tilde{\mathbf{N}}_{nd} = \text{normalize}(\tilde{\mathbf{N}}_{nd})$;
    $\tilde{\boldsymbol{z}}_n += x_{nd} \tilde{\mathbf{N}}_{nd}$
until *Converged*;

We now modify the algorithm to use variational Bayes (VB) instead of EM, i.e., we infer the parameters as well as the latent variables. There are two advantages to this. First, by setting $\beta \ll 1$, VB will encourage $\mathbf{W}$ to be sparse (as in **??**). Second, we will be able to generalize this to the online learning setting, as we discuss below.

Our new posterior approximation becomes

$$q(\boldsymbol{z}_n, \mathbf{N}_n, \mathbf{W}) = \text{Dir}(\boldsymbol{z}_n | \tilde{\boldsymbol{z}}_n) \prod_d \mathcal{M}(\mathbf{N}_{nd} | x_{nd}, \tilde{N}_{nd}) \prod_k \text{Dir}(\boldsymbol{w}_k | \tilde{\boldsymbol{w}}_k) \tag{27.18}$$

The update for $\tilde{N}_{ndk}$ changes, to the following:

$$\tilde{N}_{ndk} \propto \exp\left(\mathbb{E}[\log w_{dk}] + \mathbb{E}[\log z_{nk}]\right) \tag{27.19}$$

The M step is the same as before:

$$\hat{w}_{dk} \propto \beta_d + \sum_n x_{nd} \tilde{N}_{ndk} \tag{27.20}$$

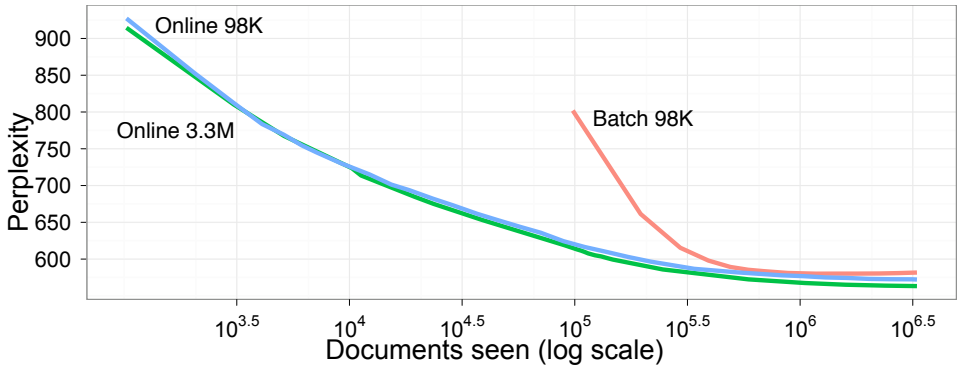


Figure 27.3: Test perplexity vs number of training documents for batch and online VB-LDA. From Figure 1 of [HBB10]. Used with kind permission of David Blei.

No normalization is required, since we are just updating the pseudcounts. The overall algorithm is summarized in Algorithm 3.

27.1.2.4 Online (SVI) version

Algorithm 4: Online VB for LDA

Input: $\{x_{nd}\}$, N_z, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, LR schedule;
Initialize $\tilde{w}_{dk}$ randomly;
for $t = 1 : \infty$ **do**
  Set step size η_t ;
  Pick document n; ;
  $(\tilde{\boldsymbol{z}}_n, \tilde{\mathbf{N}}_n) = \text{VB-Estep}(\boldsymbol{x}_n, \tilde{\mathbf{W}}, \boldsymbol{\alpha})$;
  $\tilde{w}_{dk}^{new} = \beta_d + N x_{nd} \tilde{N}_{ndk}$;
  $\tilde{w}_{dk} = (1 - \eta_t)\tilde{w}_{dk} + \eta_t \tilde{w}_{dk}^{new}$;

In the batch version, the E step takes $O(NN_zN_w)$ per mean field update. This can be slow if we have many documents. This can be reduced by using stochastic variatonal inference, as discussed in **??**. We perform an E step in the usual way. We then compute the variational parameters for $\mathbf{W}$ treating the expected sufficient statistics from the single data case as if the whole data set had those statistics. Finally, we make a partial update for the variational parameters for $\mathbf{W}$, 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 Algorithm 4. In practice, we should use mini-batches, as explained in **??**. In [HBB10], they used a batch of size 256–4096.

Figure 27.3 plots the perplexity on a test set of size 1000 vs number of analyzed documents (E steps), where the data is drawn from (English) Wikipedia. The figure shows that online variational inference is much faster than offline inference, yet produces similar results.

Chapter 28

State-space models

28.1 HMMs: More applications

28.1.1 Spelling correction

In this section, we illustrate how to use an HMM for **spelling correction**. The goal is to infer the sequence of words $\boldsymbol{z}_{1:T}$ that the user meant to type, given observations of what they actually did type, $\boldsymbol{y}_{1:T}$.

28.1.1.1 Baseline model

We start by using a simple unigram language model, so $p(\boldsymbol{z}_{1:T}) = \prod_{1:T} p(z_t)$, where $p(z_t = k)$ is the prior probability of word k being used. These probabilities can be estimated by simply normalizing word frequency counts from a large training corpus. We ignore any Markov structure.

Now we turn to the observation model, $p(y_t = v|z_t = k)$, which is the probability the user types word v when they meant to type word k. For this, we use a **noisy channel model**, in which the "message" z_t gets corrupted by one of four kinds of error: substitution error, where we swap one letter for another (e.g., "government" mistyped as "govermment"); transposition errors, where we swap the order of two adjacent letters (e.g., "government" mistyped as "govermnent"); deletion errors, where we omit one letter (e.g., "government" mistyped as "goverment"); and insertion errors, where we add an extra latter (e.g., "government" mistyped as "governmennt"). If y differs from z by d such errors, we say that y and z have an **edit distance** of d. Let $\mathcal{D}(y, d)$ be the set of words that are edit distance d away from y. We can then define the following likelihood function:

$$p(y|z) = \begin{cases} p_1 & y = z \\ p_2 & y \in \mathcal{D}(z, 1) \\ p_3 & y \in \mathcal{D}(z, 2) \\ p_4 & \text{otherwise} \end{cases} \tag{28.1}$$

where $p_1 > p_2 > p_3 > p_4$.

We can combine the likelihood with the prior to get the overall score for each hypothesis (i.e., candidate correction). This simple model, which was proposed by Peter Norvig[1], can work can quite well. However, it also has some flaws. For example, the error model assumes that the smaller the edit distance, the more likely the word, but this is not always valid. For example, "reciet" gets corrected to "recite" instead of "receipt", and "adres" gets corrected to "acres" not "address". We can fix this problem by learning the parameters of the noise model based on a labeled corpus of (z, x) pairs derived from actual spelling errors. One possible way to get such a corpus is to look at web search behavior: if a user types query q_1 and then quickly changes it to q_2 followed by a click on a link, it suggests that q_2 is a manual correction for q_1, so we can set $(z = q_2, y = q_1)$.

[1] See his excellent tutorial at http://norvig.com/spell-correct.html.

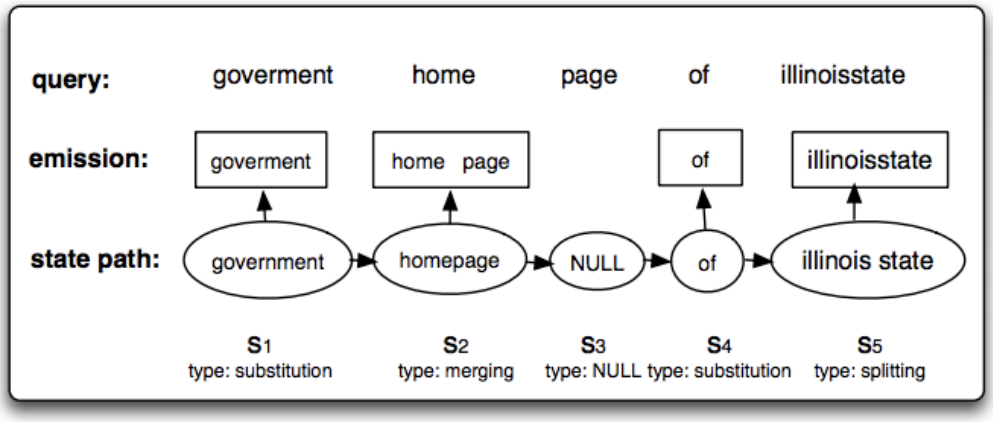


Figure 28.1: Illustration of an HMM applied to spelling correction. The top row, labeled "query", represents the search query $\boldsymbol{y}_{1:T}$ typed by the user, namely "goverment home page of illnoisstate". The bottom row, labeled "state path", represents the most probable assignment to the hidden states, $\boldsymbol{z}_{1:T}$, namely "government homepage of illinois state". (The NULL state is a silent state, that is needed to handle the generation of two tokens from a single hidden state.) The middle row, labeled "emission", represents the words emitted by each state, which match the observed data. From Figure 1 of [LDZ11].

This heuristic has been used in the Etsy search engine.[2] It is also possible to manually collect such data (see e.g., [Hag+17]), or to algorithmically create $(\boldsymbol{z}, \boldsymbol{y})$ pairs, where $\boldsymbol{y}$ is an automatically generated misspelling of $\boldsymbol{z}$ (see e.g., [ECM18]).

28.1.1.2 HMM model

The baseline model can work well, but has room for improvement. In particular, many errors will be hard to correct without context. For example, suppose the user typed "advice": did they mean "advice" or "advise"? It depends on whether they intended to use a noun or a verb, which is hard to tell without looking at the sequence of words. To do this, we will "upgrade" our model to an HMM. We just have to replace our independence prior $p(\boldsymbol{z}_{1:T}) = \prod_t p(z_t)$ by a standard first-order language model on words, $p(\boldsymbol{z}_{1:T}) = \prod_t p(z_t|z_{t-1})$. The parameters of this model can be estimated by counting bigrams in a large corpus of "clean" text (see **??**). The observation model $p(y_t|z_t)$ can remain unchanged.

Given this model, we can compute the top N most likely hidden sequences in $O(NTK^2)$ time, where K is the number of hidden states, and T is the length of the sequence, as explained in **??**. In a naive implementation, the number of hidden states K is the number of words in the vocabulary, which would make the method very slow. However, we can exploit sparsity of the likelihood function (i.e., the fact that $p(y|z)$ is 0 for most values of z) to generate small candidate lists of hidden states for each location in the sequence. This gives us a sparse belief state vector $\boldsymbol{\alpha}_t$.

28.1.1.3 Extended HMM model

We can extend the HMM model to handle higher level errors, in addition to misspellings of individual words. In particular, [LDZ11; LDZ12] proposed modeling the following kinds of errors:

- Two words merged into one, e.g., "home page" → "homepage".
- One word split into two, e.g., "illinoisstate" → "illinois state".

[2] See this blogpost by Mohit Nayyar for details: https://codeascraft.com/2017/05/01/modeling-spelling-correction-for-search-at-etsy/.

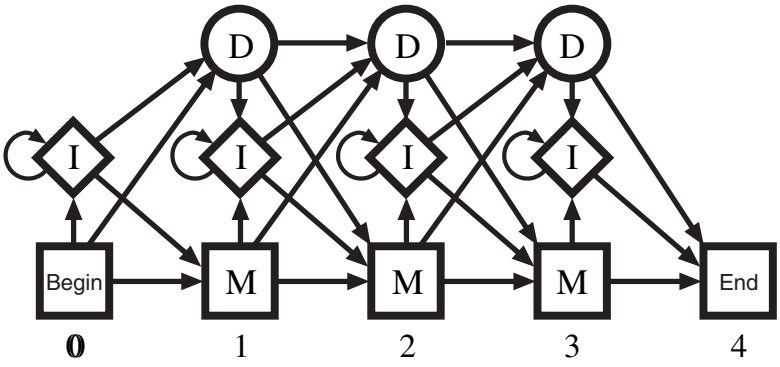


Figure 28.2: State transition diagram for a profile HMM. From Figure 5.7 of [Dur+98]. Used with kind permission of Richard Durbin.

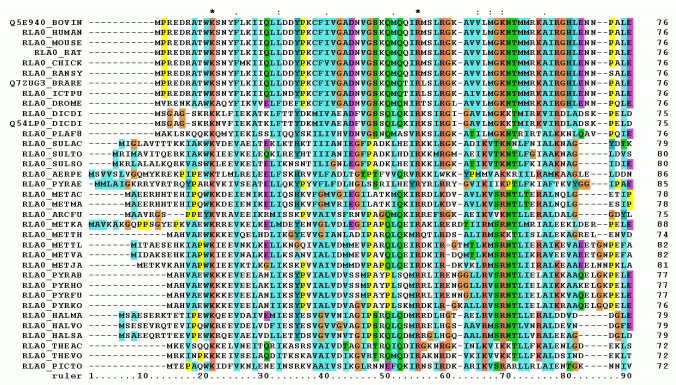


Figure 28.3: Example of multiple sequence alignment. We show the first 90 positions of the acidic ribosomal protein P0 from several organisms. Colors represent functional properties of the corresponding amino acid. Dashes represent insertions or deletions. From https://en.wikipedia.org/wiki/Multiple_sequence_alignment. Used with kind permission of Wikipedia author Miguel Andrade.

- Within-word errors, such as substitution, transposition, insertion and deletion of letters, as we discussed in Section 28.1.1.2.

We can model this with an HMM, where we augment the state space with a **silent state**, that does not emit any symbols. Figure 28.1 illustrates how this model can "denoise" the observed query "goverment home page of illnoisstate" into the correctly formulated query "government homepage of illnois state".

An alternative to using HMMs is to use supervised learning to fit a sequence-to-sequence translation model, using RNNs or transformers. This can work very well, but often needs much more training data, which can be problematic for **low-resource languages** [ECM18].

28.1.2 Protein sequence alignment

An important application of HMMs is to the problem of **protein sequence alignment** [Dur+98]. Here the goal is to determine if a test sequence $\boldsymbol{y}_{1:T}$ belongs to a protein family or not, and if so, how it aligns with the canonical representation of that family. (Similar methods can be used to align DNA and RNA sequences.) The technique we will use is similar to the spelling correction model in Section 28.1.1.2, but can be trained in an unsupervised way from raw sequences, without having to manually create labeled $(\boldsymbol{z}, \boldsymbol{y})$ pairs.

To solve the alignment problem, let us initially assume we have a set of aligned sequences from a protein family, from which we can generate a **consensus sequence**. This defines a probability distribution over

symbols at each location t in the string; denote each **position-specific scoring matrix** by $\theta_t(v) = p(y_t = v)$. These parameters can be estimated by counting.

Now we turn the PSSM into an HMM with 3 hidden states, representing the events that the location t matches the consensus sequence, $z_t = M$, or inserts its own unique symbol, $z_t = I$, or deletes (skips) the corresponding consensus symbol, $z_t = D$. We define the observation models for these 3 events as follows. For matches, we use the PSSM $p(y_t = v|z_t = M) = \theta_t(v)$. For insertions we use the uniform distribution $p(y_t = v|z_t = I) = 1/V$, where V is the size of the vocabulary. For deletions, we use $p(y_t = -|z_t = D)$, where "-" is a special deletion symbol used to pad the generated sequence to the correct length. The corresponding state transition matrix is shown in Figure 28.2: we see that matches and deletions advance one location along the consensus sequence, but insertions stay in the same location (represented by the self-transition from I to I). This model is known as a **profile HMM**.

Given a profile HMM with consensus parameters $\boldsymbol{\theta}$, we can compute $p(\boldsymbol{y}_{1:T}|\boldsymbol{\theta})$ in $O(T)$ time using the forwards algorithm, as described in **??**. This can be used to decide if the sequence belongs to this family or not, by thresholding the log-odds score, $L(\boldsymbol{y}) = \log p(\boldsymbol{y}|\boldsymbol{\theta})/p(\boldsymbol{y}|\mathcal{M}_0)$, where $\mathcal{M}_0$ is a baseline model, such as the uniform distribution. If the string matches, we can compute an alignment to the consensus using the Viterbi algorithm, as described in **??**. See Figure 28.3 for an illustration of such a **multiple sequence alignment**. If we don't have an initial set of aligned sequences from which to compute the consensus sequence $\boldsymbol{\theta}$, we can use the Baum-Welch algorithm (Section 28.2.1) to compute the MLE for the parameters $\boldsymbol{\theta}$ from a set of unaligned sequences. For details, see e.g., [Dur+98, Ch.6].

28.2 HMMs: parameter learning

In this section, we discuss how to compute a point estimate or the full posterior over the model parameters of an HMM given a set of partially observed sequences.

28.2.1 The Baum-Welch (EM) algorithm

In this section, we discuss how to compute an approximate MLE for the parameters of an HMM using the EM algorithm which is an iterative bound optimization algorithm (see **??** for details). When applied to HMMs, the resulting method is known as the **Baum-Welch** algorithm [Bau+70].

28.2.1.1 Log likelihood

The joint probability of a single sequence is given by

$$p(\boldsymbol{y}_{1:T}, \boldsymbol{z}_{1:T}|\boldsymbol{\theta}) = [p(z_1|\boldsymbol{\pi})] \left[\prod_{t=2}^{T} p(z_t|z_{t-1}, \mathbf{A})\right] \left[\prod_{t=1}^{T} p(\boldsymbol{y}_t|z_t, \mathbf{B})\right] \tag{28.2}$$

$$= \left[\prod_{k=1}^{K} \pi_k^{\mathbb{I}(z_1=k)}\right] \left[\prod_{t=2}^{T}\prod_{j=1}^{K}\prod_{k=1}^{K} A_{jk}^{\mathbb{I}(z_{t-1}=j, z_t=k)}\right] \left[\prod_{t=1}^{T}\prod_{k=1}^{K} p(\boldsymbol{y}_t|\boldsymbol{\theta}_k)^{\mathbb{I}(z_t=k)}\right] \tag{28.3}$$

where $\boldsymbol{\theta} = (\boldsymbol{\pi}, \mathbf{A}, \mathbf{B})$. Of course, we cannot compute this objective, since $\boldsymbol{z}_{1:T}$ is hidden. So instead we will optimize the expected complete data log likelihood, where expectations are taken using the parameters from the previous iteration of the algorithm:

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = \mathbb{E}_{p(\boldsymbol{z}_{1:T}|\boldsymbol{y}_{1:T}, \boldsymbol{\theta}^{\text{old}})} [\log p(\boldsymbol{y}_{1:T}, \boldsymbol{z}_{1:T}|\boldsymbol{\theta})] \tag{28.4}$$

This can be easily summed over N sequences. See Figure 28.4 for the graphical model.

