Supplementary Material for "Probabilistic Machine Learning: Advanced Topics"

Kevin Murphy

April 26, 2022

Contents

Chapter 1

Introduction

Part I

Fundamentals

Chapter 2

Probability

2.1 More fun with Gaussians

2.1.1 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

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix}, \;\; \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix} \tag{2.1}$$

In **??**, we claimed that

$$p(\boldsymbol{x}_1|\boldsymbol{x}_2) = \mathcal{N}(\boldsymbol{x}_1|\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2), \; \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}) \tag{2.2}$$

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:

$$p(\boldsymbol{x}_1, \boldsymbol{x}_2) \propto \exp\left\{ -\frac{1}{2} \begin{pmatrix} \boldsymbol{x}_1 - \boldsymbol{\mu}_1 \\ \boldsymbol{x}_2 - \boldsymbol{\mu}_2 \end{pmatrix}^\mathsf{T} \begin{pmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix}^{-1} \begin{pmatrix} \boldsymbol{x}_1 - \boldsymbol{\mu}_1 \\ \boldsymbol{x}_2 - \boldsymbol{\mu}_2 \end{pmatrix} \right\} \tag{2.3}$$

Using **??** the above exponent becomes

$$p(\boldsymbol{x}_1, \boldsymbol{x}_2) \propto \exp\left\{ -\frac{1}{2} \begin{pmatrix} \boldsymbol{x}_1 - \boldsymbol{\mu}_1 \\ \boldsymbol{x}_2 - \boldsymbol{\mu}_2 \end{pmatrix}^\mathsf{T} \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ -\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21} & \mathbf{I} \end{pmatrix} \begin{pmatrix} (\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_{22}^{-1} \end{pmatrix} \right. \tag{2.4}$$

$$\left. \times \begin{pmatrix} \mathbf{I} & -\boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_1 - \boldsymbol{\mu}_1 \\ \boldsymbol{x}_2 - \boldsymbol{\mu}_2 \end{pmatrix} \right\} \tag{2.5}$$

$$= \exp\left\{ -\frac{1}{2}(\boldsymbol{x}_1 - \boldsymbol{\mu}_1 - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2))^\mathsf{T}(\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1} \right. \tag{2.6}$$

$$\left. (\boldsymbol{x}_1 - \boldsymbol{\mu}_1 - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2)) \right\} \times \exp\left\{ -\frac{1}{2}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2)^\mathsf{T}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2) \right\} \tag{2.7}$$

This is of the form

$$\exp(\text{quadratic form in } \boldsymbol{x}_1, \boldsymbol{x}_2) \times \exp(\text{quadratic form in } \boldsymbol{x}_2) \tag{2.8}$$

Hence we have successfully factorized the joint as

$$p(\boldsymbol{x}_1, \boldsymbol{x}_2) = p(\boldsymbol{x}_1|\boldsymbol{x}_2)p(\boldsymbol{x}_2) \tag{2.9}$$

$$= \mathcal{N}(\boldsymbol{x}_1|\boldsymbol{\mu}_{1|2}, \boldsymbol{\Sigma}_{1|2})\mathcal{N}(\boldsymbol{x}_2|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{22}) \tag{2.10}$$

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

$$\boldsymbol{\mu}_{1|2} = \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2) \tag{2.11}$$

$$\boldsymbol{\Sigma}_{1|2} = \boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22} = \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21} \tag{2.12}$$

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

$$(2\pi)^{(d_1+d_2)/2}|\boldsymbol{\Sigma}|^{\frac{1}{2}} = (2\pi)^{(d_1+d_2)/2}(|\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22}|\ |\boldsymbol{\Sigma}_{22}|)^{\frac{1}{2}} \tag{2.13}$$

$$= (2\pi)^{d_1/2}|\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22}|^{\frac{1}{2}}\ (2\pi)^{d_2/2}|\boldsymbol{\Sigma}_{22}|^{\frac{1}{2}} \tag{2.14}$$

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

$$\boldsymbol{\Lambda} \triangleq \boldsymbol{\Sigma}^{-1}, \quad \boldsymbol{\eta} \triangleq \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} \tag{2.15}$$

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

$$p(\boldsymbol{x}_2) = \mathcal{N}_c(\boldsymbol{x}_2|\boldsymbol{\eta}_2 - \boldsymbol{\Lambda}_{21}\boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\eta}_1, \boldsymbol{\Lambda}_{22} - \boldsymbol{\Lambda}_{21}\boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12}) \tag{2.16}$$

$$p(\boldsymbol{x}_1|\boldsymbol{x}_2) = \mathcal{N}_c(\boldsymbol{x}_1|\boldsymbol{\eta}_1 - \boldsymbol{\Lambda}_{12}\boldsymbol{x}_2, \boldsymbol{\Lambda}_{11}) \tag{2.17}$$

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

$$\begin{pmatrix}\boldsymbol{\eta}_1 \\ \boldsymbol{\eta}_2\end{pmatrix} = \begin{pmatrix}\boldsymbol{\Lambda}_{11} & \boldsymbol{\Lambda}_{12} \\ \boldsymbol{\Lambda}_{21} & \boldsymbol{\Lambda}_{22}\end{pmatrix}\begin{pmatrix}\boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2\end{pmatrix} \tag{2.18}$$

so

$$\boldsymbol{\eta}_1 = \boldsymbol{\Lambda}_{11}\boldsymbol{\mu}_1 + \boldsymbol{\Lambda}_{12}\boldsymbol{\mu}_2 \tag{2.19}$$

$$\boldsymbol{\eta}_2 = \boldsymbol{\Lambda}_{21}\boldsymbol{\mu}_1 + \boldsymbol{\Lambda}_{22}\boldsymbol{\mu}_2 \tag{2.20}$$

Hence

$$\begin{pmatrix}\boldsymbol{\Lambda}_{11} & \boldsymbol{\Lambda}_{12} \\ \boldsymbol{\Lambda}_{21} & \boldsymbol{\Lambda}_{22}\end{pmatrix} = \begin{pmatrix}\boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22}\end{pmatrix}^{-1} \tag{2.21}$$

$$= \begin{pmatrix}(\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1} & -(\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1}\boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1} \\ -\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}(\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1} & \boldsymbol{\Sigma}_{22}^{-1} + \boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}(\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1}\boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-2}\end{pmatrix} \tag{2.22}$$

where the top left is

$$\boldsymbol{\Lambda}_{11} = (\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1} = (\boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21})^{-1} \tag{2.23}$$

Using the moment form for conditioning we have

$$\boldsymbol{\Lambda}_{1|2} = \boldsymbol{\Sigma}_{1|2}^{-1} = (\boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21})^{-1} = \boldsymbol{\Lambda}_{11} \tag{2.24}$$

$$\boldsymbol{\mu}_{1|2} = \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2) \tag{2.25}$$

but from Equation 2.22 and 2.23 we have

$$\boldsymbol{\Lambda}_{12} = -\boldsymbol{\Lambda}_{11}\boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1} \tag{2.26}$$

so

$$\boldsymbol{\mu}_{1|2} = \boldsymbol{\mu}_1 - \boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2) \tag{2.27}$$

$$\boldsymbol{\eta}_{1|2} = \boldsymbol{\Lambda}_{1|2}\boldsymbol{\mu}_{1|2} = \boldsymbol{\Lambda}_{11}\boldsymbol{\mu}_1 - \boldsymbol{\Lambda}_{12}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2) \tag{2.28}$$

$$= \boldsymbol{\Lambda}_{11}\boldsymbol{\mu}_1 + \boldsymbol{\Lambda}_{12}\boldsymbol{\mu}_2 - \boldsymbol{\Lambda}_{12}\boldsymbol{x}_2 = \boldsymbol{\eta}_1 - \boldsymbol{\Lambda}_{12}\boldsymbol{x}_2 \tag{2.29}$$

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

$$\begin{pmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\Lambda}_{11} & \boldsymbol{\Lambda}_{12} \\ \boldsymbol{\Lambda}_{21} & \boldsymbol{\Lambda}_{22} \end{pmatrix}^{-1} \tag{2.30}$$

$$= \begin{pmatrix} \boldsymbol{\Lambda}_{11}^{-1} + \boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12}(\boldsymbol{\Lambda}/\boldsymbol{\Lambda}_{11})^{-1}\boldsymbol{\Lambda}_{21}\boldsymbol{\Lambda}_{11}^{-1} & -\boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12}(\boldsymbol{\Lambda}/\boldsymbol{\Lambda}_{11})^{-1} \\ -(\boldsymbol{\Lambda}/\boldsymbol{\Lambda}_{11})^{-1}\boldsymbol{\Lambda}_{21}\boldsymbol{\Lambda}_{11}^{-1} & (\boldsymbol{\Lambda}/\boldsymbol{\Lambda}_{11})^{-1} \end{pmatrix} \tag{2.31}$$

Hence

$$\boldsymbol{\Lambda}_{22}^m = \boldsymbol{\Sigma}_{22}^{-1} = \boldsymbol{\Lambda}/\boldsymbol{\Lambda}_{11} = \boldsymbol{\Lambda}_{22} - \boldsymbol{\Lambda}_{21}\boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12} \tag{2.32}$$

$$\boldsymbol{\eta}_2^m = \boldsymbol{\Lambda}_{22}^m\boldsymbol{\mu}_2^m = (\boldsymbol{\Lambda}_{22} - \boldsymbol{\Lambda}_{21}\boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12})\boldsymbol{\mu}_2 \tag{2.33}$$

$$= \boldsymbol{\Lambda}_{22}\boldsymbol{\mu}_2 - \boldsymbol{\Lambda}_{21}\boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12}\boldsymbol{\mu}_2 \tag{2.34}$$

$$= (\boldsymbol{\Lambda}_{21}\boldsymbol{\mu}_1 + \boldsymbol{\Lambda}_{22}\boldsymbol{\mu}_2) - \boldsymbol{\Lambda}_{21}\boldsymbol{\Lambda}_{11}^{-1}(\boldsymbol{\Lambda}_{11}\boldsymbol{\mu}_1 + \boldsymbol{\Lambda}_{12}\boldsymbol{\mu}_2) \tag{2.35}$$

$$= \boldsymbol{\eta}_2 - \boldsymbol{\Lambda}_{21}\boldsymbol{\Lambda}_{11}\boldsymbol{\eta}_1 \tag{2.36}$$

2.1.2 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):

$$\log p(\boldsymbol{z}, \boldsymbol{y}) = -\frac{1}{2}(\boldsymbol{z} - \boldsymbol{\mu}_z)^T\boldsymbol{\Sigma}_z^{-1}(\boldsymbol{z} - \boldsymbol{\mu}_z) - \frac{1}{2}(\boldsymbol{y} - \mathbf{W}\boldsymbol{z} - \boldsymbol{b})^T\boldsymbol{\Sigma}_y^{-1}(\boldsymbol{y} - \mathbf{W}\boldsymbol{z} - \boldsymbol{b}) \tag{2.37}$$

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

$$Q = -\frac{1}{2}\boldsymbol{z}^T\boldsymbol{\Sigma}_z^{-1}\boldsymbol{z} - \frac{1}{2}\boldsymbol{y}^T\boldsymbol{\Sigma}_y^{-1}\boldsymbol{y} - \frac{1}{2}(\mathbf{W}\boldsymbol{z})^T\boldsymbol{\Sigma}_y^{-1}(\mathbf{W}\boldsymbol{z}) + \boldsymbol{y}^T\boldsymbol{\Sigma}_y^{-1}\mathbf{W}\boldsymbol{z} \tag{2.38}$$

$$= -\frac{1}{2}\begin{pmatrix} \boldsymbol{z} \\ \boldsymbol{y} \end{pmatrix}^T \begin{pmatrix} \boldsymbol{\Sigma}_z^{-1} + \mathbf{W}^T\boldsymbol{\Sigma}_y^{-1}\mathbf{W} & -\mathbf{W}^T\boldsymbol{\Sigma}_y^{-1} \\ -\boldsymbol{\Sigma}_y^{-1}\mathbf{W} & \boldsymbol{\Sigma}_y^{-1} \end{pmatrix} \begin{pmatrix} \boldsymbol{z} \\ \boldsymbol{y} \end{pmatrix} \tag{2.39}$$

$$= -\frac{1}{2}\begin{pmatrix} \boldsymbol{z} \\ \boldsymbol{y} \end{pmatrix}^T \boldsymbol{\Sigma}^{-1} \begin{pmatrix} \boldsymbol{z} \\ \boldsymbol{y} \end{pmatrix} \tag{2.40}$$

where the precision matrix of the joint is defined as

$$\boldsymbol{\Sigma}^{-1} = \begin{pmatrix} \boldsymbol{\Sigma}_z^{-1} + \mathbf{W}^T\boldsymbol{\Sigma}_y^{-1}\mathbf{W} & -\mathbf{W}^T\boldsymbol{\Sigma}_y^{-1} \\ -\boldsymbol{\Sigma}_y^{-1}\mathbf{W} & \boldsymbol{\Sigma}_y^{-1} \end{pmatrix} \triangleq \boldsymbol{\Lambda} = \begin{pmatrix} \boldsymbol{\Lambda}_{xx} & \boldsymbol{\Lambda}_{xy} \\ \boldsymbol{\Lambda}_{yx} & \boldsymbol{\Lambda}_{yy} \end{pmatrix} \tag{2.41}$$