The above objective is a lower bound on the observed data log likelihood, $\log p(\boldsymbol{y}_{1:T}|\boldsymbol{\theta})$, so the entire procedure is a bound optimization method that is guaranteed to converge to a local optimum. (In fact, in the case of HMMs, it can can be shown to converge to (close to) one of the global optima [YBW15].)

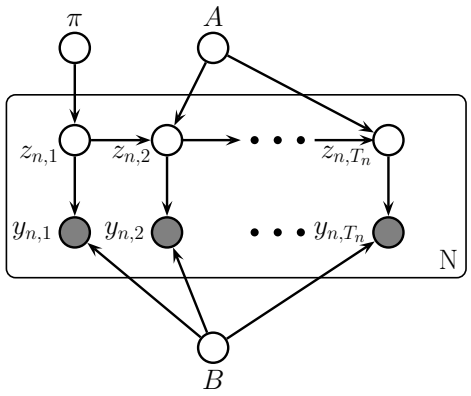


Figure 28.4: HMM with plate notation. A are the pararameters for the state transition matrix $p(z_t|z_{t-1})$ and B are the parameters for the discrete observation model $p(x_t|z_t)$. T_n is the length of the n'th sequence.

28.2.1.2 E step

Let $A_{jk} = p(z_t = k|z_{t-1} = j)$ be the $K \times K$ transition matrix. For the first time slice, let $\pi_k = p(z_1 = k)$ be the initial state distribution. Let $\boldsymbol{\theta}_k$ represent the parameters of the observation model for state k.

To compute the expected sufficient statistics, we first run the forwards-backwards algorithm on each sequence (see **??**). This returns the following node and edge marginals:

$$\gamma_{n,t}(j) \triangleq p(z_t = j|\boldsymbol{y}_{n,1:T_n}, \boldsymbol{\theta}^{old}) \tag{28.5}$$

$$\xi_{n,t}(j,k) \triangleq p(z_{t-1} = j, z_t = k|\boldsymbol{y}_{n,1:T_n}, \boldsymbol{\theta}^{\text{old}}) \tag{28.6}$$

We can then derive the expected counts as follows (note that we pool the sufficient statistics across time, since the parameters are tied, as well as across sequences):

$$\mathbb{E}\left[N_k^1\right] = \sum_{n=1}^{N} \gamma_{n,1}(k),\ \mathbb{E}\left[N_k\right] = \sum_{n=1}^{M}\sum_{t=2}^{T_n} \gamma_{n,t}(k),\ \mathbb{E}\left[N_{jk}\right] = \sum_{n=1}^{N}\sum_{t=2}^{T_n} \xi_{n,t}(j,k) \tag{28.7}$$

Given the above quantities, we can compute the expected complete data log likelihood as follows:

$$\begin{aligned} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = &\sum_{k=1}^{K} \mathbb{E}\left[N_k^1\right] \log \pi_k + \sum_{j=1}^{K}\sum_{k=1}^{K} \mathbb{E}\left[N_{jk}\right] \log A_{jk} \\ &+ \sum_{n=1}^{N}\sum_{t=1}^{T_n}\sum_{k=1}^{K} p(z_t = k|\boldsymbol{y}_n, \boldsymbol{\theta}^{\text{old}}) \log p(\boldsymbol{y}_{n,t}|\boldsymbol{\theta}_k) \end{aligned} \tag{28.8}$$

where T_n is the length of sequence n.

28.2.1.3 M step

We can estimate the transition matrix and initial state probabilities by maximizing the objective subject to the sum to one constraint. The result is just a normalized version of the expected counts:

$$\hat{A}_{jk} = \frac{\mathbb{E}\left[N_{jk}\right]}{\sum_{k'} \mathbb{E}\left[N_{jk'}\right]},\ \hat{\pi}_k = \frac{\mathbb{E}\left[N_k^1\right]}{N} \tag{28.9}$$

This result is quite intuitive: we simply add up the expected number of transitions from j to k, and divide by the expected number of times we transition from j to anything else.

For a categorical observation model, the expected sufficient statistics are

$$\mathbb{E}\left[M_{kv}\right] = \sum_{n=1}^{N}\sum_{t=1}^{T_n}\gamma_{n,t}(k)\mathbb{I}\left(y_{n,t}=v\right) = \sum_{n=1}^{N}\sum_{t:\boldsymbol{y}_{n,t}=v}\gamma_{n,t}(k) \tag{28.10}$$

The M step has the form

$$\hat{B}_{kv} = \frac{\mathbb{E}\left[M_{kv}\right]}{\mathbb{E}\left[N_k\right]} \tag{28.11}$$

This result is quite intuitive: we simply add up the expected number of times we are in state k and we see a symbol v, and divide by the expected number of times we are in state k. See Algorithm 11 for the pseudocode.

For a Gaussian observation model, the expected sufficient statistics are given by

$$\mathbb{E}\left[\overline{\boldsymbol{y}}_k\right] = \sum_{n=1}^{N}\sum_{t=1}^{T_n}\gamma_{n,t}(k)\boldsymbol{y}_{n,t},\ \mathbb{E}\left[(\overline{\boldsymbol{y}\boldsymbol{y}})_k^{\mathsf{T}}\right] = \sum_{n=1}^{N}\sum_{t=1}^{T_n}\gamma_{n,t}(k)\boldsymbol{y}_{n,t}\boldsymbol{y}_{n,t}^{\mathsf{T}} \tag{28.12}$$

The M step becomes

$$\hat{\boldsymbol{\mu}}_k = \frac{\mathbb{E}\left[\overline{\boldsymbol{y}}_k\right]}{\mathbb{E}\left[N_k\right]} \tag{28.13}$$

$$\hat{\boldsymbol{\Sigma}}_k = \frac{\mathbb{E}\left[(\overline{\boldsymbol{y}\boldsymbol{y}})_k^{\mathsf{T}}\right] - \mathbb{E}\left[N_k\right]\hat{\boldsymbol{\mu}}_k\hat{\boldsymbol{\mu}}_k^{\mathsf{T}}}{\mathbb{E}\left[N_k\right]} \tag{28.14}$$

In practice, we often need to add a log prior to these estimates to ensure the resulting $\hat{\boldsymbol{\Sigma}}_k$ estimate is well-conditioned. See [Mur22, Sec 4.5.2] for details.

Algorithm 5: Baum Welch algorithm for (discrete observation) HMMs

```
Initialize parameters θ ;
for each iteration do
    // E step ;
    Initialize expected counts: E[N_k] = 0, E[N_jk] = 0, E[M_kv] = 0;
    for each datacase n do
        Use forwards-backwards algorithm on y_n to compute γ_{n,t} and ξ_{n,t} (Equations 28.5–28.6) ;
        E[N_k] := E[N_k] + Σ_{t=2}^{T_n} γ_{n,t}(k) ;
        E[N_jk] := E[N_jk] + Σ_{t=2}^{T_n} ξ_{n,t}(j,k) ;
        E[M_kv] := E[M_kv] + Σ_{t:x_{n,t}=v} γ_{n,t}(k)
    // M step ;
    Compute new parameters θ = (A, B, π) using Equations 28.9
```

28.2.1.4 Initialization

As usual with EM, we must take care to ensure that we initialize the parameters carefully, to minimize the chance of getting stuck in poor local optima. There are several ways to do this, such as

- Use some fully labeled data to initialize the parameters.
- Initially ignore the Markov dependencies, and estimate the observation parameters using the standard mixture model estimation methods, such as K-means or EM.
- Randomly initialize the parameters, use multiple restarts, and pick the best solution.

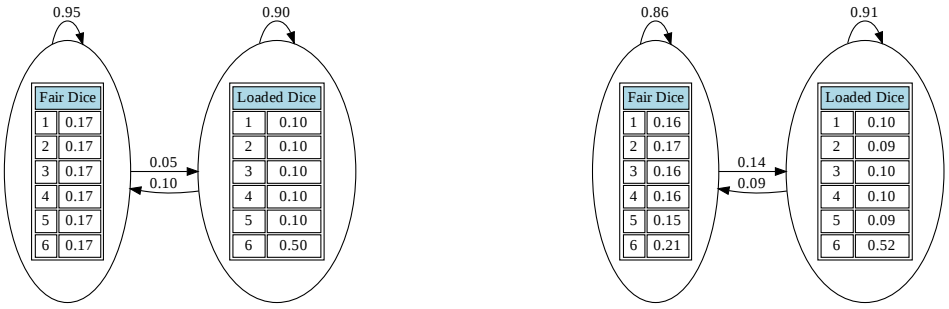


Figure 28.5: Illustration of the casino HMM. (a) True parameters used to generate the data. (b) Estimated paramters, using EM for 20 iterations. Generated by hmm_ casino_ em_ train.py.

Techniques such as deterministic annealing [UN98; RR01] can help mitigate the effect of local minima. Also, just as K-means is often used to initialize EM for GMMs, so it is common to initialize EM for HMMs using Viterbi training. The Viterbi algorithm is explained in **??**, but basically it is an algorithm to compute the single most probable path. As an approximation to the E step, we can replace the sum over paths with the statistics computed using this single path. Sometimes this can give better results [AG11].

28.2.1.5 Example: casino HMM

In this section, we fit the casino HMM from **??**. The true generative model is shown in Figure 28.5a. We used this to generate 5 sequences of varying length (max 3000), totalling 5670 observations. We initialized the model with random parameters, and then ran EM for 20 iterations. We got the results in Figure 28.5b.

28.2.1.6 Example: AR-HMM

In this section, we revisit the 2d AR-HMM example from **??**. We generate a single sequence of length 10,000, and fit the model using EM. In Figure 28.6(a) we show samples from the true model, and in Figure 28.6(b) we show samples from the learned model. We can see that the results are very similar, modulo the change in color due to label switching.

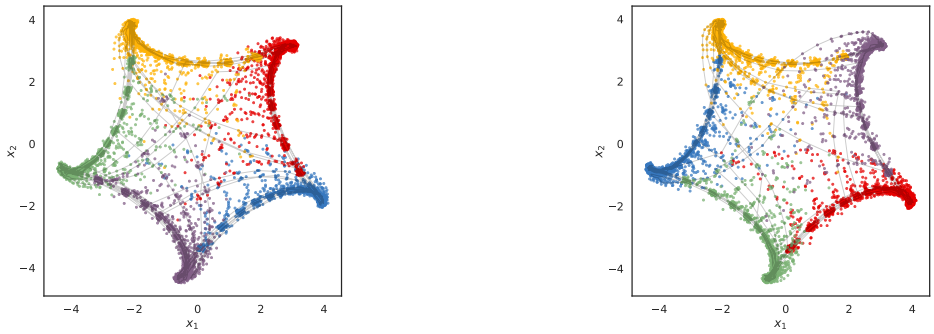
To avoid the label switching problem, we can find the optimal permutation between the learned states and the true states by solving a **linear assignment problem**, also called a **minimum weight matching** in a bipartite graph. This finds the binary matrix $\mathbf{X}$ which minimizes

$$\mathcal{L}(\mathbf{X}) = \sum_i \sum_j C_{ij} X_{ij} \tag{28.15}$$

where $X_{ij} = 1$ if row i is assigned to column j, and C_{ij} is the cost. This can be computed using the **Hungarian algorithm**. In the context of label switching, we first define N_{ij} as the number of time steps where the true label is state i and the predicted state is j, and then set $C_{ij} = -N_{ij}$ (since we want to maximize agreement).

28.2.2 Parameter estimation using SGD

Although the EM algorithm is the "traditional" way to fit HMMs, it is inherently a batch algorithm, so it does not scale well to large datasets (with many sequences). Although it is possible to extend bound optimization to the online case (see e.g., [Mai15]), this can take a lot of memory.



(a) *(b)*

Figure 28.6: (a) Samples from the true AR-HMM. (b) Samples from the learned AR-HMM.

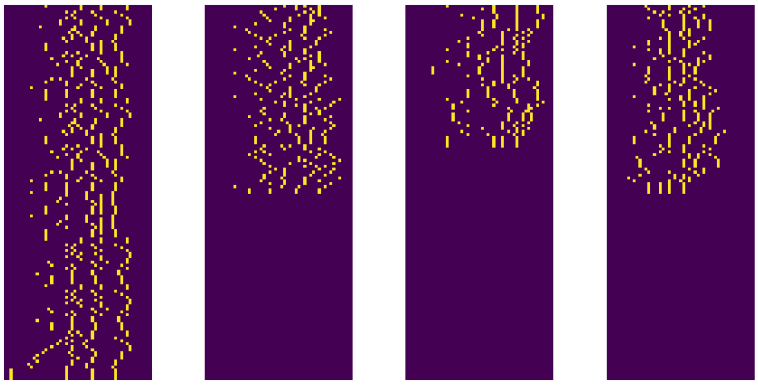


Figure 28.7: First 4 training sequences from the Bach Chorales dataset. Vertical axis represents time (start of music at top), horizontal axis represents the 51 notes. Generated by hmm_bach_chorales.ipynb.

A simple alternative is to optimize $\log p(\boldsymbol{y}_{1:T}|\boldsymbol{\theta})$ using SGD. We can compute this objective using the forwards algorithm, as shown in **??**:

$$\log p(\boldsymbol{y}_{1:T}|\boldsymbol{\theta}) = \sum_{t=1}^{T} \log p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) = \sum_{t=1}^{T} \log Z_t \tag{28.16}$$

where the normalization constant for each time step is given by

$$Z_t \triangleq p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}) = \sum_{j=1}^{K} p(z_t = j|\boldsymbol{y}_{1:t-1}) p(\boldsymbol{y}_t|z_t = j) \tag{28.17}$$

Of course, we need to ensure the transition matrix remains a valid row stochastic matrix, i.e., that $0 \leq A_{ij} \leq 1$ and $\sum_j A_{ij} = 1$. Similarly, if we have categorical observations, we need to ensure B_{jk} is a valid row stochastic matrix, and if we have Gaussian observations, we need to ensure $\boldsymbol{\Sigma}_k$ is a valid psd matrix. These constraints are automatically taken care of in EM. When using SGD, we can reparameterize to an unconstrained form, as proposed in [BC94].

28.2.2.1 Example: Bach Chorales

In this section, we give an example of fitting an HMM with SGD to some music data from [BLBV12]. The training set consists of 229 sequences of Chorales composed by J. S. Bach. Each sequence has a maximum length

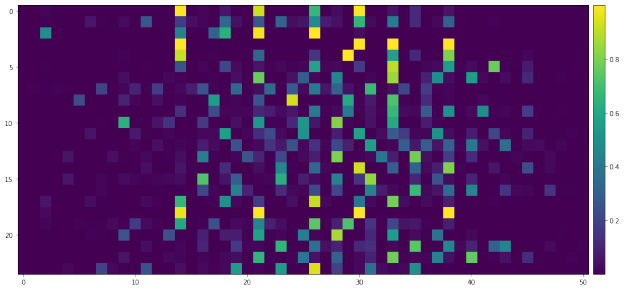


Figure 28.8: Learned observation distributions for the 51 notes from a 24 state HMM. Generated by hmm_bach_chorales.ipynb.

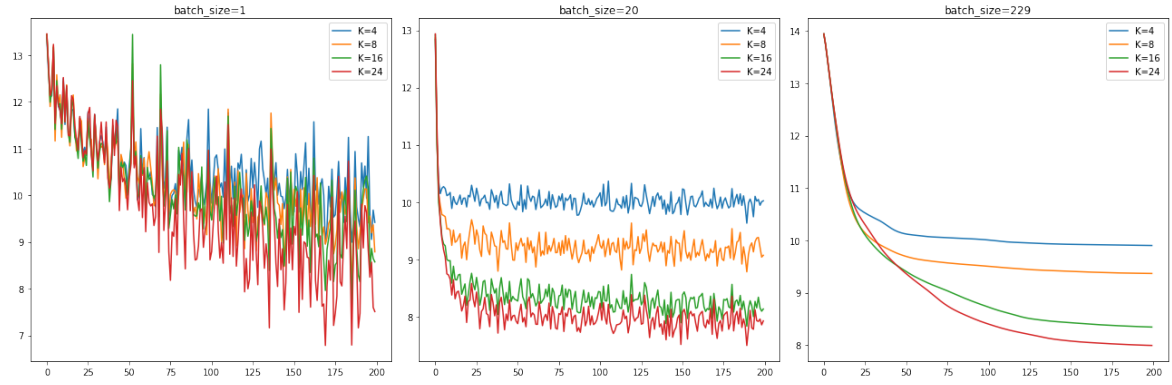


Figure 28.9: Negative log likelihood vs iterations on the Bach Chorales dataset for different number of hidden states K, and batch sizes of $B = 1$ (left), $B = 20$ (middle) and $B = N = 229$ (right). Generated by hmm_bach_chorales.ipynb.

of 129, and contains 51 notes.[3] Thus the data matrix is a boolean tensor of size $N \times T_{\max} \times D = 229 \times 129 \times 51$, although the actual sequence lengths are variable, so the data is stored in a ragged array. See Figure 28.7 for a visualization of some of the data.

We model the observation at each time step using a factored observation distribution of the form

$$p(\boldsymbol{y}_t|z_t = k) = \prod_{d=1}^{D} \text{Ber}(y_{td}|B_d(k)) \tag{28.18}$$

where $B_d(k) = p(y_{td} = 1|z_t = k)$. We then fit an HMM with $K \in \{4, 8, 16, 24\}$ states to this using SGD. In Figure 28.8 we show the learned observation model for $K = 24$ (which was the model with the best test set loglikelihood). We see that some states generate multiple notes at once, presumably corresponding to chords.

In Figure 28.9, we show how the negative log likelihood on the training set is reduced across SGD iterations. Obviously the training loss is lower for larger K. In addition, the loss curve is noisier for small batch sizes. The test set negative log likelihood using $K = 24$ is 8.05 nats per time step, which is comparable to the value of 8.28 reported in Table 1 of [Obe+19] using an HMM with $K = 16$.