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

$$p(\boldsymbol{z}|\boldsymbol{y}) = \mathcal{N}(\boldsymbol{\mu}_{z|y}, \boldsymbol{\Sigma}_{z|y}) \tag{2.42}$$

$$\boldsymbol{\Sigma}_{z|y} = \boldsymbol{\Lambda}_{xx}^{-1} = (\boldsymbol{\Sigma}_z^{-1} + \mathbf{W}^T\boldsymbol{\Sigma}_y^{-1}\mathbf{W})^{-1} \tag{2.43}$$

$$\boldsymbol{\mu}_{z|y} = \boldsymbol{\Sigma}_{z|y}\left(\boldsymbol{\Lambda}_{xx}\boldsymbol{\mu}_z - \boldsymbol{\Lambda}_{xy}(\boldsymbol{y} - \boldsymbol{\mu}_y)\right) \tag{2.44}$$

$$= \boldsymbol{\Sigma}_{z|y}\left(\boldsymbol{\Sigma}_z^{-1}\boldsymbol{\mu}_z + \mathbf{W}^\mathsf{T}\boldsymbol{\Sigma}_y^{-1}\mathbf{W}\boldsymbol{\mu}_z + \mathbf{W}^T\boldsymbol{\Sigma}_y^{-1}(\boldsymbol{y} - \boldsymbol{\mu}_y)\right) \tag{2.45}$$

$$= \boldsymbol{\Sigma}_{z|y}\left(\boldsymbol{\Sigma}_z^{-1}\boldsymbol{\mu}_z + \mathbf{W}^\mathsf{T}\boldsymbol{\Sigma}_y^{-1}(\mathbf{W}\boldsymbol{\mu}_z + \boldsymbol{y} - \boldsymbol{\mu}_y)\right) \tag{2.46}$$

$$= \boldsymbol{\Sigma}_{z|y}\left(\boldsymbol{\Sigma}_z^{-1}\boldsymbol{\mu}_z + \mathbf{W}^T\boldsymbol{\Sigma}_y^{-1}(\boldsymbol{y} - \boldsymbol{b})\right) \tag{2.47}$$

2.1.3 Sensor fusion with unknown measurement noise

In this section, we extend the sensor fusion results from **??** 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

$$x_1 = 1.1, x_2 = 1.9, y_1 = 2.9, y_2 = 4.1 \tag{2.48}$$

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

$$p(z|\mathcal{D}, \lambda_x, \lambda_y) = \mathcal{N}(z|m_N, \lambda_N^{-1}) \tag{2.49}$$

$$\lambda_N = \lambda_0 + N_x\lambda_x + N_y\lambda_y \tag{2.50}$$

$$m_N = \frac{\lambda_x N_x \overline{x} + \lambda_y N_y \overline{y}}{N_x\lambda_x + N_y\lambda_y} \tag{2.51}$$

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

$$\ell(z, \lambda_x, \lambda_y) = \frac{N_x}{2}\log\lambda_x - \frac{\lambda_x}{2}\sum_n (x_n - z)^2 + \frac{N_y}{2}\log\lambda_y - \frac{\lambda_y}{2}\sum_n (y_n - z)^2 \tag{2.52}$$

The MLE is obtained by solving the following simultaneous equations:

$$\frac{\partial \ell}{\partial z} = \lambda_x N_x(\overline{x} - z) + \lambda_y N_y(\overline{y} - z) = 0 \tag{2.53}$$

$$\frac{\partial \ell}{\partial \lambda_x} = \frac{1}{\lambda_x} - \frac{1}{N_x}\sum_{n=1}^{N_x}(x_n - z)^2 = 0 \tag{2.54}$$

$$\frac{\partial \ell}{\partial \lambda_y} = \frac{1}{\lambda_y} - \frac{1}{N_y}\sum_{n=1}^{N_y}(y_n - z)^2 = 0 \tag{2.55}$$

This gives

$$\hat{z} = \frac{N_x\hat{\lambda}_x\overline{x} + N_y\hat{\lambda}_y\overline{y}}{N_x\hat{\lambda}_x + N_y\hat{\lambda}_y} \tag{2.56}$$

$$1/\hat{\lambda}_x = \frac{1}{N_x}\sum_n (x_n - \hat{z})^2 \tag{2.57}$$

$$1/\hat{\lambda}_y = \frac{1}{N_y}\sum_n (y_n - \hat{z})^2 \tag{2.58}$$

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 2.1(a). This weights each sensor according to its estimated precision. Since sensor y was estimated to be much less reliable than sensor x, we have $\mathbb{E}\left[z|\mathcal{D}, \hat{\lambda}_x, \hat{\lambda}_y\right] \approx \overline{x}$, 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

$$p(z|\mathcal{D}) \propto p(z)\left[\int p(\mathcal{D}_x|z,\lambda_x)p(\lambda_x|z)d\lambda_x\right]\left[\int p(\mathcal{D}_y|z,\lambda_y)p(\lambda_y|z)d\lambda_y\right] \tag{2.59}$$

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

$$I = \int p(\mathcal{D}_x|z,\lambda_x)p(\lambda_x|z)d\lambda_x \tag{2.60}$$

$$\propto \int \lambda_x^{-1}\lambda_x^{N_x/2}\exp\left(-\frac{N_x}{2}\lambda_x(\overline{x}-z)^2 - \frac{N_x}{2}s_x^2\lambda_x\right)d\lambda_x \tag{2.61}$$

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

$$I = \int \lambda_x^{-1}\lambda_x^1\exp(-\lambda_x[(\overline{x}-z)^2 + s_x^2])d\lambda_x \tag{2.62}$$

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

$$\mathrm{Ga}(\lambda|a,b) \propto \lambda^{a-1}e^{-\lambda b} \tag{2.63}$$

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

$$I \propto \int p(\mathcal{D}_x|z,\lambda_x)p(\lambda_x|z)d\lambda_x \propto \left((\overline{x}-z)^2 + s_x^2\right)^{-1} \tag{2.64}$$

and the posterior becomes

$$p(z|\mathcal{D}) \propto \frac{1}{(\overline{x}-z)^2 + s_x^2}\ \frac{1}{(\overline{y}-z)^2 + s_y^2} \tag{2.65}$$

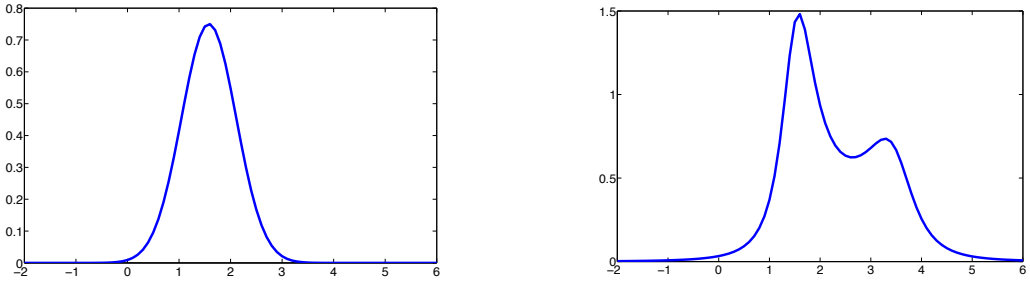
The exact posterior is plotted in Figure 2.1(b). We see that it has two modes, one near $\overline{x} = 1.5$ and one near $\overline{y} = 3.5$. These correspond to the beliefs that the x sensor is more reliable than the y one, and vice versa. The weight of the first mode is larger, since the data from 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:

$$p(\boldsymbol{z}|\mathcal{D}) \propto \int p(\boldsymbol{z}|\mathcal{D},\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta} \tag{2.66}$$

$$\approx \frac{1}{S}\sum_s p(\boldsymbol{z}|\mathcal{D},\boldsymbol{\theta}^s) \tag{2.67}$$

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

2.2 Google's PageRank algorithm

In this section, we discuss Google's **PageRank** algorithm, since it provides an interesting application of Markov chain theory. PageRanke is one of the components used for ranking web page search results. We sketch the basic idea below; see [BL06b] for a more detailed explanation.

2.2.1 Retrieving relevant pages using inverted indices

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

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

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

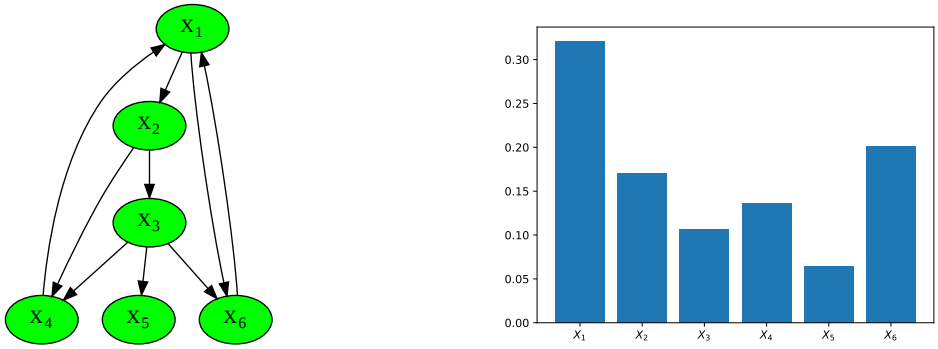
```
"a":      {(2, 2)}
"banana": {(2, 3)}
"is":     {(0, 1), (0, 4), (1, 1), (2, 1)}
"it":     {(0, 0), (0, 3), (1, 2), (2, 0)}
"what":   {(0, 2), (1, 0)}
```

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:

$$\{D_0, D_1\} \cap \{D_0, D_1, D_2\} \cap \{D_0, D_1, D_2\} = \{D_0, D_1\} \tag{2.68}$$

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

[1] 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 2.2: (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.

2.2.2 The PageRank score

So far, we have described the standard process of information retrieval. But the link structure of the web provides an additional source of information. The basic idea is that some web pages are more authoritative than others, so these should be ranked higher (assuming they match the query). A web page is 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**:

$$\pi_j = \sum_i A_{ij} \pi_i \tag{2.69}$$

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

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

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

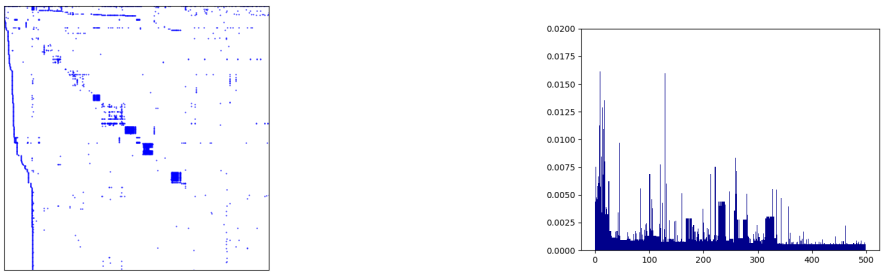
$$\boldsymbol{\pi} = (0.3209, 0.1706, 0.1065, 0.1368, 0.0643, 0.2008) \tag{2.70}$$

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

2.2.3 Efficiently computing the PageRank vector

Let $G_{ij} = 1$ iff there is a link from j to i. Now imagine performing a random walk on this graph, where at every time step, with probability p you follow one of the outlinks uniformly at random, and with probability



(a) (b)

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

$1-p$ 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:

$$M_{ij} = \begin{cases} pG_{ij}/c_j + \delta & \text{if } c_j \neq 0 \\ 1/n & \text{if } c_j = 0 \end{cases} \tag{2.71}$$

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

$$d_{jj} = \begin{cases} 1/c_j & \text{if } c_j \neq 0 \\ 0 & \text{if } c_j = 0 \end{cases} \tag{2.72}$$

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

$$z_j = \begin{cases} \delta & \text{if } c_j \neq 0 \\ 1/n & \text{if } c_j = 0 \end{cases} \tag{2.73}$$

Then we can rewrite Equation (2.71) as follows:

$$\mathbf{M} = p\mathbf{G}\mathbf{D} + \mathbf{1}\boldsymbol{z}^\mathsf{T} \tag{2.74}$$

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:

$$\boldsymbol{v} \propto \mathbf{M}\boldsymbol{v} = p\mathbf{G}\mathbf{D}\boldsymbol{v} + \mathbf{1}\boldsymbol{z}^\mathsf{T}\boldsymbol{v} \tag{2.75}$$

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 2.3(a) yields the stationary distribution in Figure 2.3(b). This took 13 iterations to converge, starting from a uniform distribution. To handle changing web structure, we can re-run this algorithm every day or every week, starting $\boldsymbol{v}$ 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].

2.2.4 Web spam

PageRank is not foolproof. For example, consider the strategy adopted by JC Penney, a department store in the USA. During the Christmas season of 2010, it planted many links to its home page on 1000s of irrelevant web pages, thus increasing its ranking on Google's search engine [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.

2.2.5 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

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

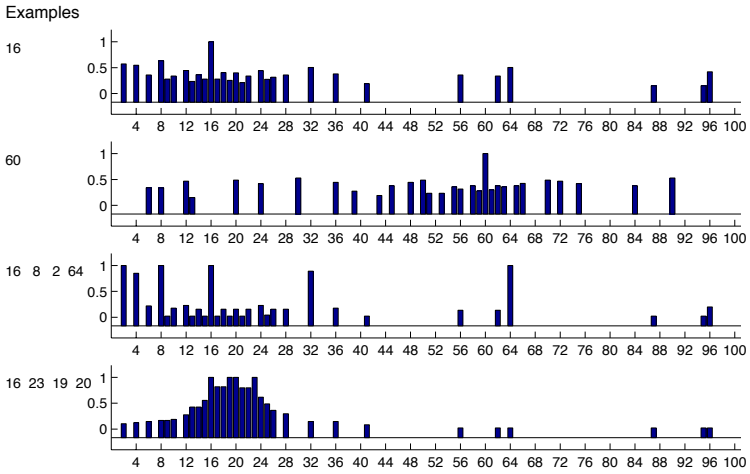


Figure 3.1: Empirical membership distribution in the numbers game, derived from predictions from 8 humans. First two rows: after seeing $\mathcal{D} = \{16\}$ and $\mathcal{D} = \{60\}$. This illustrates diffuse similarity. Third row: after seeing $\mathcal{D} = \{16, 8, 2, 64\}$. This illustrates rule-like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