28.2.3 Parameter estimation using spectral methods

Fitting HMMs using maximum likelihood is difficult, because the log likelihood is not convex. Thus there are many local optima, and EM and SGD can give poor results. An alternative approach is to marginalize out the hidden variables, and work instead with predictive distributions in the visible space. For discrete observation HMMs, with observation matrix $B_{jk} = p(y_t = k|z_t = j)$, such a distribution has the form

$$[\boldsymbol{y}_t]_k \triangleq p(y_t = k|\boldsymbol{y}_{1:t-1}) \tag{28.19}$$

This is called a **predictive state representation** [SJR04].

Suppose there are m possible hidden states, and n possible visible symbols, where $n \geq m$. One can show [HKZ12; Joh12] that the PSR vectors lie in a subspace in $\mathbb{R}^n$ with a dimensionality of $m \leq n$. Intuitively this is because the linear operator $\mathbf{A}$ defining the hidden state update in **??**, combined with the mapping to observables via $\mathbf{B}$, induces low rank structure in the output space. Furthermore, we can estimate a basis for this low rank subspace using SVD applied to the observable matrix of co-occurrence counts:

$$[\mathbf{P}_2]_{ij} = p(y_t = i, y_{t-1} = j) \tag{28.20}$$

We also need to estimate the third order statistics

$$[\mathbf{P}_3]_{ijk} = p(y_t = i, y_{t-1} = j, y_{t-2} = k) \tag{28.21}$$

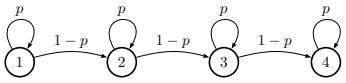
Using these quantities, it possible to perform recursive updating of our predictions while working entirely in visible space. This is called **spectral estimation**, or **tensor decomposition** [HKZ12; AHK12; Rod14; Ana+14; RSG17].

We can use spectral methods to get a good initial estimate of the parameters for the latent variable model, which can then be refined using EM (see e.g., [Smi+00]). Alternatvely, we can use them "as is", without needing EM at all. See [Mat14] for a comparison of these methods. See also Section 28.5.3 where we discuss spectral methods for fitting linear dynamical systems.

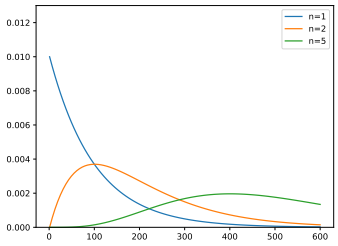
28.2.4 Bayesian parameter inference

MLE methods can easily overfit, and can suffer from numerical problems, especially when sample sizes are small. In [Fot+14], they show how to apply stochastic variational inference (**??**) to HMMs, by leveraging the

[3] We follow the same preprocessing as used in https://pyro.ai/examples/hmm.html, where they drop 37 notes that are never played.



(a)



(b)

Figure 28.10: (a) A Markov chain with $n = 4$ repeated states and self loops. (b) The resulting distribution over sequence lengths, for $p = 0.99$ and various n. Generated by hmm_self_loop_dist.py.

conjugate structure. An alternative approach is to marginalize out the discrete latents (using the forwards algorithm), and then to use SVI to target the log posterior

$$\log p(\boldsymbol{\theta}, \mathcal{D}) = \log p(\boldsymbol{\theta}) + \sum_{n=1}^{N} \log p(\boldsymbol{y}_{1:T,n}|\boldsymbol{\theta}) \tag{28.22}$$

This can be thought of as "collapsed SVI", and was proposed in [Obe+19].

We can also apply HMC (**??**) to the same log joint. This is a form of "collapsed MCMC". This is generally faster than the older technique of block Gibbs sampling [Sco02], that alternates between sampling latent sequences $\boldsymbol{z}^s_{1:T,1:N}$ using the forwards filtering backwards sampling algorithm (**??**) and sampling the parameters from their full conditionals, $p(\boldsymbol{\theta}|\boldsymbol{y}_{1:T}, \boldsymbol{z}^s_{1:T,1:N})$, since the high correlation between $\boldsymbol{z}$ and $\boldsymbol{\theta}$ makes this coordinate-wise approach rather slow.

28.3 HMMs: Generalizations

In this section, we discuss various extensions of the vanilla HMM introduced in **??**.

28.3.1 Hidden semi-Markov model (HSMM)

In a standard HMM (**??**), the probability we remain in state i for exactly d steps is

$$p(d_i = d) = (1 - A_{ii})A_{ii}^d \propto \exp(d \log A_{ii}) \tag{28.23}$$

where A_{ii} is the self-loop probability. This is called the **geometric distribution**. However, this kind of exponentially decaying function of d is sometimes unrealistic.

A simple way to model non-geometric waiting times is to replace each state with n new states, each with the same emission probabilities as the original state. For example, consider the model in Figure 28.10(a). Obviously the smallest sequence this can generate is of length $n = 4$. Any path of length d through the model has probability $p^{d-n}(1-p)^n$; multiplying by the number of possible paths we find that the total probability of a path of length d is

$$p(d) = \binom{d-1}{n-1} p^{d-n}(1-p)^n \tag{28.24}$$

This is equivalent to the negative binomial distribution. By adjusting n and the self-loop probabilities p of each state, we can model a wide range of waiting times: see Figure 28.10(b).

A more general solution is to use a **semi-Markov model**, in which the next state not only depends on the previous state, but also on how how long we've been in that state. When the state space is not observed

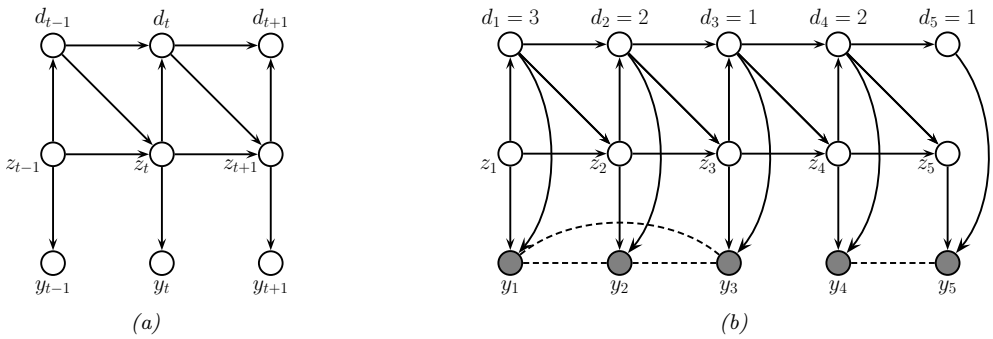


Figure 28.11: Encoding a hidden semi-Markov model as a PGM-D. (a) d_t is a deterministic down counter (duration variable). Each observation is generated independently. (b) Similar to (a), except now we generate the observations within each segment as a block. In this figure, we represent the non-Markovian dependencies between the observations within each segment by using undirected edges. We represent the conditional independence between the observations across different segments by disconnecting $\boldsymbol{y}_{1:3}$ from $\boldsymbol{y}_{4:5}$; this can be enforced by "breaking the link" whenever $d_t = 1$ (representing the end of a segment).

directly, the result is called a **hidden semi-Markov model** (**HSMM**), a **variable duration HMM**, or an **explicit duration HMM** [Yu10].

One way to represent a HSMM is to use the graphical model shown in Figure 28.11(a). The $d_t \in \{1, \ldots, D\}$ node is a state duration counter, where D is the maximum duration of any state. When we first enter state j, we sample d_t from the duration distribution for that state, $d_t \sim p_j(\cdot)$. Thereafter, d_t deterministically counts down until $d_t = 1$. More precisely, we define the following CPD:

$$p(d_t = d' | d_{t-1} = d, z_t = j) = \begin{cases} D_j(d') & \text{if } d = 1 \\ 1 & \text{if } d' = d - 1 \text{ and } d > 0 \\ 0 & \text{otherwise} \end{cases} \tag{28.25}$$

Note that $D_j(d)$ could be represented as a table (a non-parametric approach) or as some kind of parametric distribution, such as a Gamma distribution. If $D_j(d)$ is a geometric distribution, this emulates a standard HMM.

While $d_t > 1$, the state z_t is not allowed to change. When $d_t = 1$, we make a stochastic transition to a new state. More precisely, we define the state CPD as follows:

$$p(z_t = k | z_{t-1} = j, d_{t-1} = d) = \begin{cases} 1 & \text{if } d > 0 \text{ and } j = k \\ A_{jk} & \text{if } d = 1 \\ 0 & \text{otherwise} \end{cases} \tag{28.26}$$

This ensures that the model stays in the same state for the entire duration of the segment. At each step within this segment, an observation is generated.

HSMMs are useful not only because they can model the duration of each state explicitly, but also because they can model the distribution of a whole subsequence of observations at once, instead of assuming all observations are generated independently at each time step. That is, they can use likelihood models of the form $p(\boldsymbol{y}_{t:t+l} | z_t = k, d_t = l)$, which generate l correlated observations if the duration in state k is for l time steps. This approach, known as a **segmental HMM**, is useful for modeling data that is piecewise linear, or shows other local trends [ODK96]. We can also use an RNN to model each segment, resulting in an **RNN-HSMM** model [Dai+17].

More precisely, we can define a segmental HMM as follows:

$$p(\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{d}) = \left[p(z_1) p(d_1 | z_1) \prod_{t=2}^{T} p(z_t | z_{t-1}, d_{t-1}) p(d_t | z_t, d_{t-1}) \right] p(\boldsymbol{y} | \boldsymbol{z}, \boldsymbol{d}) \tag{28.27}$$

In a standard HSMM, we assume

$$p(\boldsymbol{y}|\boldsymbol{z},\boldsymbol{d}) = \prod_{t=1}^{T} p(y_t|z_t) \tag{28.28}$$

so the duration variables only determine the hidden state dynamics. To define $p(\boldsymbol{y}|\boldsymbol{z},\boldsymbol{d})$ for a segmental HMM, let us use s_i and e_i to denote the start and end times of segment i. This sequence can be computed deterministically from $\boldsymbol{d}$ using $s_1 = 1$, $s_i = s_{i-1} + d_{s_{i-1}}$, and $e_i = s_i + d_{s_i} - 1$. We now define the observation model as follows:

$$p(\boldsymbol{y}|\boldsymbol{z},\boldsymbol{d}) = \prod_{i=1}^{|\boldsymbol{s}|} p(\boldsymbol{y}_{s_i:e_i}|z_{s_i}, d_{s_i}) \tag{28.29}$$

See Figure 28.11(b) for the PGM-D.

If we use an RNN for each segment, we have

$$p(\boldsymbol{y}_{s_i:e_i}|z_{s_i}, d_{s_i}) = \prod_{t=s_i}^{e_i} p(y_t|\boldsymbol{y}_{s_i:t-1}, z_{s_i}) = \prod_{t=s_i}^{e_i} p(y_t|h_t, z_{s_i}) \tag{28.30}$$

where h_t is the hidden state that is deterministically updated given the previous observations in this sequence.

As shown in [Chi14], it is possible to compute $p(z_t, d_t|\boldsymbol{y}_{1:T})$ in $O(TK^2 + TKD)$ time, where T is the sequence length, K is the number of states, and D is the maximum duration of any segment. In [Dai+17], they show how to train an approximate inference algorithm, based on a mean field approximation $q(\boldsymbol{z},\boldsymbol{d}|\boldsymbol{y}) = \prod_t q(z_t|\boldsymbol{y})q(d_t|\boldsymbol{y})$, to compute the posterior in $O(TK + TD)$ time.

28.3.2 HSMMs for changepoint detection

In this section, we discuss how to use HSMM-like models for the problem of **changepoint detection**., which is the task of detecting when there are "abrupt" changes in the distribution of the observed values in a time series. For a review of offline methods to this problem, see e.g., [AC17; TOV18]. (See also [BW20] for a recent empirical evaluation of various methods, focused on the 1d time series case.)

In this section, we focus on the online case. The methods we discuss can (in principle) be used for high-dimensional time series segmentation. Our starting point is the hidden semi-Markov models (HSMM) discussed in Section 28.3.1. This is like an HMM in which we explicitly model the duration spent in each state. This is done by augmenting the latent state z_t with a duration variable d_t which is initialized according to a duration distribution, $d_t \sim D_{z_t}(\cdot)$, and which then *counts down* to 0. An alternative approach is to add a variable $r_t\{0, 1, \dots, \}$ which encodes the **run length** for the current state; this starts at 0 whenever a new segment is created, and then *counts up* by one at each step. The transition dynamics is specified by

$$p(r_t|r_{t-1}) = \begin{cases} H(r_{t-1}+1) & \text{if } r_t = 0 \\ 1 - H(r_{t-1}+1) & \text{if } r_t = r_{t-1}+1 \\ 0 & \text{otherwise} \end{cases} \tag{28.31}$$

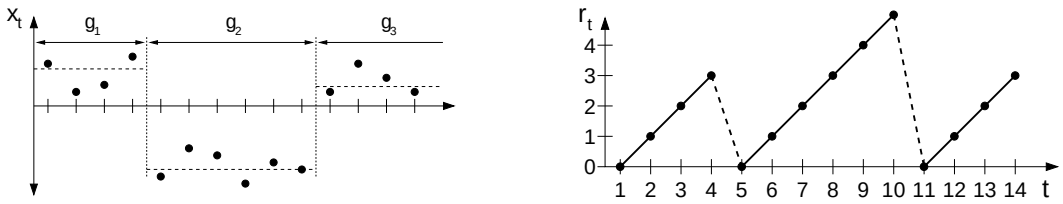
where $H(\tau)$ is a **hazard function**:

$$H(\tau) = \frac{p_g(\tau)}{\sum_{t=\tau}^{\infty} p_g(t)} \tag{28.32}$$

where $p_g(t)$ is the probability of a gap of length t. See Figure 28.12 for an illustration. If we set p_g to be a geometric distribution with parameter λ, then the hazard function is the constant $H(\tau) = 1/\lambda$.

The advantage of the run-length representation is that we can define the observation model for a segment in a causal way (that only depends on past data):

$$p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}, r_t = r, z_t = k) = p(\boldsymbol{y}_t|\boldsymbol{y}_{t-r:t-1}, z_t = k) = \int p(\boldsymbol{y}_t|\boldsymbol{\eta})p(\boldsymbol{\eta}|\boldsymbol{y}_{t-r:t-1}, z_t = k)d\boldsymbol{\eta} \tag{28.33}$$



(a) *(b)*

Figure 28.12: Illustration of Bayesian online changepoint detection (BOCPD). (a) Hypothetical segmentation of a univariate time series divided by changepoints on the mean into three segments of lengths $g_1 = 4$, $g_2 = 6$, and an undetermined length for g_3 (since it the third segment has not yet ended). From Figure 1 of [AM07]. Used with kind permission of Ryan Adams.

where $\boldsymbol{\eta}$ are the parameters that are "local" to this segment. This called the **underlying predictive model** or **UPM** for the segment. The posterior over the UPM parameters is given by

$$p(\boldsymbol{\eta}|\boldsymbol{y}_{t-r:t-1}, z_t = k) \propto p(\boldsymbol{\eta}|z_t = k) \prod_{i=t-r}^{t-1} p(\boldsymbol{y}_i|\boldsymbol{\eta}) \tag{28.34}$$

where we initialize the prior for $\boldsymbol{\eta}$ using hyper-parameters chosen by state k. If the model is conjugate exponential, we can compute this marginal likelihood in closed form, and we have

$$\pi_t^{r,k} = p(\boldsymbol{y}_t|\boldsymbol{y}_{t-r:t-1}, z_t = k) = p(\boldsymbol{y}_t|\boldsymbol{\psi}_t^{r,k}) \tag{28.35}$$

where $\boldsymbol{\psi}_t^{r,k}$ are the parameters of the posterior predictive distribution at time t based on the last r observations (and using a prior from state k).

In the special case in which we have $K = 1$ hidden states, then each segment is modeled independently, and we get a **product partition model** [BH92]:

$$p(\boldsymbol{y}|\boldsymbol{r}) = p(\boldsymbol{y}_{s_1:e_1}) \dots p(\boldsymbol{y}_{s_N:e_N}) \tag{28.36}$$

where s_i and e_i are the start and end of segment i, which can be computed from the run lengths $\boldsymbol{r}$. (We initialize with $r_0 = 0$.) Thus there is no information sharing between segments. This can be useful for timeseries in which there are abrubt changes, and where the new parameters are unrelated to the old ones.

Detecting the locations of these changes is called **changepoint detection**. An exact online algorithm for solving this task was proposed in [FL07] and independently in [AM07]; in the latter paper, they call the method **Bayesian online changepoint detection** or **BOCPD**. We can compute a posterior over the current run length recursively as follows:

$$p(r_t|\boldsymbol{y}_{1:t}) \propto p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-}, r_t) p(r_t|\boldsymbol{y}_{1:t-1}) \tag{28.37}$$

where we initialize with $p(r_0 = 0) = 1$. The marginal likelihood $p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-}, r_t)$ is given by Equation (28.33) (with $z_t = 1$ dropped, since there is just one state). The prior predictive is given by

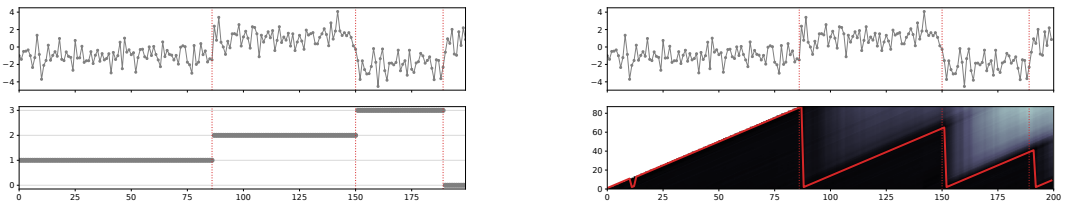
$$p(r_t|\boldsymbol{y}_{1:t-1}) = \sum_{r_{t-1}} p(r_t|r_{t-1}) p(r_{t-1}|\boldsymbol{y}_{1:t-1}) \tag{28.38}$$

The one step ahead predictive distribution is given by

$$p(\boldsymbol{y}_{t+1}|\boldsymbol{y}_{1:t}) = \sum_{r_t} p(\boldsymbol{y}_{t+1}|\boldsymbol{y}_{1:t}, r_t) p(r_t|\boldsymbol{y}_{1:t}) \tag{28.39}$$

28.3.2.1 Example

We give an example of the method in Figure 28.13 applied to a synthetic 1d dataset generated from a 4 state GMM. The likelihood is a univariate Gaussian, $p(y_t|\mu) = \mathcal{N}(y_t|\mu, \sigma^2)$, where $\sigma^2 = 1$ is fixed, and μ is inferred using a Gaussian prior. The hazard function is set to a geometric distribution with rate N/T, where $N = 4$ is the true number of change points and $T = 200$ is the length of the sequence.