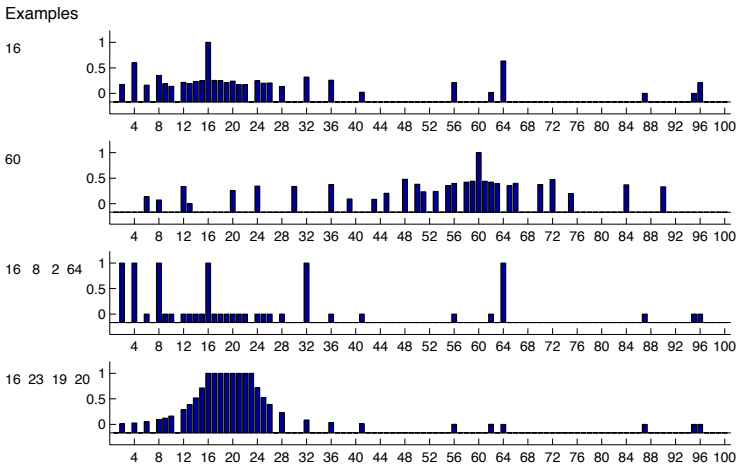


Figure 3.2: Posterior membership probabilities derived using the full hypothesis space. Compare to Figure 3.1. The predictions of the Bayesian model are only plotted for those values for which human data is available; this is why the top line looks sparser than Figure 3.4. From Figure 5.6 of [Ten99].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

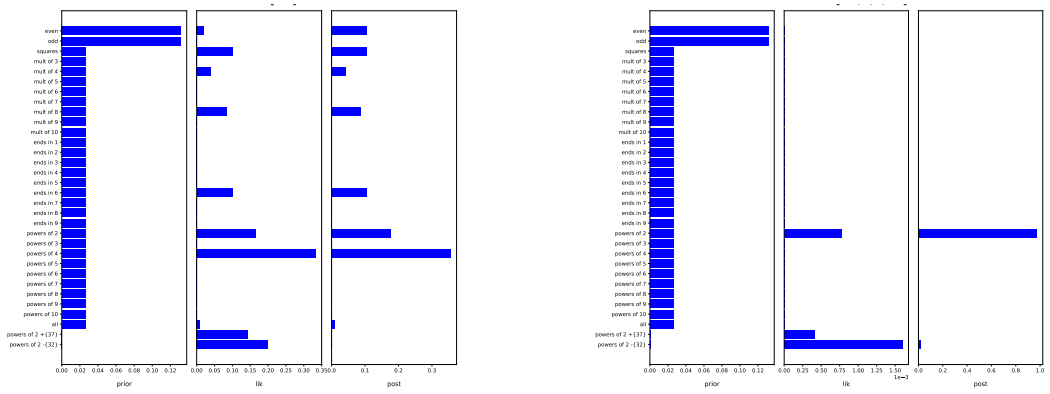
3.1.1.2 Prior

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.

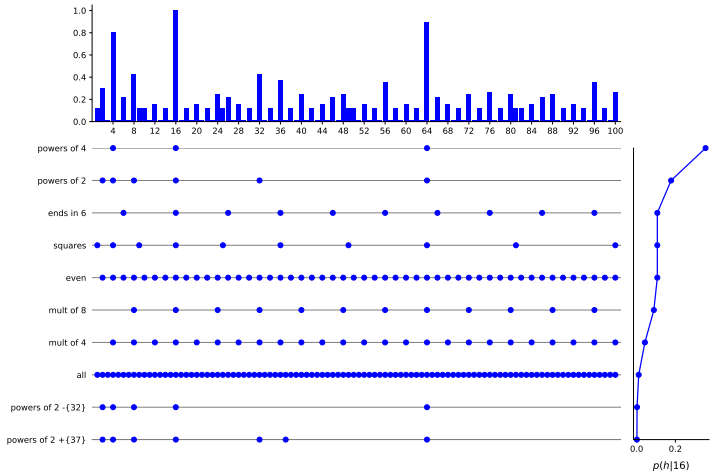


Figure 3.4: Posterior over hypotheses, and the induced posterior over membership, after seeing one example, $\mathcal{D} = \{16\}$. A dot means this number is consistent with this hypothesis. The graph $p(h|\mathcal{D})$ on the right is the weight given to hypothesis h. By taking a weighed sum of dots, 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.

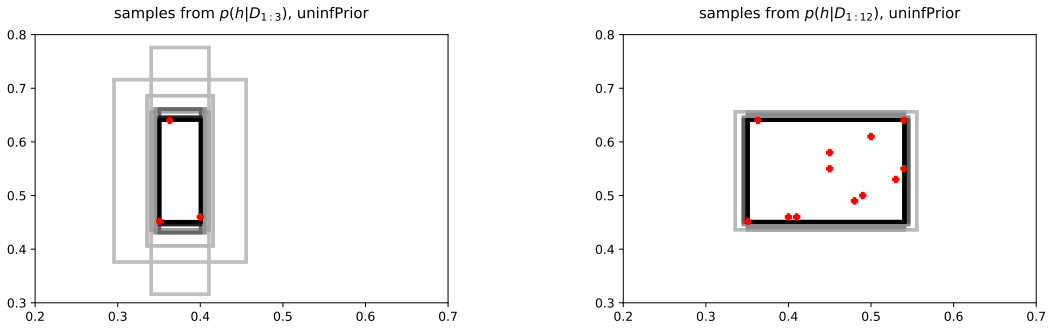
We illustrate these ideas by considering another example of concept learning called the **healthy levels game**, also due to Tenenbaum. The idea is this: we measure two continuous variables, representing the cholesterol and insulin levels of some randomly chosen healthy patients. We would like to know 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 $\boldsymbol{y} \in \mathbb{R}^2$.

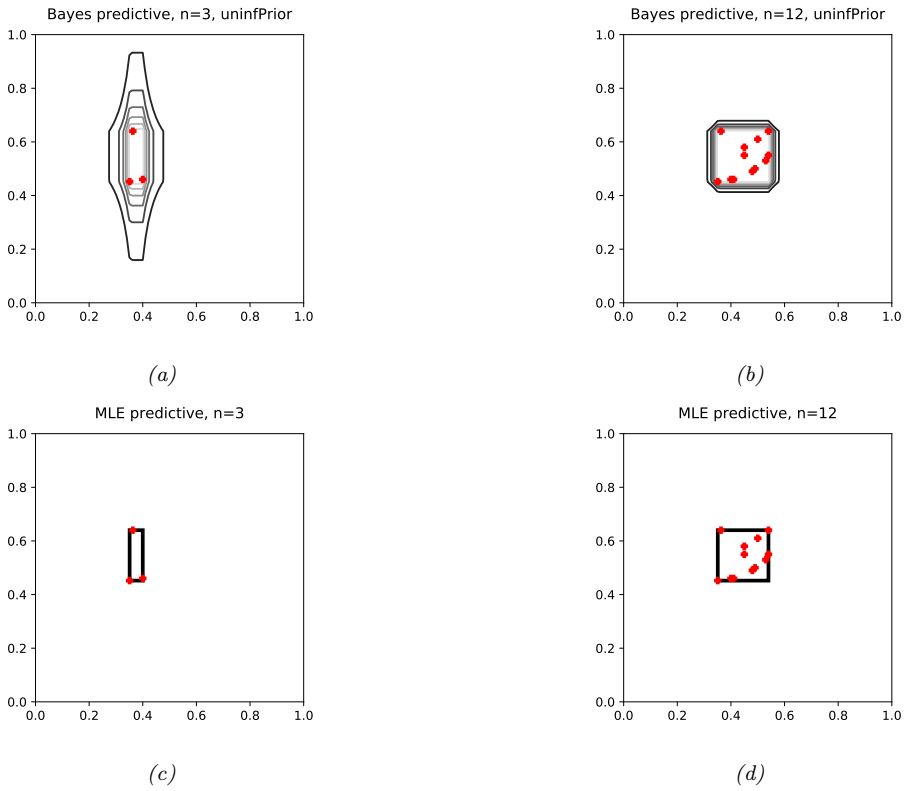


Figure 3.6: Posterior predictive distribution for the healthy levels game. Red crosses are observed data points. Left column: $N = 3$. Right column: $N = 12$. First row: Bayesian prediction. Second row: Plug-in prediction using MLE (smallest enclosing rectangle). We see that the Bayesian prediction goes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

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 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:

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

where

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

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

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

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

3.2.1 Domain specific priors

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

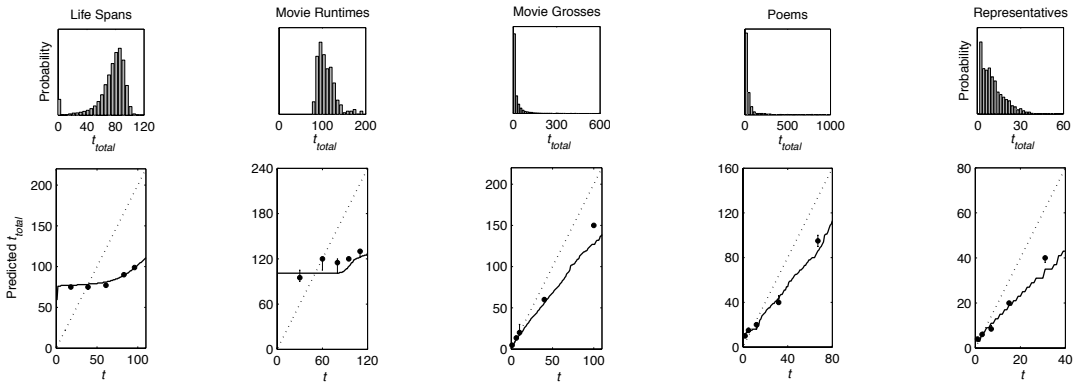


Figure 3.7: Top: empirical distribution of various durational quantities. Bottom: predicted total duration as a function of observed duration, $p(T|t)$. Dots are observed median responses of people. Solid line: Bayesian prediction using informed prior. Dotted line: Bayesian prediction using uninformative prior. From Figure 2a of [GT06]. Used with kind permission of Tom Griffiths.

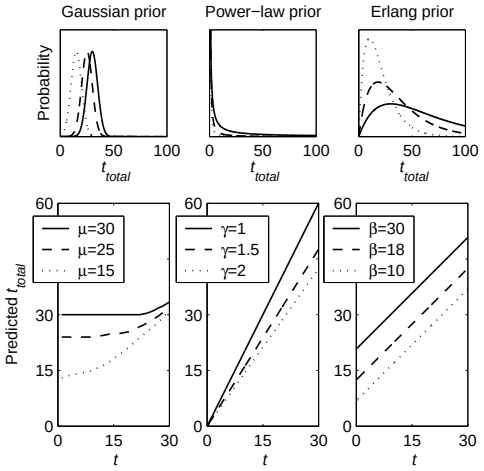


Figure 3.8: Top: three different prior distributions, for three different parameter values. Bottom: corresponding predictive distributions. From Figure 1 of [GT06]. Used with kind permission of Tom Griffiths.

3.2.2 Gaussian prior

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

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

3.2.3 Power-law prior

Looking at Figure 3.7(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 **??**.) Power-laws are characterized by having very **long tails**. This captures the fact that most 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

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

Hence the posterior becomes

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

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

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

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 3.8 (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.

3.2.4 Erlang prior

Looking at Figure 3.7(e), it seems clear that the number of years a US Representative is approximately modeled by a gamma distribution (**??**). 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**:

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

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

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

so the posterior has the form

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

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

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

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

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

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.

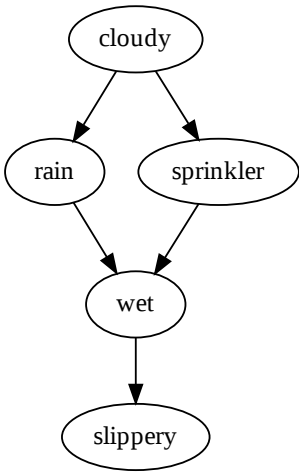
In addition to the graph structure, we need to specify the CPDs. For discrete random variables, we can represent the CPD as a table, which means we have a separate row (i.e., a separate categorical distribution) for each **conditioning case**, i.e., for each combination of parent values. This is known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

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

Thus $\boldsymbol{\theta}_i$ is a **row stochastic matrix**, that satisfies the properties $0 \le \theta_{ijk} \le 1$ 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 $J_i = \prod_{p \in \mathrm{pa}(i)} K_p$. 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,