(a) *(b)*

Figure 28.13: Illustration of BOCPD. (a) Synthetic data from a GMM with 4 states. Top row is the data, bottom row is the generating state. (b) Output of algorithm. Top row: Estimated changepoint locations. Bottom row: posterior predicted probability of a changepoint at each step. Generated by changepoint_ detection.py.

28.3.2.2 Extensions

Although the above method is exact, each update step takes $O(t)$ time, so the total cost of the algorithm is $O(T^2)$. We can reduce this by pruning out states with low probability. In particular, we can use particle filtering (**??**) with N particles, together with a stratified optimal resampling method, to reduce the cost to $O(TN)$. See [FL07] for details.

In addition, the above method relies on a conjugate exponential model in order to compute the marginal likelihood, and udpate the posterior parameters for each r, in $O(1)$ time. For more complex models, we need to use approximations. In [TBS13], they use variational Bayes (**??**), and in [Mav16], they use particle filtering (**??**), which is more general, but much slower.

It is possible to extend the model in various other ways. In [FL11], they allow for Markov dependence between the parameters of neighboring segments. In [STR10], they use a Gaussian process (**??**) to represent the UPM, which captures correlations between observations within the same segment. In [KJD18], they use generalized Bayesian inference (**??**) to create a method that is more robust to model misspecification.

In [Gol+17], they extend the model by modeling the probability of a sequence of observations, rather than having to make the decision about whether to insert a changepont or not based on just on the likelihood ratio of a single time step.

In [AE+20], they extend the model by allowing for multiple discrete states, as in an HSMM. In addition, they add both the run length r_t and the duration d_t to the state space. This allows the method to specify not just when the current segment started, but also when it is expected to end. In addition, it allows the UPM to depend on the duration of the segment, and not just on past observations. For example, we can use

$$p(y_t|r_t, d_t, \eta) = \mathcal{N}(y_t|\boldsymbol{\phi}(r_t/d_t)^\mathsf{T}\boldsymbol{\eta}, \sigma^2) \tag{28.40}$$

where $0 \leq r_t/d_t \leq 1$, and $\boldsymbol{\phi}()$ is a set of learned basis functions. This allows observation sequences for the same hidden state to have a common functional shape, even if the time spent in each state is different.

28.3.3 Hierarchical HMMs

A **hierarchical HMM** (HHMM) [FST98] is an extension of the HMM that is designed to model domains with hierarchical structure. Figure 28.14 gives an example of an HHMM used in automatic speech recognition, where words are composed of phones which are composed of subphones. We can always "flatten" an HHMM to a regular HMM, but a factored representation is often easier to interpret, and allows for more efficient inference and model fitting.

HHMMs have been used in many application domains, e.g., speech recognition [Bil01], gene finding [Hu+00], plan recognition [BVW02], monitoring transportation patterns [Lia+07], indoor robot localization [TMK04], etc. HHMMs are less expressive than stochastic context free grammars (SCFGs) since they only allow hierarchies of bounded depth, but they support more efficient inference. In particular, inference in

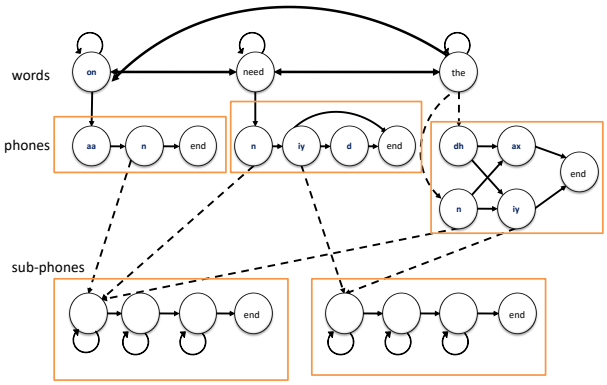


Figure 28.14: An example of an HHMM for an ASR system which can recognize 3 words. The top level represents bigram word probabilities. The middle level represents the phonetic spelling of each word. The bottom level represents the subphones of each phone. (It is traditional to represent a phone as a 3 state HMM, representing the beginning, middle and end; these are known as subphones.) Adapted from Figure 7.5 of [JM00].

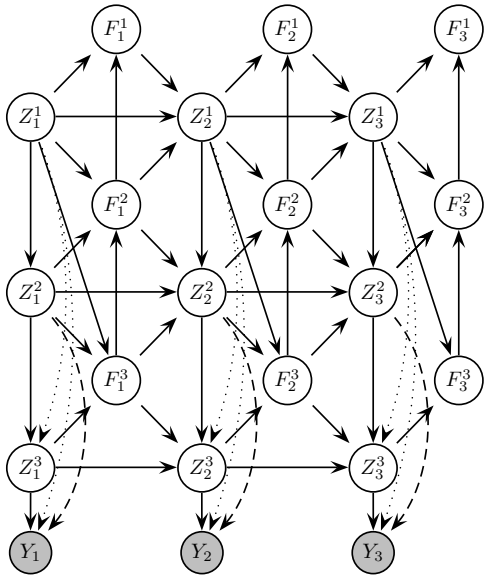


Figure 28.15: An HHMM represented as a PGM-D. Z_t^ℓ is the state at time t, level ℓ; $F_t^\ell = 1$ if the HMM at level ℓ has finished (entered its exit state), otherwise $F_t^\ell = 0$. Shaded nodes are observed; the remaining nodes are hidden. We may optionally clamp $F_T^\ell = 1$, where T is the length of the observation sequence, to ensure all models have finished by the end of the sequence. From Figure 2 of [MP01].

SCFGs (using the inside outside algorithm, [JM08]) takes $O(T^3)$ whereas inference in an HHMM takes $O(T)$ time [MP01; WM12].

We can represent an HHMM as a directed graphical model as shown in Figure 28.15. Q_t^ℓ represents the state at time t and level ℓ. A state transition at level ℓ is only "allowed" if the chain at the level below has "finished", as determined by the $F_t^{\ell-1}$ node. (The chain below finishes when it chooses to enter its end state.) This mechanism ensures that higher level chains evolve more slowly than lower level chains, i.e., lower levels are nested within higher levels.

A variable duration HMM can be thought of as a special case of an HHMM, where the top level is a deterministic counter, and the bottom level is a regular HMM, which can only change states once the counter has "timed out". See [MP01] for further details.

28.3.4 Factorial HMMs

An HMM represents the hidden state using a single discrete random variable $z_t \in \{1, \ldots, K\}$. To represent 10 bits of information would require $K = 2^{10} = 1024$ states. By contrast, consider a **distributed representation** of the hidden state, where each $z_{t,m} \in \{0, 1\}$ represents the m'th bit of the t'th hidden state. Now we can represent 10 bits using just 10 binary variables. This model is called a **factorial HMM** [GJ97].

More precisely, the model is defined as follows:

$$p(\boldsymbol{z}, \boldsymbol{y}) = \prod_t \left[\prod_m p(z_{tm}|z_{t-1,m}) \right] p(\boldsymbol{y}_t|\boldsymbol{z}_t) \tag{28.41}$$

where $p(z_{tm} = k|z_{t-1,m} = j) = A_{mjk}$ is an entry in the transition matrix for chain m, $p(z_{1m} = k|z_{0m}) = p(z_{1m} = k) = \pi_{mk}$, is the initial state distribution for chain m, and

$$p(\boldsymbol{y}_t|\boldsymbol{z}_t) = \mathcal{N}\left(\boldsymbol{y}_t| \sum_{m=1}^{M} \mathbf{W}_m \boldsymbol{z}_{tm}, \boldsymbol{\Sigma}\right) \tag{28.42}$$

is the observation model, where $\boldsymbol{z}_{tm}$ is a 1-of-K encoding of z_{tm} and $\mathbf{W}_m$ is a $D \times K$ matrix (assuming $\boldsymbol{y}_t \in \mathbb{R}^D$). Figure 28.16(a) illustrates the model for the case where $M = 3$.

An interesting application of FHMMs is to the problem of **energy disaggregation** [KJ12]. In this problem, we observe the total energy usage of a house at each moment in time, i.e., the observation model has the form

$$p(y_t|\boldsymbol{z}_t) = \mathcal{N}(y_t| \sum_{m=1}^{M} w_m z_{tm}, \sigma^2) \tag{28.43}$$

where w_m is the amount of energy used by device m, and $z_{tm} = 1$ if device m is being used at time t and $z_{tm} = 0$ otherwise. The transition model is assumed to be

$$p(z_{t,m} = 1|z_{t-1,m}) = \begin{cases} A_{01} & \text{if } z_{t-1,m} = 0 \\ A_{11} & \text{if } z_{t-1,m} = 1 \end{cases} \tag{28.44}$$

We do not know which devices are turned on at each time step (i.e., the z_{tm} are hidden), but by applying inference in the FHMM over time, we can separate the total energy into its parts, and thereby determine which devices are using the most electricity.

28.3.4.1 Structured mean field for FHMMs

Even though each chain in an FHMM is a priori independent, they become coupled in the posterior due to having an observed common child, $\boldsymbol{y}_t$. Exact inference for this model therefore takes $O(TMK^{M+1})$ time. In this section, we derive a structured mean field algorithm, from [GJ97], that takes $O(TMK^2I)$ time, where I is the number of mean field iterations (typically $I \sim 10$ suffices for good performance).

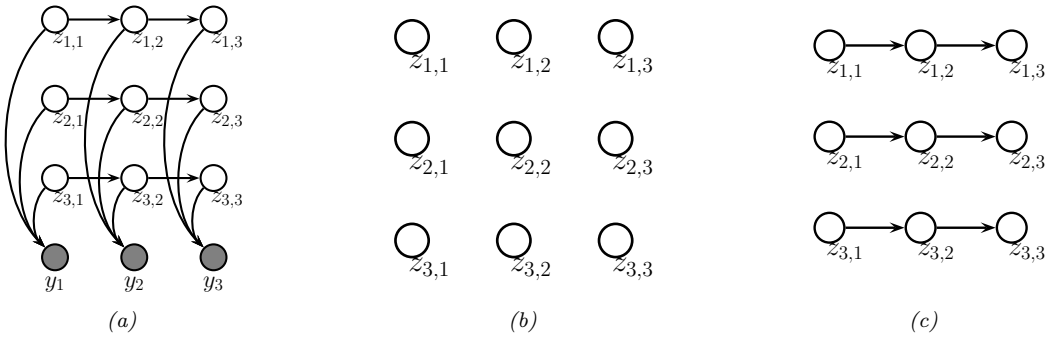


Figure 28.16: (a) A factorial HMM with 3 chains. (b) A fully factorized approximation. (c) A product-of-chains approximation. Adapted from Figure 2 of [GJ97].

We can write the exact posterior in the following form:

$$p(\boldsymbol{z}|\boldsymbol{y}) = \frac{1}{Z(\boldsymbol{y})} \exp(-\mathcal{E}(\boldsymbol{z}, \boldsymbol{y})) \tag{28.45}$$

$$\begin{aligned}\mathcal{E}(\boldsymbol{z}, \boldsymbol{y}) &= \frac{1}{2} \sum_{t=1}^{T} \left(\boldsymbol{y}_t - \sum_m \mathbf{W}_m \boldsymbol{z}_{tm} \right)^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} \left(\boldsymbol{y}_t - \sum_m \mathbf{W}_m \boldsymbol{z}_{tm} \right) \\ &\quad - \sum_{m=1}^{M} \boldsymbol{z}_{1m}^{\mathsf{T}} \tilde{\boldsymbol{\pi}}_m - \sum_{t=2}^{T} \sum_{m=1}^{M} \boldsymbol{z}_{t-1,m}^{\mathsf{T}} \tilde{\mathbf{A}}_m \boldsymbol{z}_{t,m}\end{aligned} \tag{28.46}$$

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.

We can approximate the posterior as a product of marginals, as in Figure 28.16(b), but a better approximation is to use a product of chains, as in Figure 28.16(c). Each chain can be tractably updated individually, using the forwards-backwards algorithm (**??**). More precisely, we assume

$$q(\boldsymbol{z}|\boldsymbol{y}) = \frac{1}{Z_q} \prod_{m=1}^{M} q(z_{1m}|\boldsymbol{\psi}_{1m}) \prod_{t=2}^{T} q(z_{tm}|z_{t-1,m}, \boldsymbol{\psi}_{tm}) \tag{28.47}$$

$$q(z_{1m}|\boldsymbol{\psi}_{1m}) = \prod_{k=1}^{K} (\xi_{1mk} \pi_{mk})^{z_{1mk}} \tag{28.48}$$

$$q(z_{tm}|z_{t-1,m}, \boldsymbol{\psi}_{tm}) = \prod_{k=1}^{K} \left(\xi_{tmk} \prod_{j=1}^{K} (A_{mjk})^{z_{t-1,m,j}} \right)^{z_{tmk}} \tag{28.49}$$

Here the variational parameter ξ_{tmk} plays the role of an approximate local evidence, averaging out the effects of the other chains. This is in contrast to the exact local evidence, which couples all the chains together.

By separating out the approximate local evidence terms, we can rewrite the above as $q(\boldsymbol{z}|\boldsymbol{y}) = \frac{1}{Z_q(\boldsymbol{y})} \exp(-\mathcal{E}_q(\boldsymbol{z}, \boldsymbol{y}))$, where

$$\mathcal{E}_q(\boldsymbol{z}, \boldsymbol{y}) = -\sum_{t=1}^{T} \sum_{m=1}^{M} \boldsymbol{z}_{tm}^{\mathsf{T}} \tilde{\boldsymbol{\psi}}_{tm} - \sum_{m=1}^{M} \boldsymbol{z}_{1m}^{\mathsf{T}} \tilde{\boldsymbol{\pi}}_m - \sum_{t=2}^{T} \sum_{m=1}^{M} \boldsymbol{z}_{t-1,m}^{\mathsf{T}} \tilde{\mathbf{A}}_m \boldsymbol{z}_{t,m} \tag{28.50}$$

where $\tilde{\boldsymbol{\psi}}_{tm} = \log \boldsymbol{\psi}_{tm}$. We see that this has the same temporal factors as the exact log joint in Equation (28.46), but the local evidence terms are different: the dependence on the visible data $\boldsymbol{y}$ has been replaced by dependence on "virtual data" $\boldsymbol{\psi}$.

The objective function is given by

$$D_{\mathbb{KL}}(q \parallel \tilde{p}) = \mathbb{E}_q[\log q - \log \tilde{p}] \tag{28.51}$$

$$= -\mathbb{E}_q[\mathcal{E}_q(\boldsymbol{z}, \boldsymbol{y})] - \log Z_q(\boldsymbol{y}) + \mathbb{E}_q[\mathcal{E}(\boldsymbol{z}, \boldsymbol{y})] + \log Z(\boldsymbol{y}) \tag{28.52}$$

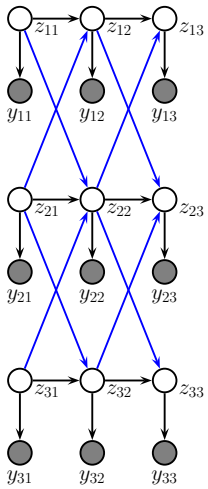


Figure 28.17: A coupled HMM wth 3 chains.

where $q = q(\boldsymbol{z}|\boldsymbol{y})$ and $\tilde{p} = p(\boldsymbol{z}|\boldsymbol{y})$. One can show that we can optimize this using coordinate descent, where each update step is given by

$$\boldsymbol{\psi}_{tm} = \exp\left(\mathbf{W}_m^\mathsf{T}\boldsymbol{\Sigma}^{-1}\tilde{\boldsymbol{y}}_{tm} - \frac{1}{2}\boldsymbol{\delta}_m\right) \tag{28.53}$$

$$\boldsymbol{\delta}_m \triangleq \text{diag}(\mathbf{W}_m^\mathsf{T}\boldsymbol{\Sigma}^{-1}\mathbf{W}_m) \tag{28.54}$$

$$\tilde{\boldsymbol{y}}_{tm} \triangleq \boldsymbol{y}_t - \sum_{\ell \neq m}^{M} \mathbf{W}_\ell \mathbb{E}\left[\boldsymbol{z}_{t,\ell}\right] \tag{28.55}$$

The intuitive interpretation of $\tilde{\boldsymbol{y}}_{tm}$ is that it is the observation $\boldsymbol{y}_t$ minus the predicted effect from all the other chains apart from m. This is then used to compute the approximate local evidence, $\boldsymbol{y}_{tm}$. Having computed the $\boldsymbol{\psi}_{tm}$ terms for each chain, we can perform forwards-backwards in parallel, using these approximate local evidence terms to compute $q(\boldsymbol{z}_{t,m}|\boldsymbol{y}_{1:T})$ for each m and t.

The update cost is $O(TMK^2)$ for a full "sweep" over all the variational parameters, since we have to run forwards-backwards M times, for each chain independently. This is the same cost as a fully factorized approximation, but is much more accurate.

28.3.5 Coupled HMMs

If we have multiple related data streams, we can use a **coupled HMM** [Bra96]. This is a series of HMMs where the state transitions depend on the states of neighboring chains. That is, we represent the conditional distribution for each time slice as

$$p(\boldsymbol{z}_t, \boldsymbol{y}_t|\boldsymbol{z}_{t-1}) = \prod_m p(\boldsymbol{y}_{tm}|z_{tm})p(z_{tm}|\boldsymbol{z}_{t-1,m-1:m+1}) \tag{28.56}$$

with boundary conditions defined in the obvious way. See Figure 28.17 for an illustration with $M = 3$ chains.

Coupled HMMs have been used for various tasks, such as **audio-visual speech recognition** [Nef+02], modeling freeway traffic flows [KM00], and modeling conversational interactions between people [Bas+01].

However, there are two drawbacks to this model. First, exact inference takes $O(T(K^M)^2)$, as in an factorial HMM; however, in practice this is not usually a problem, since M is often small. Second, the model requires $O(MK^4)$ parameters to specify, if there are M chains with K states per chain, because each state depends on its own past plus the past of its two neighbors. There is a closely related model, known as the **influence model** [Asa00], which uses fewer parameters, by computing a convex combination of pairwise transition matrices.