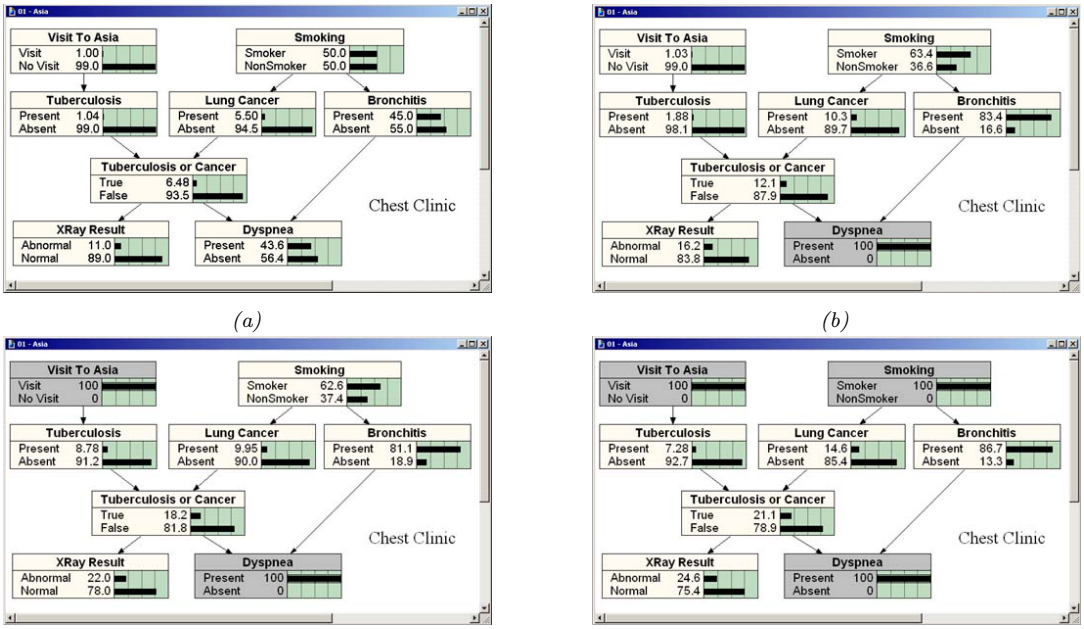


Figure 4.2: Illustration of belief updating in the "Asia" PGM. The histograms show the marginal distribution of each node. (a) Prior. (b) Posterior after conditioning on Dyspnea=Present. (c) Posterior after also conditioning on VisitToAsia=True. (d) Posterior after also conditioning on Smoking=True.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

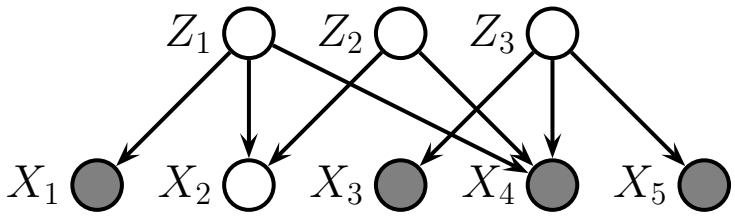


Figure 4.3: A small version of the QMR network. All nodes are binary. The hidden nodes z_k represent diseases, and the visible nodes x_d represent symptoms. In the full network, there are 570 hidden (disease) nodes and 4075 visible (symptom) nodes. The shaded (solid gray) leaf nodes are observed; in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

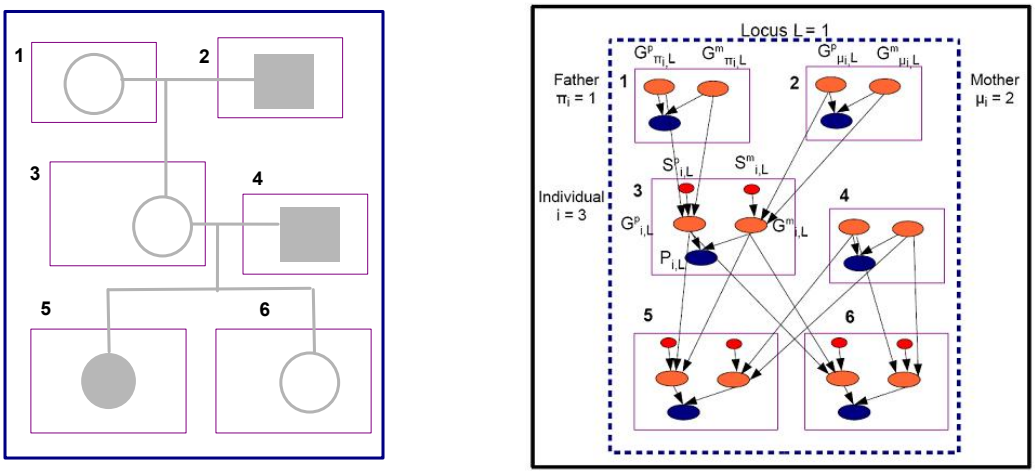
$$p(x_d = 1|\boldsymbol{z}) = 1 - \exp\left(w_{0d} + \sum_k z_k w_{kd}\right) \tag{4.6}$$

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

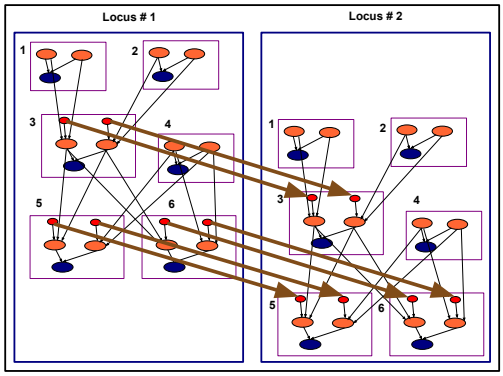


Figure 4.5: Extension of Figure 4.4 to two loci, showing how the switching variables are spatially correlated. This is indicated by the $S^m_{ij} \to S^m_{i,j+1}$ and $S^p_{ij} \to S^p_{i,j+1}$ edges. Adapted from Figure 3 from [FGL00].

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

4.2.0.1 Potts models for protein structure prediction

One interesting application of Potts models arises in the area of **protein structure prediction**. The goal is to predict the 3d shape of a protein from its 1d sequence of amino acids. A common approach to this is known as **direct coupling analysis** (DCA). We give a brief summary below; for details, see [Mor+11].

First we compute a **multiple sequence alignment** (MSA), from a set of related amino acid sequences from the same protein family; this can be done using HMMs, as explained in **??**. The MSA can be represented by an $N \times T$ matrix $\mathbf{X}$, where N is the number of sequences, T is the length of each sequence, and $X_{ni} \in \{1, \ldots, V\}$ is the identity of the letter at location i in sequence n. For protein sequences, $V = 21$, representing the 20 amino acids plus the gap character.

Once we have the MSA matrix $\mathbf{X}$, we fit the Potts model using maximum likelihood estimation, or some approximation, such as pseudo likelihood [Eke+13]; see **??** for details.[1] After fitting the model, we select the edges with the highest J_{ij} coefficients, where $i, j \in \{1, \ldots, T\}$ are locations or **residues** in the protein. Since these locations are highly coupled, they are likely to be in physical contact, since interacting residues must coevolve to avoid destroying the function of the protein (see e.g., [LHF17] for a review). This graph is called a **contact map**.

Once the contact map is established, it can be used as input to a 3d structural prediction algorithm, such as [Xu18] or the **alphafold** system [Eva+18], which won the 2018 CASP competition. Such methods use neural networks to learn functions of the form $p(d(i,j)|\{c(i,j)\})$, where $d(i,j)$ is the 3d distance between residues i and j, and $c(i,j)$ is the contact map.

4.2.1 Hopfield networks

A **Hopfield network** [Hop82] is a fully connected Ising model (**??**) with a symmetric weight matrix, $\mathbf{W} = \mathbf{W}^\mathsf{T}$. The corresponding energy function has the form

$$\mathcal{E}(\boldsymbol{x}) = -\frac{1}{2}\boldsymbol{x}^\mathsf{T}\mathbf{W}\boldsymbol{x} \tag{4.8}$$

where $x_i \in \{-1, +1\}$.