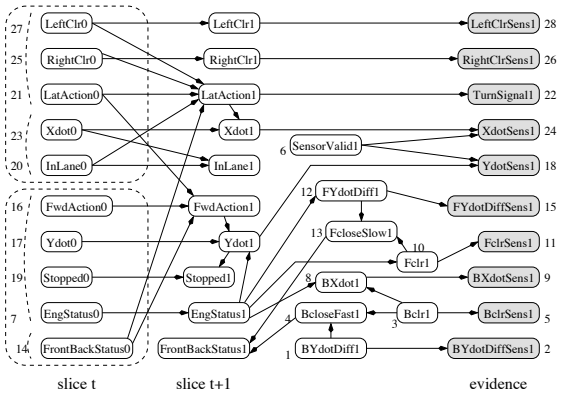


Figure 28.18: The BATnet DBN. The transient nodes are only shown for the second slice, to minimize clutter. The dotted lines are used to group related variables. Used with kind permission of Daphne Koller.

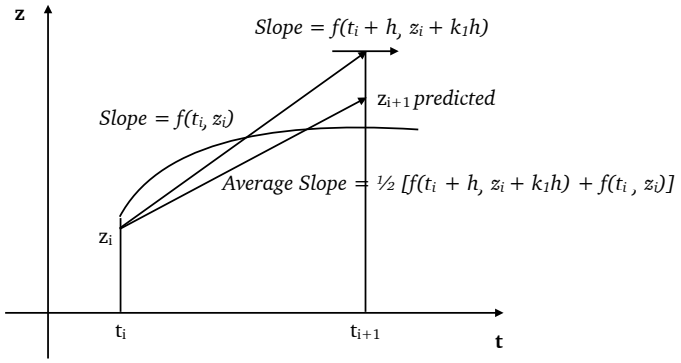


Figure 28.19: Illustration of one step of second-order Runge-Kutta method with step size h.

28.3.6 Dynamic Bayes nets (DBN)

A **dynamic Bayesian network** (**DBN**) is a way to represent a stochastic process using a directed graphical model [Mur02]. (Note that the network is not dynamic (the structure and parameters are fixed), rather it is a network representation of a dynamical system.) A DBN can be considered as a natural generalization of an HMM.

An example is shown in Figure 28.18, which is a DBN designed to monitor the state of a simulated autonomous car known as the "Bayesian Automated Taxi", or "BATmobile" [For+95]. To define the model, you just need to specify the structure of the first time-slice, the structure between two time-slices, and the form of the CPDs. For details, see [KF09].

28.4 Continuous time SSMs

In this section, we briefly discuss continuous time dynamical systems.

28.4.1 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**:

$$\frac{dz}{dt} = f(t, z) \tag{28.57}$$

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

$$z_{i+1} = z_i + \int_{t_{i-1}}^{t_i} f(t, z_i) dt \tag{28.58}$$

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:

$$k_1 = f(t_i, z_i) \tag{28.59}$$
$$k_2 = f(t_i + \delta, z_i + k_1 \delta) \tag{28.60}$$
$$z_{i+1} = z_i + \frac{\delta}{2}(k_1 + k_2) \tag{28.61}$$

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 28.19.

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.

28.4.2 Example: Noiseless 1d spring-mass system

In this section, we consider an example from Wikipedia[4] of a **spring mass system** operating in 1d. Like many physical systems, this is best modeled in **continuous time**, although we will later discretize it.

Let $x(t)$ be the position of an object which is attached by a spring to a wall, and let $\dot{x}(t)$ and $\ddot{x}(t)$ be its velocity and acceleration. By **Newton's laws of motion**, we have the following **ordinary differential equation**:

$$m\ddot{x}(t) = u(t) - b\dot{x}(t) - kx(t) \tag{28.62}$$

where $u(t)$ is an externally applied force (e.g., someone tugging on the object), b is the viscous friction coefficient, k is the spring constant, and m is the mass of the object. See Figure 28.21 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, $\boldsymbol{z}(t) = [x(t), \dot{x}(t)]$. Thus the model becomes

$$\dot{\boldsymbol{z}}(t) = \mathbf{A}\boldsymbol{z}(t) + \mathbf{B}\boldsymbol{u}(t) \tag{28.63}$$
$$\boldsymbol{y}(t) = \mathbf{C}\boldsymbol{z}(t) + \mathbf{D}\boldsymbol{u}(t) \tag{28.64}$$

where

$$\begin{pmatrix} \dot{x}(t) \\ \ddot{x}(t) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \end{pmatrix} \begin{pmatrix} x(t) \\ \dot{x}(t) \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{m} \end{pmatrix} u(t) \tag{28.65}$$
$$y(t) = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} x(t) \\ \dot{x}(t) \end{pmatrix} \tag{28.66}$$

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.[5] Here we discuss the **generalized bilinear transform** [ZCC07]. In this approach, we specify a step size Δ and compute

$$\boldsymbol{z}_{k+1} = \underbrace{(\mathbf{I} - \alpha\Delta\mathbf{A})^{-1}(\mathbf{I} + (1-\alpha)\Delta\mathbf{A})}_{\overline{\mathbf{A}}} \boldsymbol{z}_k + \underbrace{\Delta(\mathbf{I} - \alpha\Delta\mathbf{A})^{-1}\mathbf{B}}_{\overline{\mathbf{B}}} \boldsymbol{u}_k \tag{28.67}$$

[4] https://en.wikipedia.org/wiki/State-space_representation#Moving_object_example

[5] See discussion at https://en.wikipedia.org/wiki/Discretization.

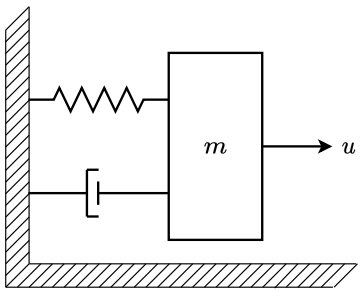


Figure 28.20: Illustration of the spring mass system.

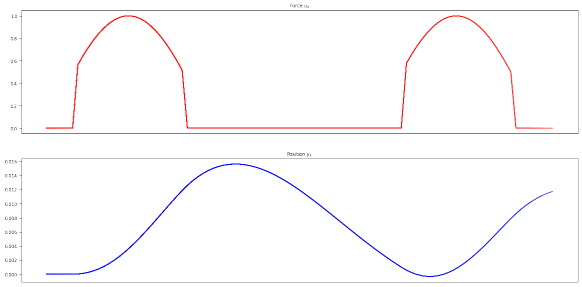


Figure 28.21: Signals genreated by the spring mass system. Top row shows the input force. Bottom row shows the observed location of the end-effector. Adapted from A figure by Sasha Rush. Generated by ssm_spring_demo.ipynb.

If we set $\alpha = 0$, we recover **Euler's method**, which simplifies to

$$\boldsymbol{z}_{k+1} = \underbrace{(\mathbf{I} + \Delta\mathbf{A})}_{\overline{\mathbf{A}}} \boldsymbol{z}_k + \underbrace{\Delta\mathbf{B}}_{\overline{\mathbf{B}}} \boldsymbol{u}_k \tag{28.68}$$

If we set $\alpha = 1$, we recover the **backward Euler method**. If we set $\alpha = \frac{1}{2}$ we get the **bilinear method**, which preserves the stability of the system [ZCC07]; we will use this in Section 28.6. Regardless of how we do the discretization, the resulting discrete time SSM becomes

$$\boldsymbol{z}_k = \overline{\mathbf{A}}\boldsymbol{z}_{k-1} + \overline{\mathbf{B}}\boldsymbol{u}_k \tag{28.69}$$

$$\boldsymbol{y}_k = \mathbf{C}\boldsymbol{z}_k + \mathbf{D}\boldsymbol{u}_k \tag{28.70}$$

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 28.21. 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 28.21. We see that the object's location changes smoothly, since it integrates the force over time.

28.4.3 Example: tracking a moving object in continuous time

In this section, we consider a variant of the example in **??**. We modify the dynamics so that energy is conserved, by ensuring the velocity is constant, and by working in continuous time. Thus the particle moves in a circle, rather than spiralling in towards the origin. (See [Str15] for details.) The dynamics model becomes

$$\boldsymbol{z}_t = \mathbf{A}\boldsymbol{z}_{t-1} + \boldsymbol{\epsilon}_t \tag{28.71}$$

$$\mathbf{A} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \tag{28.72}$$

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.

To see why this results in circular dynamics, we follow a derivation from Gerardo Durán-Martín. We can represent a point in the plane using polar coordinates, (r, θ), or Euclidean coordinates, (x, y). We can switch between these using standard trigonometric identities:

$$x_t = r_t \cos\theta_t, y_t = r_t \sin\theta_t \tag{28.73}$$

The (noiseless) dynamical system has the following form:

$$\begin{pmatrix} \dot{x}_t \\ \dot{y}_t \end{pmatrix} = \begin{pmatrix} y_t \\ -x_t \end{pmatrix} \tag{28.74}$$

We now show that this implies that

$$\dot{r} = 0, \dot{\theta} = -1 \tag{28.75}$$

which means the radius is constant, and the angle changes at a constant rate. To see why, first note that

$$r_t^2 = x_t^2 + y_t^2 \tag{28.76}$$

$$\frac{d}{dt} r_t^2 = \frac{d}{dt}(x_t^2 + y_t^2) \tag{28.77}$$

$$2r_t\dot{r}_t = 2x_t\dot{x}_t + 2y_t\dot{y}_t \tag{28.78}$$

$$\dot{r}_t = \frac{x_t\dot{x}_t + y_t\dot{y}_t}{r_t} \tag{28.79}$$

Also,

$$\tan\theta_t = \frac{y_t}{x_t} \tag{28.80}$$

$$\frac{d}{dt}\tan\theta_t = \frac{d}{dt}\frac{y_t}{x_t} \tag{28.81}$$

$$\dot{\theta}_t \sec^2(\theta_t) = \frac{1}{x_t}\dot{y}_t - \frac{y_t}{x_t^2}\dot{x}_t = \frac{x_t\dot{y}_t - \dot{x}_t y_t}{x_t^2} \tag{28.82}$$

$$\dot{\theta}_t = \frac{x_t\dot{y}_t - \dot{x}_t y_t}{r_t^2} \tag{28.83}$$

Plugging into Equation (28.74), and using the fact that $\cos^2 + \sin^2 = 1$, we have

$$\dot{r} = \frac{(r\cos\theta)(r\sin\theta) - (r\sin\theta)(r\cos\theta)}{r} = 0 \tag{28.84}$$

$$\dot{\theta} = \frac{-(r\cos\theta)(r\cos\theta) - (r\sin\theta)(r\sin\theta)}{r^2} = \frac{-r^2(\cos^2\theta + \sin^2\theta)}{r^2} = -1 \tag{28.85}$$

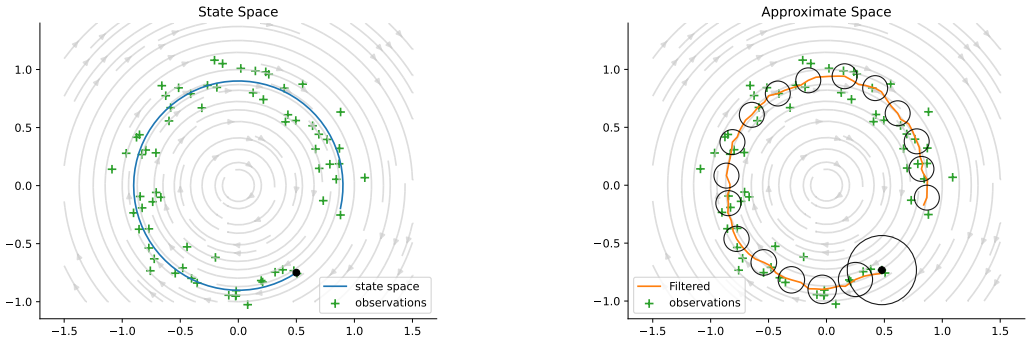
In most applications, we cannot directly see the underlying states. Instead, we just get noisy obervations. Thus we will assume the following observation model:

$$\boldsymbol{y}_t = \mathbf{C}\boldsymbol{z}_t + \boldsymbol{\delta}_t \tag{28.86}$$

$$\mathbf{C} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{28.87}$$

where $\boldsymbol{\delta}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ is the measurement noise. We set $\mathbf{R} = 0.01\mathbf{I}$.

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 each observation in continuous time, using the RK2 method in Section 28.4.1. The result is shown in Figure 28.22. We see that the method is able to filter out the noise, and track the underlying hidden state.



(a) *(b)*

Figure 28.22: 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.py.

28.5 LG-SSMs: parameter estimation

In this section, we discuss how to learn the parameters of an SSM. We mostly focus on the linear-Gaussian case.

There are many approaches for estimating the parameters of state space models. (In the control theory community, this is known as **systems identification** [Lju87].) In the case of linear dynamical systems, many of the methods are similar to techniques used to fit HMMs, discussed in Section 28.2. For example, we can use EM, SGD, or spectral methods, as we discuss below. We can also use Bayesian inference, see **??**.

28.5.1 Training with fully observed data

If we observe the hidden state sequences, we can fit the model by computing the MLEs for the parameters by solving a multivariate linear regression problem for $\boldsymbol{z}_{t-1} \to \boldsymbol{z}_t$ and for $\boldsymbol{z}_t \to \boldsymbol{y}_t$. That is, we can estimate $\mathbf{A}$ by solving the least squares problem $\mathcal{L}(\mathbf{A}) = \sum_{t=1}^{T}(\boldsymbol{z}_t - \mathbf{A}\boldsymbol{z}_{t-1})^2$, Similarly, we can estimate $\mathbf{C}$ by minimizing $\mathcal{L}(\mathbf{C}) = \sum_{t=1}^{T}(\mathbf{C}\boldsymbol{z}_t - \boldsymbol{y}_t)^2$.

We can estimate the system noise covariance $\mathbf{Q}$ from the residuals in predicting $\boldsymbol{z}_t$ from $\boldsymbol{z}_{t-1}$, and estimate the observation noise covariance $\mathbf{R}$ from the residuals in predicting $\boldsymbol{y}_t$ from $\boldsymbol{z}_t$. However, we can set $\mathbf{Q} = \mathbf{I}$ without loss of generality, since an arbitrary noise covariance can be modeled by appropriately modifying $\mathbf{A}$. Also, by analogy with factor analysis, we can require $\mathbf{R}$ to be diagonal without loss of generality. Doing this reduces the number of free parameters and improves numerical stability.

28.5.2 EM for LG-SSM

If we only observe the output sequence, we can compute ML or MAP estimates of the parameters using EM. The method is conceptually quite similar to the Baum-Welch algorithm for HMMs (Section 28.2.1), except we use Kalman smoothing instead of forwards-backwards in the E step, and use different calculations in the M step. In particular, we need to compute the following expected sufficient statistics:

$$\sum_{t=1}^{T} \mathbb{E}[\boldsymbol{z}_t \boldsymbol{z}_t^{\mathsf{T}}], \sum_{t=1}^{T} \mathbb{E}[\boldsymbol{z}_t \boldsymbol{z}_{t+1}^{\mathsf{T}}], \sum_{t=1}^{T} \mathbb{E}[\boldsymbol{z}_t \boldsymbol{y}_t^{\mathsf{T}}] \tag{28.88}$$

where all expectations are wrt $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:T}, \boldsymbol{\theta})$. We can the maximize the expected complete data log likelihood by using a weighted least squares method. The details can be found in in [GH96].

Note that computing the expected sufficient statistics in Equation (28.88) in the inner loop of EM takes $O(T)$ time, which can be expensive for long sequences. In [Mar10], a faster method, known as **ASOS**

(approximate second order statistics), is proposed. In this approach, various statistics are precomputed in a single pass over the sequence, and from then on, all iterations take constant time (independent of T).

28.5.3 Subspace identification methods

EM does not always give satisfactory results, because it is sensitive to the initial parameter estimates. One way to avoid this is to use a different approach known as a **subspace identification** (**SSID**) [OM96; Kat05].

To understand this approach, let us initially assume there is no observation noise and no system noise. In this case, we have $\boldsymbol{z}_t = \mathbf{A}\boldsymbol{z}_{t-1}$ and $\boldsymbol{y}_t = \mathbf{C}\boldsymbol{z}_t$, and hence $\boldsymbol{y}_t = \mathbf{C}\mathbf{A}^{t-1}\boldsymbol{z}_1$. Consequently all the observations must be generated from a $\dim(\boldsymbol{z}_t)$-dimensional linear manifold or subspace. We can identify this subspace using PCA. Once we have an estimate of the $\boldsymbol{z}_t$'s, we can fit the model as if it were fully observed. We can either use these estimates in their own right, or use them to initialize EM. Several papers (e.g., [Smi+00; BK15]) have shown that initializing EM this way gives much better results than initializing EM at random, or just using SSID without EM.

Although the theory only works for noise-free data, we can try to estimate the system noise covariance $\mathbf{Q}$ from the residuals in predicting $\boldsymbol{z}_t$ from $\boldsymbol{z}_{t-1}$, and to estimate the observation noise covariance $\mathbf{R}$ from the residuals in predicting $\boldsymbol{y}_t$ from $\boldsymbol{z}_t$. We can either use these estimates in their own right, or use them to initialize EM. Because this method relies on taking an SVD, it is called a **spectral estimation method**. Similar methods can also be used for HMMs (see Section 28.2.3).

28.5.4 Ensuring stability of the dynamical system

When estimating the dynamics matrix $\mathbf{A}$, it is very useful to impose a constraint on its eigenvalues. To see why this is important, consider the case of no system noise. In this case, the hidden state at time t is given by

$$\boldsymbol{z}_t = \mathbf{A}^t \boldsymbol{z}_1 = \mathbf{U}\boldsymbol{\Lambda}^t \mathbf{U}^{-1}\boldsymbol{z}_1 \tag{28.89}$$

where $\mathbf{U}$ is the matrix of eigenvectors for $\mathbf{A}$, and $\boldsymbol{\Lambda} = \text{diag}(\lambda_i)$ contains the eigenvalues. If any $\lambda_i > 1$, then for large t, $\boldsymbol{z}_t$ will blow up in magnitude. Consequently, to ensure stability, it is useful to require that all the eigenvalues are less than 1 [SBG07]. Of course, if all the eigenvalues are less than 1, then $\mathbb{E}[\boldsymbol{z}_t] = \mathbf{0}$ for large t, so the state will return to the origin. Fortunately, when we add noise, the state becomes non-zero, so the model does not degenerate.

28.5.5 Laplace EM method

In this section we discuss how to fit an SSM with linear-Gaussian latent dynamics and a GLM likelihood using the **Laplace-EM** algorithm. As we discussed in Section 28.5.2, we need to compute the expected sufficient statistics in the E step, and then we can easily maximize the expected complete data log likelihood in the M step. We focus here on the E (inference) step.

First we compute the maximum $\boldsymbol{z}^*_{1:T} = \text{argmax}_{\boldsymbol{z}_{1:T}} \log p(\boldsymbol{z}_{1:T}, \boldsymbol{y}_{1:T})$ using some gradient-based optimizer. We then we compute the Hessian at this point, leveraging the fact that this is block tridiagonal. In particular, let

$$\mathbf{H}_{t,y} = -\nabla^2_{\boldsymbol{z}_t} \log p(\boldsymbol{y}_t|\boldsymbol{z}_t), \mathbf{H}_0 = \nabla \log p(\boldsymbol{z}_0) \tag{28.90}$$

$$\mathbf{H}_{t,11} = -\nabla^2_{\boldsymbol{z}_t} \log p(\boldsymbol{z}_{t+1}|\boldsymbol{z}_t), \mathbf{H}_{t,22} = -\nabla^2_{\boldsymbol{z}_t} \log p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}), \mathbf{H}_{t,12} = -\nabla_{\boldsymbol{z}_t}\nabla_{\boldsymbol{z}_{t+1}} \log p(\boldsymbol{z}_{t+1}|\boldsymbol{z}_t) \tag{28.91}$$

The Gaussian approximation to the posterior has the form $p(\boldsymbol{z}|\boldsymbol{y}) \propto \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu} = \boldsymbol{z}^*_{1:T}$, and the precision matrix is given by

$$\boldsymbol{\Sigma}^{-1} = \begin{pmatrix} \mathbf{J}_{0,0} & \mathbf{J}_{0,1} & \mathbf{0} & \mathbf{0} & \cdots \\ \mathbf{J}_{1,0} & \mathbf{J}_{1,1} & \mathbf{J}_{1,2} & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{J}_{2,1} & \mathbf{J}_{2,2} & \mathbf{J}_{1,2} & \cdots \end{pmatrix} \tag{28.92}$$

where $\mathbf{J}_{t,t} = \mathbf{H}_{t,y} + \mathbf{H}_t + \mathbf{H}_{t,11}$ for t= 0, $\mathbf{J}_{t,t} = \mathbf{H}_{t,y} + \mathbf{H}_{t,11} + \mathbf{H}_{t,22}$ for $t = 1 : T-1$, and $\mathbf{J}_{t,t} = \mathbf{H}_{t,y} + \mathbf{H}_{t,22}$ for $t = T$.

Using this, the log joint has the following form, where $\mathbf{J} = \boldsymbol{\Sigma}^{-1}$ and $\boldsymbol{h} = \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}$:

$$\log p(\boldsymbol{z}, \boldsymbol{y}) = \exp[-\frac{1}{2}\boldsymbol{z}^\mathsf{T}\mathbf{J}\boldsymbol{z} + \boldsymbol{z}^\mathsf{T}\boldsymbol{h} - \log Z(\mathbf{J}, \boldsymbol{h})] \tag{28.93}$$

$$= \exp\left[-\frac{1}{2}\sum_{t=1}^{T}\boldsymbol{z}_t^\mathsf{T}\mathbf{J}_{t,t}\boldsymbol{z}_t - \sum_{t=1}^{t-1}\boldsymbol{z}_{t+1}^\mathsf{T}\mathbf{J}_{t+1,t}\boldsymbol{z}_t + \sum_{t=1}^{T}\boldsymbol{z}_t^\mathsf{T}\boldsymbol{h}_t - \log Z(\mathbf{J}, \boldsymbol{h})\right] \tag{28.94}$$

From this, we can compute the expected sufficient statistics needed for the E step using $\mathbb{E}[\boldsymbol{z}_t|\boldsymbol{y}] = \nabla_{\boldsymbol{h}_t} \log Z(\mathbf{J}, \boldsymbol{h})$, $\mathbb{E}\left[\boldsymbol{z}_t\boldsymbol{z}_t^\mathsf{T}|\boldsymbol{y}\right] = -2\nabla_{\mathbf{J}_{t,t}} \log Z(\mathbf{J}, \boldsymbol{h})$, and $\mathbb{E}\left[\boldsymbol{z}_{t+1}\boldsymbol{z}_t^\mathsf{T}|\boldsymbol{y}\right] = -\nabla_{\mathbf{J}_{+1t,t}} \log Z(\mathbf{J}, \boldsymbol{h})$,

To derive the MLE for the dynamics model from this, we can use a weighted least squares method [GH96]. For the observation model, we can maximize the expected complete data log likelihood by sampling from the posterior.

28.6 Structured State Space Sequence model (S4)

In this section, we briefly discuss a new sequence-to-sequence model known as the **Structured State Space Sequence model** or **S4** [GGR21a; GGR21b; Gu+20]. Our presentation of S4 is based in part on the excellent tutorial [Rus22].

An S4 model is basically a deep stack of (noiseless) linear SSMs (**??**). In between each layer we add pointwise nonlinearities and a linear mapping. Becauses SSMs are recurrent first-order models, we can easily generate (sample) from them in $O(L)$ time, where L is the length of the sequence. This is much faster than the $O(L^2)$ required by standard transformers (**??**). 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 $\boldsymbol{u}(t) \in \mathbb{R}$ to an output sequence $\boldsymbol{y}(t) \in \mathbb{R}^1$ via a sequence of hidden states $\boldsymbol{z}(t) \in \mathbb{R}^N$:

$$\dot{\boldsymbol{z}}(t) = \mathbf{A}\boldsymbol{z}(t) + \mathbf{B}\boldsymbol{u}(t) \tag{28.95}$$

$$\boldsymbol{y}(t) = \mathbf{C}\boldsymbol{z}(t) + \mathbf{D}\boldsymbol{u}(t) \tag{28.96}$$

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 28.4.2. If we set $\alpha = \frac{1}{2}$ in Equation (28.67) we get the **bilinear method**, which preserves the stability of the system [ZCC07]. The result is

$$\boldsymbol{z}_t = \overline{\mathbf{A}}\boldsymbol{z}_{t-1} + \overline{\mathbf{B}}\boldsymbol{u}_t \tag{28.97}$$

$$\boldsymbol{y}_t = \overline{\mathbf{C}}\boldsymbol{z}_t \tag{28.98}$$

$$\overline{\mathbf{A}} = (\mathbf{I} - \Delta/2 \cdot \mathbf{A})^{-1}(\mathbf{I} + \Delta/ \cdot \mathbf{A}) \in \mathbb{R}^{N \times N} \tag{28.99}$$

$$\overline{\mathbf{B}} = (\mathbf{I} - \Delta/2 \cdot \mathbf{A})^{-1}\Delta\mathbf{B} \in \mathbb{R}^N \tag{28.100}$$

$$\overline{\mathbf{C}} = \mathbf{C} \in \mathbb{R}^{N \times 1} \tag{28.101}$$

We now discuss what is happening inside the LSSL layer. Let us assume the initial state is $\boldsymbol{z}_{-1} = \mathbf{0}$. We can unroll the recursion to get

$$\boldsymbol{z}_0 = \overline{\mathbf{B}}u_0,\ \boldsymbol{z}_1 = \overline{\mathbf{A}\mathbf{B}}u_0 + \overline{\mathbf{B}}u_1,\ \boldsymbol{z}_2 = \overline{\mathbf{A}}^2\overline{\mathbf{B}}u_0 + \overline{\mathbf{A}\mathbf{B}}u_1 + \overline{\mathbf{B}}u_2,\ \cdots \tag{28.102}$$

$$y_0 = \overline{\mathbf{C}\mathbf{B}}u_0,\ y_1 = \overline{\mathbf{C}\mathbf{A}\mathbf{B}}u_0 + \overline{\mathbf{C}\mathbf{B}}u_1,\ y_2 = \overline{\mathbf{C}\mathbf{A}}^2\overline{\mathbf{B}}u_0 + \overline{\mathbf{C}\mathbf{A}\mathbf{B}}u_1 + \overline{\mathbf{C}\mathbf{B}}u_2, \cdots \tag{28.103}$$

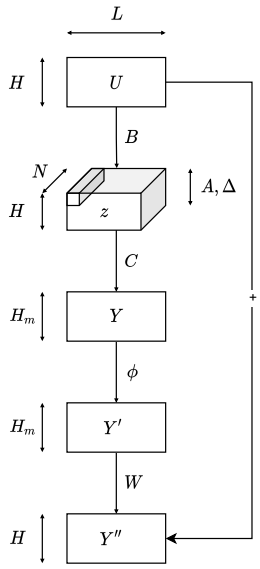


Figure 28.23: Illustration of one S4 block. The input $\mathbf{X}$ *has* H *sequences, each of length* L*. These get mapped (in parallel for each of the sequences) to the output* $\mathbf{Y}$ *by a (noiseless) linear SSM with state size* N *and parameters* $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$*. The output* $\mathbf{Y}$ *is mapped pointwise through a nonlinearity* $\boldsymbol{\phi}$ *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*.*

We see that $\boldsymbol{z}_t$ is computing a weighted sum of all the past inputs, where the weights are controlled by powers of the $\mathbf{A}$ matrix. (See also the discussion of subspace identification techniques in Section 28.5.3.) It turns out that we can define $\mathbf{A}$ to have a special structure, known as **HiPPO** (High-order Polynomial Projection Operator) [Gu+20], such that (1) it ensures $\boldsymbol{z}_t$ embeds "relevant" parts of the past history in a compact manner, (2) enables recursive computation of $\boldsymbol{z}_{1:L}$ in $O(N)$ time per step, instead of the naive $O(N^2)$ time for the matrix vector multiply; and (3) only has $O(3N)$ (complex-valued) parameters to learn.

However, recursive computation still takes time linear in L. At training time, when all inputs to each location are already available, we can further speed thing up by recognizing that the output sequence can be computed in parallel using a convolution:

$$y_k = \overline{\mathbf{C}}\overline{\mathbf{A}}^k\overline{\mathbf{B}}u_0 + \overline{\mathbf{C}}\overline{\mathbf{A}}^{k-1}\overline{\mathbf{B}}u_1 + \cdots + \overline{\mathbf{C}}\overline{\mathbf{B}}u_k \tag{28.104}$$

$$\boldsymbol{y} = \overline{\mathbf{K}} \odot \boldsymbol{u} \tag{28.105}$$

$$\overline{\mathbf{K}} = (\overline{\mathbf{C}}\overline{\mathbf{B}}, \overline{\mathbf{C}}\overline{\mathbf{A}}\overline{\mathbf{B}}, \cdots, \overline{\mathbf{C}}\overline{\mathbf{A}}^{L-1}\overline{\mathbf{B}}) \tag{28.106}$$

We can compute the convolution kernel $\overline{\mathbf{K}}$ matrix in $O(N + L)$ time and space, using the S4 representation, and then use FFT to efficiently compute the output. Unfortunately the details of how to do this are rather complicated, so we refer the reader to [GGR21a].

Once we have constructed an LSSL layer, we can process a stack of H sequences independently in parallel by replicating the above process with different parameters, for $h = 1 : H$. If we let each of the H $\mathbf{C}$ matrices be of size $\mathbf{N} \times \mathbf{M}$, so they return a vector of channels instead of a scalar at each location, the overall mapping is from $\boldsymbol{u}_{1:L} \in \mathbb{R}^{H \times L}$ to $\boldsymbol{y}_{1:L} \in \mathbb{R}^{HM \times L}$. We can add a pointwise nonlinearity to the output and then apply a projection matrix $\mathbf{W} \in \mathbb{R}^{MH \times H}$ to linearly combine the channels and map the result back to size $\boldsymbol{y}_{1:L} \in \mathbb{R}^{H \times L}$, as shown in Figure 28.23. This overall block can then be repeated a desired number of times. The input to the whole model is an encoder matrix which embeds each token, and the output is a decoder that creates the softmax layer at each location, as in the transformer. We can now learn the $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$, and $\mathbf{W}$ matrices (as well as the step size Δ) for each layer using backpropagation, using whatever loss function we want on the output layer.

Chapter 29

Graph learning

Chapter 30

Non-parametric Bayesian models

Chapter 31

Representation learning

Chapter 32

Interpretability

Part VI

Decision making

Chapter 33

Multi-step decision problems

Chapter 34

Reinforcement learning

Chapter 35

Causality

Bibliography

[AC17] S. Aminikhanghahi and D. J. Cook. "A Survey of Methods for Time Series Change Point Detection". en. In: *Knowl. Inf. Syst.* 51.2 (May 2017), pp. 339–367.

[ACL16] L. Aitchison, N. Corradi, and P. E. Latham. "Zipf's Law Arises Naturally When There Are Underlying, Unobserved Variables". en. In: *PLoS Comput. Biol.* 12.12 (Dec. 2016), e1005110.

[AD19a] H. Asi and J. C. Duchi. "Modeling simple structures and geometry for better stochastic optimization algorithms". In: *AISTATS*. 2019.

[AD19b] H. Asi and J. C. Duchi. "Stochastic (Approximate) Proximal Point Methods: Convergence, Optimality, and Adaptivity". In: *SIAM J. Optim.* (2019).

[AD19c] H. Asi and J. C. Duchi. "The importance of better models in stochastic optimization". en. In: *PNAS* 116.46 (Nov. 2019), pp. 22924–22930.

[Ada00] L. Adamic. *Zipf, Power-laws, and Pareto - a ranking tutorial*. Tech. rep. 2000.

[AE+20] D. Agudelo-España, S. Gomez-Gonzalez, S. Bauer, B. Schölkopf, and J. Peters. "Bayesian Online Prediction of Change Points". In: *UAI*. Vol. 124. Proceedings of Machine Learning Research. PMLR, 2020, pp. 320–329.

[AEM18] Ö. D. Akyildiz, V. Elvira, and J. Miguez. "The Incremental Proximal Method: A Probabilistic Perspective". In: *ICASSP*. 2018.

[AG11] A. Allahverdyan and A. Galstyan. "Comparative Analysis of Viterbi Training and Maximum Likelihood Estimation for HMMs". In: *NIPS*. 2011, pp. 1674–1682.

[AH09] I. Arasaratnam and S. Haykin. "Cubature Kalman Filters". In: *IEEE Trans. Automat. Contr.* 54.6 (June 2009), pp. 1254–1269.

[AHE07] I. Arasaratnam, S. Haykin, and R. J. Elliott. "Discrete-Time Nonlinear Filtering Algorithms Using Gauss–Hermite Quadrature". In: *Proc. IEEE* 95.5 (May 2007), pp. 953–977.

[AHK12] A. Anandkumar, D. Hsu, and S. M. Kakade. "A Method of Moments for Mixture Models and Hidden Markov Models". In: *COLT*. Vol. 23. Proceedings of Machine Learning Research. Edinburgh, Scotland: PMLR, 2012, pp. 33.1–33.34.

[Aky+19] Ö. D. Akyildiz, É. Chouzenoux, V. Elvira, and J. Míguez. "A probabilistic incremental proximal gradient method". In: *IEEE Signal Process. Lett.* 26.8 (2019).

[ALK06] C. Albers, M. Leisink, and H. Kappen. "The Cluster Variation Method for Efficient Linkage Analysis on Extended Pedigrees". In: *BMC Bioinformatics* 7 (2006).

[AM07] R. P. Adams and D. J. C. MacKay. "Bayesian Online Changepoint Detection". In: (Oct. 2007). arXiv: 0710.3742 [stat.ML].

[Ana+14] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky. "Tensor Decompositions for Learning Latent Variable Models". In: *JMLR* 15 (2014), pp. 2773–2832.

[And01] J. L. Anderson. "An Ensemble Adjustment Kalman Filter for Data Assimilation". In: *Mon. Weather Rev.* 129.12 (Dec. 2001), pp. 2884–2903.

[Ara+09] A. Aravkin, B. Bell, J. Burke, and G. Pillonetto. *An L1-Laplace Robust Kalman Smoother*. Tech. rep. U. Washington, 2009.

[Ara10] A. Aravkin. *Student's t Kalman Smoother*. Tech. rep. U. Washington, 2010.

[Ara+17] A. Aravkin, J. V. Burke, L. Ljung, A. Lozano, and G. Pillonetto. "Generalized Kalman smoothing: Modeling and algorithms". In: *Automatica* 86 (Dec. 2017), pp. 63–86.

[Arm05] H. Armstrong. "Bayesian estimation of decomposable Gaussian graphical models". PhD thesis. UNSW, 2005.

[AS19] B. G. Anderson and S. Sojoudi. "Global Optimality Guarantees for Nonconvex Unsupervised Video Segmentation". In: *57th Annual Allerton Conference on Communication, Control, and Computing* (2019).

[Asa00] C. Asavathiratham. "The Influence Model: A Tractable Representation for the Dynamics of Networked Markov Chains". PhD thesis. MIT, Dept. EECS, 2000.

[Ban+05] A. Banerjee, I. S. Dhillon, J. Ghosh, and S. Sra. "Clustering on the unit hypersphere using von Mises-Fisher distributions". In: *JMLR*. 2005, pp. 1345–1382.

[Bas+01] S. Basu, T. Choudhury, B. Clarkson, and A. Pentland. *Learning Human Interactions with the Influence Model*. Tech. rep. 539. MIT Media Lab, 2001.

[Bau+70] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. "A maximization technique occuring in the statistical analysis of probabalistic functions in Markov chains". In: *The Annals of Mathematical Statistics* 41 (1970), pp. 164–171.

[BC94] P. Baldi and Y. Chauvin. "Smooth online learning algorithms for Hidden Markov Models". In: *Neural Computation* 6 (1994), pp. 305–316.

[BCH20] M. Briers, M. Charalambides, and C. Holmes. "Risk scoring calculation for the current NHSx contact tracing app". In: (May 2020). arXiv: 2005.11057 [cs.CY].

[Ber15] D. P. Bertsekas. "Incremental Gradient, Subgradient, and Proximal Methods for Convex Optimization: A Survey". In: (July 2015). arXiv: 1507.01030 [cs.SY].

[BH92] D. Barry and J. A. Hartigan. "Product partition models for change point problems". In: *Annals of statistics* 20 (1992), pp. 260–279.

[BHO75] P. J. Bickel, E. A. Hammel, and J. W. O'connell. "Sex bias in graduate admissions: data from berkeley". en. In: *Science* 187.4175 (Feb. 1975), pp. 398–404.

[Bil01] J. A. Bilmes. *Graphical Models and Automatic Speech Recognition*. Tech. rep. UWEETR-2001-0005. Univ. Washington, Dept. of Elec. Eng., 2001.

[Bil10] J. Bilmes. "Dynamic Graphical Models". In: *IEEE Signal Processing Magazine* 27.6 (2010), pp. 29–42.

[Bis06] C. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[Bit16] S. Bitzer. *The UKF exposed: How it works, when it works and when it's better to sample*. 2016.

[BK04] Y. Boykov and V. Kolmogorov. "An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision". en. In: *IEEE PAMI* 26.9 (Sept. 2004), pp. 1124–1137.

[BK15] D. Belanger and S. Kakade. "A Linear Dynamical System Model for Text". en. In: *ICML*. June 2015, pp. 833–842.

[BKR11] A. Blake, P. Kohli, and C. Rother, eds. *Advances in Markov Random Fields for Vision and Image Processing*. MIT Press, 2011.

[BL06] K. Bryan and T. Leise. "The $25,000,000,000 Eigenvector: The Linear Algebra behind Google". In: *SIAM Review* 48.3 (2006).

[BLBV12] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. "Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription". In: *ICML*. June 2012.

[BMR97] J. Binder, K. Murphy, and S. Russell. "Space-efficient inference in dynamic probabilistic networks". In: *IJCAI*. 1997.

[BNJ03] D. Blei, A. Ng, and M. Jordan. "Latent Dirichlet allocation". In: *JMLR* 3 (2003), pp. 993–1022.

[Boh92] D. Bohning. "Multinomial logistic regression algorithm". In: *Annals of the Inst. of Statistical Math.* 44 (1992), pp. 197–200.

[Bou+17] T. Bouwmans, A. Sobral, S. Javed, S. K. Jung, and E.-H. Zahzah. "Decomposition into low-rank plus additive matrices for background/foreground separation: A review for a comparative evaluation with a large-scale dataset". In: *Computer Science Review* 23 (Feb. 2017), pp. 1–71.

[Bra96] M. Brand. *Coupled hidden Markov models for modeling interacting processes*. Tech. rep. 405. MIT Lab for Perceptual Computing, 1996.

[Bre67] L. M. Bregman. "The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming". In: *USSR Computational Mathematics and Mathematical Physics* 7.3 (Jan. 1967), pp. 200–217.

[BSF88] Y. Bar-Shalom and T. Fortmann. *Tracking and data association*. Academic Press, 1988.

[BT03] A. Beck and M. Teoulle. "Mirror descent and nonlinear projected subgradient methods for convex optimization". In: *Operations Research Letters* 31.3 (2003), pp. 167–175.

[BT09] A Beck and M Teboulle. "A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems". In: *SIAM J. Imaging Sci.* 2.1 (Jan. 2009), pp. 183–202.

[BVW02] H. Bui, S. Venkatesh, and G. West. "Policy Recognition in the Abstract Hidden Markov Model". In: *JAIR* 17 (2002), pp. 451–499.

[BVZ01] Y. Boykov, O. Veksler, and R. Zabih. "Fast Approximate Energy Minimization via Graph Cuts". In: *IEEE PAMI* 23.11 (2001).

[BW20] G. J. J. van den Burg and C. K. I. Williams. "An Evaluation of Change Point Detection Algorithms". In: (Mar. 2020). arXiv: 2003.06222 [stat.ML].

[BWL19] Y. Bai, Y.-X. Wang, and E. Liberty. "ProxQuant: Quantized Neural Networks via Proximal Operators". In: *ICLR*. 2019.

[Can+11] E. J. Candes, X. Li, Y. Ma, and J. Wright. "Robust Principal Component Analysis?" In: *JACM* 58.3 (June 2011), 11:1–11:37.

[CGJ17] Y. Cherapanamjeri, K. Gupta, and P. Jain. "Nearly-optimal Robust Matrix Completion". In: *ICML*. 2017.

[Cha+20] P. E. Chang, W. J. Wilkinson, M. E. Khan, and A. Solin. "Fast Variational Learning in State-Space Gaussian Process Models". In: *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*. Sept. 2020, pp. 1–6.

[Che+16] T. Chen, B. Xu, C. Zhang, and C. Guestrin. "Training Deep Nets with Sublinear Memory Cost". In: (Apr. 2016). arXiv: 1604.06174 [cs.LG].

[Chi14] S. Chiappa. "Explicit-Duration Markov Switching Models". In: *Foundations and Trends in Machine Learning* 7.6 (2014), pp. 803–886.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *An Introduction to Algorithms*. MIT Press, 1990.

[Dai+17] H. Dai, B. Dai, Y.-M. Zhang, S. Li, and L. Song. "Recurrent Hidden Semi-Markov Model". In: *ICLR*. 2017.

[DDDM04] I Daubechies, M Defrise, and C De Mol. "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint". In: *Commun. Pure Appl. Math.* Advances in E 57.11 (Nov. 2004), pp. 1413–1457.

[DeG70] M. DeGroot. *Optimal Statistical Decisions*. McGraw-Hill, 1970.

[DHS11] J. Duchi, E. Hazan, and Y. Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *JMLR* 12 (2011), pp. 2121–2159.

[Don95] D. L. Donoho. "De-noising by soft-thresholding". In: *IEEE Trans. Inf. Theory* 41.3 (May 1995), pp. 613–627.

[Dur+98] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[ECM18] P. Etoori, M. Chinnakotla, and R. Mamidi. "Automatic Spelling Correction for Resource-Scarce Languages using Deep Learning". In: *Proceedings of ACL 2018, Student Research Workshop*. Melbourne, Australia: Association for Computational Linguistics, 2018, pp. 146–152.

[Eks+18] C. Eksombatchai, P. Jindal, J. Z. Liu, Y. Liu, R. Sharma, C. Sugnet, M. Ulrich, and J. Leskovec. "Pixie: A System for Recommending 3+ Billion Items to 200+ Million Users in Real-Time". In: *WWW*. 2018.

[Eve09] G. Evensen. *Data Assimilation: The Ensemble Kalman Filter*. en. 2nd ed. 2009 edition. Springer, Aug. 2009.

[Fan+17] H. Fang, N. Tian, Y. Wang, M. Zhou, and M. A. Haile. "Nonlinear Bayesian Estimation: From Kalman Filtering to a Broader Horizon". In: (Dec. 2017). arXiv: 1712.01406 [cs.SY].

[Fat18] S. Fattahi. "Exact Guarantees on the Absence of Spurious Local Minima for Non-negative Robust Principal Component Analysis". In: *JMLR* (2018).

[FG02] M. Fishelson and D. Geiger. "Exact genetic linkage computations for general pedigrees". In: *BMC Bioinformatics* 18 (2002).

[FGL00] N. Friedman, D. Geiger, and N. Lotner. "Likelihood computation with value abstraction". In: *UAI*. 2000.

[FK13] M Frei and H. R. Künsch. "Bridging the ensemble Kalman and particle filters". In: *Biometrika* 100.4 (Dec. 2013), pp. 781–800.

[FL07] P. Fearnhead and Z. Liu. "Online Inference for Multiple Changepoint Problems". In: *J. of Royal Stat. Soc. Series B* 69 (2007), pp. 589–605.

[FL11] P. Fearnhead and Z. Liu. "Efficient Bayesian analysis of multiple changepoint models with dependence across segments". In: *Statistics and Computing* 21.2 (2011), pp. 217–229.

[For+95] J. Forbes, T. Huang, K. Kanazawa, and S. Russell. "The BATmobile: Towards a Bayesian Automated Taxi". In: *IJCAI*. 1995.

[Fot+14] N. Foti, J. Xu, D. Laird, and E. Fox. "Stochastic variational inference for hidden Markov models". In: *NIPS*. 2014, pp. 3599–3607.

[FST98] S. Fine, Y. Singer, and N. Tishby. "The Hierarchical Hidden Markov Model: Analysis and Applications". In: *Machine Learning* 32 (1998), p. 41.

[GGR21a] A. Gu, K. Goel, and C. Ré. "Efficiently Modeling Long Sequences with Structured State Spaces". In: (Oct. 2021). arXiv: 2111.00396 [cs.LG].

[GGR21b] A. Gu, K. Goel, and C. Ré. "Efficiently Modeling Long Sequences with Structured State Spaces". In: (Oct. 2021). arXiv: 2111.00396 [cs.LG].

[GH12] F. Gustafsson and G. Hendeby. "Some Relations Between Extended and Unscented Kalman Filters". In: *IEEE Trans. Signal Process.* 60.2 (Feb. 2012), pp. 545–555.

[GH96] Z. Ghahramani and G. Hinton. *Parameter estimation for linear dynamical systems*. Tech. rep. CRG-TR-96-2. Dept. Comp. Sci., Univ. Toronto, 1996.

[GJ07] A. Globerson and T. Jaakkola. "Approximate inference using planar graph decomposition". In: *AISTATS*. 2007.

[GJ97] Z. Ghahramani and M. Jordan. "Factorial Hidden Markov Models". In: *Machine Learning* 29 (1997), pp. 245–273.

[Gol+17] N. Gold, M. G. Frasch, C. Herry, B. S. Richardson, and X. Wang. "A Doubly Stochastic Change Point Detection Algorithm for Noisy Biological Signals". en. In: *Front. Physiol.* (Oct. 2017), p. 106088.

[Gop98] A. Gopnik. "Explanation as Orgasm". In: *Minds and Machines* 8.1 (1998), pp. 101–118.

[GPS89] D. Greig, B. Porteous, and A. Seheult. "Exact maximum a posteriori estimation for binary images". In: *J. of Royal Stat. Soc. Series B* 51.2 (1989), pp. 271–279.

[GS04] T. Griffiths and M. Steyvers. "Finding scientific topics". In: *PNAS* 101 (2004), pp. 5228–5235.

[GT06] T. Griffiths and J. Tenenbaum. "Optimal predictions in everyday cognition". In: *Psychological Science* 17.9 (2006), pp. 767–773.

[Gu+20] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Re. "HiPPO: Recurrent Memory with Optimal Polynomial Projections". In: *NIPS* 33 (2020).

[GW08] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Second. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2008.

[Hag+17] M. Hagen, M. Potthast, M. Gohsen, A. Rathgeber, and B. Stein. "A Large-Scale Query Spelling Correction Corpus". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '17. Shinjuku, Tokyo, Japan: ACM, 2017, pp. 1261–1264.

[HBB10] M. Hoffman, D. Blei, and F. Bach. "Online learning for latent Dirichlet allocation". In: *NIPS*. 2010.

[HKZ12] D. Hsu, S. Kakade, and T. Zhang. "A spectral algorithm for learning hidden Markov models". In: *J. of Computer and System Sciences* 78.5 (2012), pp. 1460–1480.

[Hoe+99] J. Hoeting, D. Madigan, A. Raftery, and C. Volinsky. "Bayesian Model Averaging: A Tutorial". In: *Statistical Science* 4.4 (1999).

[HP10] J. Hacker and P. Pierson. *Winner-Take-All Politics: How Washington Made the Rich Richer — and Turned Its Back on the Middle Class*. Simon & Schuster, 2010.

[HS08] T. Hazan and A. Shashua. "Convergent message-passing algorithms for inference over general graphs with convex free energy". In: *UAI*. 2008.

[Hu+00] M. Hu, C. Ingram, M.Sirski, C. Pal, S. Swamy, and C. Patten. *A Hierarchical HMM Implementation for Vertebrate Gene Splice Site Prediction*. Tech. rep. Dept. Computer Science, Univ. Waterloo, 2000.

[Hua+17] Y Huang, Y Zhang, N Li, Z Wu, and J. A. Chambers. "A Novel Robust Student's t-Based Kalman Filter". In: *IEEE Trans. Aerosp. Electron. Syst.* 53.3 (June 2017), pp. 1545–1554.

[Hua+19] Y. Huang, Y. Zhang, Y. Zhao, and J. A. Chambers. "A Novel Robust Gaussian–Student's t Mixture Distribution Based Kalman Filter". In: *IEEE Trans. Signal Process.* 67.13 (July 2019), pp. 3606–3620.

[IX00] K Ito and K Xiong. "Gaussian filters for nonlinear filtering problems". In: *IEEE Trans. Automat. Contr.* 45.5 (May 2000), pp. 910–927.

[Jay03] E. T. Jaynes. *Probability theory: the logic of science*. Cambridge university press, 2003.

[Jia+13] Y. Jia, J. T. Abbott, J. L. Austerweil, T. Griffiths, and T. Darrell. "Visual Concept Learning: Combining Machine Vision and Bayesian Generalization on Concept Hierarchies". In: *NIPS*. 2013.

[JJ00] T. S. Jaakkola and M. I. Jordan. "Bayesian parameter estimation via variational methods". In: *Statistics and Computing* 10 (2000), pp. 25–37.

[JJ96] T. Jaakkola and M. Jordan. "A variational approach to Bayesian logistic regression problems and their extensions". In: *AISTATS*. 1996.

[JJ99] T. Jaakkola and M. Jordan. "Variational probabilistic inference and the QMR-DT network". In: *JAIR* 10 (1999), pp. 291–322.

[JM00] D. Jurafsky and J. H. Martin. *Speech and language processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice-Hall, 2000.

[JM08] D. Jurafsky and J. H. Martin. *Speech and language processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 2nd edition. Prentice-Hall, 2008.

[Joh12] M. J. Johnson. "A Simple Explanation of A Spectral Algorithm for Learning Hidden Markov Models". In: (Apr. 2012). arXiv: 1204.2477 [stat.ME].

[JU97] S. Julier and J. Uhlmann. "A New Extension of the Kalman Filter to Nonlinear Systems". In: *Proc. of AeroSense: The 11th Intl. Symp. on Aerospace/Defence Sensing, Simulation and Controls*. 1997.

[Kat05] T. Katayama. *Subspace Methods for Systems Identification*. Springer Verlag, 2005.

[KF09] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[Kha12] M. E. Khan. "Variational Bounds for Learning and Inference with Discrete Data in Latent Gaussian Models". PhD thesis. UBC, 2012.

[KJ12] J. Z. Kolter and T. S. Jaakkola. "Approximate Inference in Additive Factorial HMMs with Application to Energy Disaggregation". In: *AISTATS*. 2012.

[KJD18] J. Knoblauch, J. Jewson, and T. Damoulas. "Doubly Robust Bayesian Inference for Non-Stationary Streaming Data with β-Divergences". In: *NIPS*. June 2018.

[KL17] M. E. Khan and W. Lin. "Conjugate-Computation Variational Inference : Converting Variational Inference in Non-Conjugate Models to Inferences in Conjugate Models". In: *AISTATS*. 2017.

[KM00] J. Kwon and K. Murphy. *Modeling Freeway Traffic with Coupled HMMs*. Tech. rep. Univ. California, Berkeley, 2000.

[KNH11] K. B. Korb, E. P. Nyberg, and L. Hope. "A new causal power theory". In: *Causality in the Sciences*. Oxford University Press, 2011.

[Kol06] V. Kolmogorov. "Convergent Tree-reweighted Message Passing for Energy Minimization". In: *IEEE PAMI* 28.10 (2006), pp. 1568–1583.

[Koy+10] S. Koyama, L. C. Pérez-Bolde, C. R. Shalizi, and R. E. Kass. "Approximate Methods for State-Space Models". en. In: *JASA* 105.489 (Mar. 2010), pp. 170–180.

[Kru13] J. K. Kruschke. "Bayesian estimation supersedes the t test". In: *J. Experimental Psychology: General* 142.2 (2013), pp. 573–603.

[Lau92] S. L. Lauritzen. "Propagation of probabilities, means and variances in mixed graphical association models". In: *JASA* 87.420 (1992), pp. 1098–1108.

[LDZ11] Y Li, H Duan, and C. X. Zhai. "Cloudspeller: Spelling correction for search queries by using a unified hidden markov model with web-scale resources". In: *SIGIR*. 2011.

[LDZ12] Y. Li, H. Duan, and C. Zhai. "A Generalized Hidden Markov Model with Discriminative Training for Query Spelling Correction". In: *SIGIR*. 2012, pp. 611–620.

[LGMT11] F. Le Gland, V. Monbet, and V.-D. Tran. "Large Sample Asymptotics for the Ensemble Kalman Filter". In: *Oxford Handbook of Nonlinear Filtering*. Ed. by D Crisan And. 2011.

[Li+14] A. Q. Li, A. Ahmed, S. Ravi, and A. J. Smola. "Reducing the sampling complexity of topic models". In: *KDD*. ACM, 2014, pp. 891–900.

[Li+17] T.-C. Li, J.-Y. Su, W. Liu, and J. M. Corchado. "Approximate Gaussian conjugacy: parametric recursive filtering under nonlinearity, multimodality, uncertainty, and constraint, and beyond". In: *Frontiers of Information Technology & Electronic Engineering* 18.12 (Dec. 2017), pp. 1913–1939.

[Li+19] X. Li, L. Vilnis, D. Zhang, M. Boratko, and A. McCallum. "Smoothing the Geometry of Probabilistic Box Embeddings". In: *ICLR*. 2019.

[Li+20] R. Li, S. Pei, B. Chen, Y. Song, T. Zhang, W. Yang, and J. Shaman. "Substantial undocumented infection facilitates the rapid dissemination of novel coronavirus (SARS-CoV2)". en. In: *Science* (Mar. 2020).

[Lia+07] L. Liao, D. J. Patterson, D. Fox, and H. Kautz. "Learning and Inferring Transportation Routines". In: *Artificial Intelligence* 171.5 (2007), pp. 311–331.

[Lju87] L. Ljung. *System Identificiation: Theory for the User*. Prentice Hall, 1987.

[LKJ09] D. Lewandowski, D. Kurowicka, and H. Joe. "Generating random correlation matrices based on vines and extended onion method". In: *J. Multivar. Anal.* 100.9 (Oct. 2009), pp. 1989–2001.

[LL15] H. Li and Z. Lin. "Accelerated Proximal Gradient Methods for Nonconvex Programming". In: *NIPS*. 2015, pp. 379–387.

[LL18] Z. Li and J. Li. "A Simple Proximal Stochastic Gradient Method for Nonsmooth Nonconvex Optimization". In: (Feb. 2018). arXiv: 1802.04477 [math.OC].

[LLT89] K. Lange, R. Little, and J. Taylor. "Robust Statistical Modeling Using the T Disribution". In: *JASA* 84.408 (1989), pp. 881–896.

[LM06] A. Langville and C. Meyer. "Updating Markov chains with an eye on Google's PageRank". In: *SIAM J. on Matrix Analysis and Applications* 27.4 (2006), pp. 968–987.

[LMW17] L Landrieu, C Mallet, and M Weinmann. "Comparison of belief propagation and graph-cut approaches for contextual classification of 3D lidar point cloud data". In: *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. July 2017, pp. 2768–2771.

[Lof15] P. Lofgren. "Efficient Algorithms for Personalized PageRank". PhD thesis. Stanford, 2015.

[Lov+20] T. Lovett, M. Briers, M. Charalambides, R. Jersakova, J. Lomax, and C. Holmes. "Inferring proximity from Bluetooth Low Energy RSSI with Unscented Kalman Smoothers". In: (July 2020). arXiv: 2007.05057 [eess.SP].

[LS88] S. L. Lauritzen and D. J. Spiegelhalter. "Local computations with probabilities on graphical structures and their applications to expert systems". In: *J. of Royal Stat. Soc. Series B* B.50 (1988), pp. 127–224.

[LT79] R. J. Lipton and R. E. Tarjan. "A separator theorem for planar graphs". In: *SIAM Journal of Applied Math* 36 (1979), pp. 177–189.

[LW16] C. Louizos and M. Welling. "Structured and Efficient Variational Deep Learning with Matrix Gaussian Posteriors". In: *ICML*. 2016.

[Mah08] H. Mahmoud. *Polya Urn Models*. en. 1 edition. Chapman and Hall/CRC, June 2008.

[Mai15] J Mairal. "Incremental Majorization-Minimization Optimization with Application to Large-Scale Machine Learning". In: *SIAM J. Optim.* 25.2 (Jan. 2015), pp. 829–855.

[Mar10] J. Martens. "Learning the Linear Dynamical System with ASOS". In: *ICML*. ICML'10. Haifa, Israel: Omnipress, 2010, pp. 743–750.

[Mat14] C. Mattfeld. "Implementing spectral methods for hidden Markov models with real-valued emissions". MA thesis. ETH Zurich, Apr. 2014.

[Mav16] Mavrogonatou, L And Vyshemirsky, "Sequential Importance Sampling for Online Bayesian Changepoint Detection". In: *22nd International Conference on Computational Statistics*. 2016.

[May79] P. Maybeck. *Stochastic models, estimation, and control*. Academic Press, 1979.

[McE20] R. McElreath. *Statistical Rethinking: A Bayesian Course with Examples in R and Stan (2nd edition)*. en. Chapman and Hall/CRC, 2020.

[MH97] C. Meek and D. Heckerman. "Structure and Parameter Learning for Causal Independence and Causal Interaction Models". In: *UAI*. 1997, pp. 366–375.

[Min01] T. Minka. *Statistical Approaches to Learning and Discovery 10-602: Homework assignment 2, question 5*. Tech. rep. CMU, 2001.

[Mit97] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[MKM11] B. Marlin, E. Khan, and K. Murphy. "Piecewise Bounds for Estimating Bernoulli-Logistic Latent Gaussian Models". In: *ICML*. 2011.

[MKS21] K. Murphy, A. Kumar, and S. Serghiou. "Risk score learning for COVID-19 contact tracing apps". In: *Machine Learning for Healthcare*. Apr. 2021.

[MM01] T. K. Marks and J. R. Movellan. *Diffusion networks, products of experts, and factor analysis*. Tech. rep. University of California San Diego, 2001.

[MP01] K. Murphy and M. Paskin. "Linear time inference in hierarchical HMMs". In: *NIPS*. 2001.

[Mur02] K. Murphy. "Dynamic Bayesian Networks: Representation, Inference and Learning". PhD thesis. Dept. Computer Science, UC Berkeley, 2002.

[Mur22] K. P. Murphy. *Probabilistic Machine Learning: An introduciton*. MIT Press, 2022.

[Nal+19] E. Nalisnick, A. Matsukawa, Y. W. Teh, D. Gorur, and B. Lakshminarayanan. "Do Deep Generative Models Know What They Don't Know?" In: *ICLR*. 2019.

[Nea92] R. Neal. "Connectionist learning of belief networks". In: *Artificial Intelligence* 56 (1992), pp. 71–113.

[Nef+02] A. Nefian, L. Liang, X. Pi, X. Liu, and K. Murphy. "Dynamic Bayesian Networks for Audio-Visual Speech Recognition". In: *J. Applied Signal Processing* (2002).

[Nit14] A. Nitanda. "Stochastic Proximal Gradient Descent with Acceleration Techniques". In: *NIPS*. 2014, pp. 1574–1582.

[NY83] A. Nemirovski and D. Yudin. *Problem Complexity and Method Efficiency in Optimization*. Wiley, 1983.

[Obe+19] F. Obermeyer, E. Bingham, M. Jankowiak, J. Chiu, N. Pradhan, A. Rush, and N. Goodman. "Tensor Variable Elimination for Plated Factor Graphs". In: *ICML*. Feb. 2019.

[ODK96] M. Ostendorf, V. Digalakis, and O. Kimball. "From HMMs to segment models: a unified view of stochastic modeling for speech recognition". In: *IEEE Trans. on Speech and Audio Processing* 4.5 (1996), pp. 360–378.

[Oll18] Y. Ollivier. "Online natural gradient as a Kalman filter". en. In: *Electron. J. Stat.* 12.2 (2018), pp. 2930–2961.

[OM96] P. V. Overschee and B. D. Moor. *Subspace Identification for Linear Systems: Theory, Implementation, Applications*. Kluwer Academic Publishers, 1996.

[PB+14] N. Parikh, S. Boyd, et al. "Proximal algorithms". In: *Foundations and Trends in Optimization* 1.3 (2014), pp. 127–239.

[PN18] A. Patrascu and I. Necoara. "Nonasymptotic convergence of stochastic proximal point methods for constrained convex optimization". In: *JMLR* 18.198 (2018), pp. 1–42.

[Pre05] S. J. Press. *Applied multivariate analysis, using Bayesian and frequentist methods of inference*. Second edition. Dover, 2005.

[Pri12] S. Prince. *Computer Vision: Models, Learning and Inference*. Cambridge, 2012.

[PSW15] N. G. Polson, J. G. Scott, and B. T. Willard. "Proximal Algorithms in Statistics and Machine Learning". en. In: *Stat. Sci.* 30.4 (Nov. 2015), pp. 559–581.

[Red+16] S. J. Reddi, S. Sra, B. Póczos, and A. J. Smola. "Proximal Stochastic Methods for Nonsmooth Nonconvex Finite-sum Optimization". In: *NIPS*. NIPS'16. USA, 2016, pp. 1153–1161.

[Rei+10] J. Reisinger, A. Waters, B. Silverthorn, and R. Mooney. "Spherical topic models". In: *ICML*. 2010.

[Rei13] S. Reich. "A Nonparametric Ensemble Transform Method for Bayesian Inference". In: *SIAM J. Sci. Comput.* 35.4 (Jan. 2013), A2013–A2024.

[RH10] M. Ranzato and G. Hinton. "Modeling pixel means and covariances using factored third-order Boltzmann machines". In: *CVPR*. 2010.

[RHG16] M. Roth, G. Hendeby, and F. Gustafsson. "Nonlinear Kalman Filters Explained: A Tutorial on Moment Computations and Sigma Point Methods". In: *J. Advanced Information Fusion* (2016).

[RM15] G. Raskutti and S. Mukherjee. "The information geometry of mirror descent". In: *IEEE Trans. Info. Theory* 61.3 (2015), pp. 1451–1457.

[Rod14] J. Rodu. "Spectral estimation of hidden Markov models". PhD thesis. U. Penn, 2014.

[RÖG13] M. Roth, E. Özkan, and F. Gustafsson. "A Student's t filter for heavy tailed process and measurement noise". In: *ICASSP*. May 2013, pp. 5770–5774.

[Rot+17] M. Roth, G. Hendeby, C. Fritsche, and F. Gustafsson. "The Ensemble Kalman filter: a signal processing perspective". In: *EURASIP J. Adv. Signal Processing* 2017.1 (Aug. 2017), p. 56.

[RR01] A. Rao and K. Rose. "Deterministically Annealed Design of Hidden Markov Model Speech Recognizers". In: *IEEE Trans. on Speech and Audio Proc.* 9.2 (2001), pp. 111–126.

[RSG17] S. Rabanser, O. Shchur, and S. Günnemann. "Introduction to Tensor Decompositions and their Applications in Machine Learning". In: (Nov. 2017). arXiv: 1711.10781 [stat.ML].

[RU10] A. Rajaraman and J. Ullman. *Mining of massive datasets*. Self-published, 2010.

[Rus22] S. Rush. *Illustrated S4*. Tech. rep. 2022.

[Sar13] S. Sarkka. *Bayesian Filtering and Smoothing*. Cambridge University Press, 2013.

[SBG07] S. Siddiqi, B. Boots, and G. Gordon. "A constraint generation approach to learning stable linear dynamical systems". In: *NIPS*. 2007.

[SCC17] S. Sun, C. Chen, and L. Carin. "Learning Structured Weight Uncertainty in Bayesian Neural Networks". In: *AISTATS*. 2017, pp. 1283–1292.

[Sch10] N. Schraudolph. "Polynomial-Time Exact Inference in NP-Hard Binary MRFs via Reweighted Perfect Matching". In: *AISTATS*. 2010.

[Sch+17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. "Proximal Policy Optimization Algorithms". In: (July 2017). arXiv: 1707.06347 [cs.LG].

[Sco02] S Scott. "Bayesian methods for hidden Markov models: Recursive computing in the 21st century." In: *JASA* (2002).

[Seg11] D. Segal. "The dirty little secrets of search". In: *New York Times* (2011).

[SF08] E. Sudderth and W. Freeman. "Signal and Image Processing with Belief Propagation". In: *IEEE Signal Processing Magazine* (2008).

[SF19] A. Shekhovtsov and B. Flach. "Feed-forward Propagation in Probabilistic Neural Networks with Categorical and Max Layers". In: *ICLR*. 2019.

[SG07] M. Steyvers and T. Griffiths. "Probabilistic topic models". In: *Latent Semantic Analysis: A Road to Meaning*. Ed. by T. Landauer, D McNamara, S. Dennis, and W. Kintsch. Laurence Erlbaum, 2007.

[SH03] A. Siepel and D. Haussler. "Combining phylogenetic and hid- den Markov models in biosequence analysis". In: *Proc. 7th Intl. Conf. on Computational Molecular Biology (RECOMB)*. 2003.

[SH10] R. Salakhutdinov and G. Hinton. "Replicated Softmax: an Undirected Topic Model". In: *NIPS*. 2010.

[SHJ97] P. Smyth, D. Heckerman, and M. I. Jordan. "Probabilistic Independence Networks for Hidden Markov Probability Models". In: *Neural Computation* 9.2 (1997), pp. 227–269.

[Shw+91] M. Shwe, B. Middleton, D. Heckerman, M. Henrion, E. Horvitz, H. Lehmann, and G. Cooper. "Probabilistic Diagnosis Using a Reformulation of the INTERNIST-1/QMR Knowledge Base". In: *Methods. Inf. Med* 30.4 (1991), pp. 241–255.

[SJR04] S. Singh, M. James, and M. Rudary. "Predictive state representations: A new theory for modeling dynamical systems". In: *UAI*. 2004.

[SKZ19] J. Shi, M. E. Khan, and J. Zhu. "Scalable Training of Inference Networks for Gaussian-Process Models". In: *ICML*. Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 5758–5768.

[SMH07] R. R. Salakhutdinov, A. Mnih, and G. E. Hinton. "Restricted Boltzmann machines for collaborative filtering". In: *ICML*. Vol. 24. 2007, pp. 791–798.

[Smi+00] G. Smith, J. F. G. de Freitas, T. Robinson, and M. Niranjan. "Speech Modelling Using Subspace and EM Techniques". In: *NIPS*. MIT Press, 2000, pp. 796–802.

[SS02] D. Scharstein and R. Szeliski. "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms". In: *Intl. J. Computer Vision* 47.1 (2002), pp. 7–42.

[STR10] Y. Saatchi, R. Turner, and C. E. Rasmussen. "Gaussian Process Change Point Models". In: *ICML*. unknown, Aug. 2010, pp. 927–934.

[Str15] S. H. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering, Second Edition*. CRC Press, 2015.

[SWW08] E. Sudderth, M. Wainwright, and A. Willsky. "Loop series and Bethe variational bounds for attractive graphical models". In: *NIPS*. 2008.

[Sze+08] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother. "A Comparative Study of Energy Minimization Methods for Markov Random Fields with Smoothness-Based Priors". In: *IEEE PAMI* 30.6 (2008), pp. 1068–1080.

[Sze10] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010.

[Tad15] M. Taddy. "Distributed multinomial regression". en. In: *Annals of Applied Statistics* 9.3 (Sept. 2015), pp. 1394–1414.

[TBF06] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2006.

[TBS13] R. Turner, S. Bottone, and C. Stanek. "Online variational approximations to non-exponential family change point models: With application to radar tracking". In: *NIPS*. 2013.

[Ten+11] J. Tenenbaum, C. Kemp, T. Griffiths, and N. Goodman. "How to Grow a Mind: Statistics, Structure, and Abstraction". In: *Science* 6022 (2011), pp. 1279–1285.

[Ten99] J. Tenenbaum. "A Bayesian framework for concept learning". PhD thesis. MIT, 1999.

[TF03] M. Tappen and B. Freeman. "Comparison of graph cuts with belief propagation for stereo, using identical MRF parameters". In: *ICCV*. Oct. 2003, 900–906 vol.2.

[TMK04] G. Theocharous, K. Murphy, and L. Kaelbling. "Representing hierarchical POMDPs as DBNs for multi-scale robot localization". In: *ICRA*. 2004.

[TOV18] C. Truong, L. Oudre, and N. Vayatis. "Selective review of offline change point detection methods". In: (Jan. 2018). arXiv: 1801.00718 [cs.CE].

[Tse08] P. Tseng. *On accelerated proximal gradient methods for convex-concave optimization*. Unpublished manuscript. 2008.

[UN98] N. Ueda and R. Nakano. "Deterministic annealing EM algorithm". In: *Neural Networks* 11 (1998), pp. 271–282.

[Ver18] R. Vershynin. *High-Dimensional Probability: An Introduction with Applications in Data Science*. en. 1 edition. Cambridge University Press, Sept. 2018.

[Wer07] T. Werner. "A linear programming approach to the max-sum problem: A review". In: *IEEE PAMI* 29.7 (2007), pp. 1165–1179.

[WH97] M. West and J. Harrison. *Bayesian forecasting and dynamic models*. Springer, 1997.

[WHT19] Y. Wang, H. He, and X. Tan. "Truly Proximal Policy Optimization". In: *UAI*. 2019.

[WJ08] M. J. Wainwright and M. I. Jordan. "Graphical models, exponential families, and variational inference". In: *Foundations and Trends in Machine Learning* 1–2 (2008), pp. 1–305.

[WJW05a] M. Wainwright, T. Jaakkola, and A. Willsky. "A new class of upper bounds on the log partition function". In: *IEEE Trans. Info. Theory* 51.7 (2005), pp. 2313–2335.

[WJW05b] M. Wainwright, T. Jaakkola, and A. Willsky. "MAP estimation via agreement on trees: message-passing and linear programming". In: *IEEE Trans. Info. Theory* 51.11 (2005), pp. 3697–3717.

[WM01] E. A. Wan and R. V. der Merwe. "The Unscented Kalman Filter". In: *Kalman Filtering and Neural Networks*. Ed. by S. Haykin. Wiley, 2001.

[WM12] K. Wakabayashi and T. Miura. "Forward-Backward Activation Algorithm for Hierarchical Hidden Markov Models". In: *NIPS*. 2012.

[Wu+06] Y Wu, D Hu, M Wu, and X Hu. "A Numerical-Integration Perspective on Gaussian Filters". In: *IEEE Trans. Signal Process.* 54.8 (Aug. 2006), pp. 2910–2921.

[Wüt+16] M. Wüthrich, S. Trimpe, C. Garcia Cifuentes, D. Kappler, and S. Schaal. "A new perspective and extension of the Gaussian Filter". en. In: *The International Journal of Robotics Research* 35.14 (Dec. 2016), pp. 1731–1749.

[XAH19] Z. Xu, T. Ajanthan, and R. Hartley. "Fast and Differentiable Message Passing for Stereo Vision". In: (Oct. 2019). arXiv: 1910.10892 [cs.CV].

[XT07] F. Xu and J. Tenenbaum. "Word learning as Bayesian inference". In: *Psychological Review* 114.2 (2007).

[Yam+12] K. Yamaguchi, T. Hazan, D. McAllester, and R. Urtasun. "Continuous Markov Random Fields for Robust Stereo Estimation". In: (Apr. 2012). arXiv: 1204.1393 [cs.CV].

[Yao+20] Q. Yao, J. Xu, W.-W. Tu, and Z. Zhu. "Efficient Neural Architecture Search via Proximal Iterations". In: *AAAI*. 2020.

[YBW15] F. Yang, S. Balakrishnan, and M. J. Wainwright. "Statistical and Computational Guarantees for the Baum-Welch Algorithm". In: (2015). arXiv: 1512.08269 [stat.ML].

[Yu10] S.-Z. Yu. "Hidden Semi-Markov Models". In: *Artificial Intelligence J.* 174.2 (2010).

[ZCC07] G. Zhang, T. Chen, and X. Chen. "Performance Recovery in Digital Implementation of Analogue Systems". In: *SIAM J. Control Optim.* 45.6 (Jan. 2007), pp. 2207–2223.

[Zha+20] A. Zhang, Z. Lipton, M. Li, and A. Smola. *Dive into deep learning*. 2020.

[ZK14] W. Zhong and J. Kwok. "Fast Stochastic Alternating Direction Method of Multipliers". In: *ICML*. Ed. by E. P. Xing and T. Jebara. Vol. 32. Proceedings of Machine Learning Research. Bejing, China: PMLR, 2014, pp. 46–54.

[ZM96] G. Zweig and K. Murphy. "The Factored Frontier Algorithm". Unpublished manuscript. 1996.

[ZP00] G. Zweig and M. Padmanabhan. "Exact alpha-beta computation in logarithmic space with application to map word graph construction". In: *ICSLP*. 2000.