The main application of Hopfield networks is as an **associative memory** or **content addressable memory**. The idea is this: suppose we train on a set of fully observed bit vectors, corresponding to patterns we want to memorize. (We discuss how to do this below). Then, at test time, we present a partial pattern to the network. We would like to estimate the missing variables; this is called **pattern completion**. That is, we want to compute

$$\boldsymbol{x}^* = \operatorname*{argmin}_{\boldsymbol{x}} \mathcal{E}(\boldsymbol{x}) \tag{4.9}$$

We can solve this optimization problem using **iterative conditional modes** (**ICM**), in which we set each hidden variable to its most likely state given its neighbors. Picking the most probable state amounts to using the rule

$$\boldsymbol{x}^{t+1} = \text{sgn}(\mathbf{W}\boldsymbol{x}^t) \tag{4.10}$$

This can be seen as a deterministic version of Gibbs sampling (see **??**).

[1] To encourage the model to learn sparse connectivity, we can also compute a MAP estimate with a sparsity promoting prior, as discussed in [IM17].

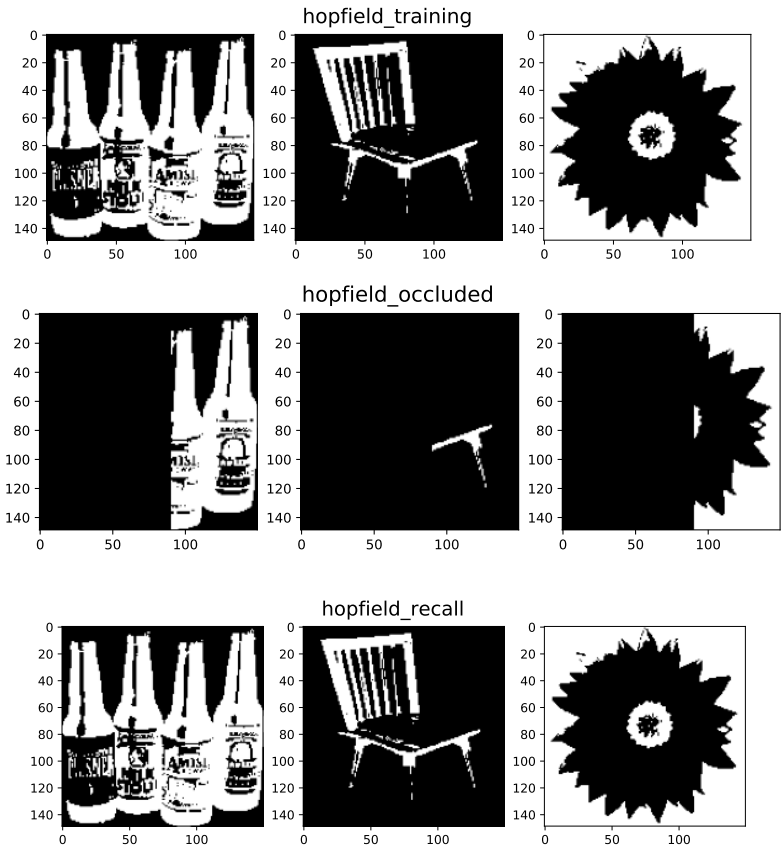


Figure 4.6: Examples of how an associative memory can reconstruct images. These are binary images of size 150×150 *pixels. Top: training images. Middle row: partially visible test images. Bottom row: final state estimate. Adapted from Figure 2.1 of [HKP91]. Generated by hopfield_ demo.py.*

We illustrate this process in Figure 4.6. In the top row, we show some training examples. In the middle row, we show a corrupted input, corresponding to the initial state $\boldsymbol{x}^0$. In the bottom row, we show the final state after 30 iterations of ICM. The overall process can be thought of as retrieving a complete example from memory based on a piece of the example.

To learn the weights $\mathbf{W}$, we could use the maximum likelihood estimate method described in **??**. (See also [HSDK12].) However, a simpler heuristic method, proposed in [Hop82], is to use the following **outer product method**:

$$\mathbf{W} = \left(\frac{1}{N} \sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^{\mathsf{T}} \right) - \mathbf{I} \tag{4.11}$$

This normalizes the output product matrix by N, and then sets the diagonal to 0. This ensures the energy is low for patterns that match any of the examples in the training set. This is the technique we used in Figure 4.6. Note, however, that this method not only stores the original patterms but also their inverses, and other linear combinations. Consequently there is a limit to how many examples the model can store before they start to "collide" in the memory. Hopfield proved that, for random patterns, the network capacity is $\sim 0.14N$.

4.2.2 Restricted Boltzmann machines (RBMs) in more detail

In this section, we discuss RBMs in more detail.

4.2.2.1 Binary RBMs

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

$$p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp(-\mathcal{E}(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{\theta})) \tag{4.12}$$

$$\mathcal{E}(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{\theta}) \triangleq -\sum_{d=1}^{D}\sum_{k=1}^{K} x_d z_k W_{dk} - \sum_{d=1}^{D} x_d b_d - \sum_{k=1}^{K} z_k c_k \tag{4.13}$$

$$= -(\boldsymbol{x}^\mathsf{T}\mathbf{W}\boldsymbol{z} + \boldsymbol{x}^\mathsf{T}\boldsymbol{b} + \boldsymbol{z}^\mathsf{T}\boldsymbol{c}) \tag{4.14}$$

$$Z(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}}\sum_{\boldsymbol{z}} \exp(-E(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{\theta})) \tag{4.15}$$

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:

$$p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{\theta}) = \prod_{k=1}^{K} p(z_k|\boldsymbol{x}, \boldsymbol{\theta}) = \prod_k \text{Ber}(z_k|\sigma(\boldsymbol{w}_{:,k}^\mathsf{T}\boldsymbol{x})) \tag{4.16}$$

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

$$p(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{\theta}) = \prod_d p(x_d|\boldsymbol{z}, \boldsymbol{\theta}) = \prod_d \text{Ber}(x_d|\sigma(\boldsymbol{w}_{d,:}^\mathsf{T}\boldsymbol{z})) \tag{4.17}$$

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

$$\mathbb{E}\left[\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{\theta}\right] = \sigma(\mathbf{W}^\mathsf{T}\boldsymbol{x}) \tag{4.18}$$

$$\mathbb{E}\left[\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{\theta}\right] = \sigma(\mathbf{W}\boldsymbol{z}) \tag{4.19}$$

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 4.16, 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.

4.2.2.2 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]:

$$\mathcal{E}(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{\theta}) \triangleq -\sum_{d=1}^{D}\sum_{k=1}^{K}\sum_{c=1}^{C} x_d^c z_k w_{dk}^c - \sum_{d=1}^{D}\sum_{c=1}^{C} x_d^c b_d^c - \sum_{k=1}^{K} z_k c_k \tag{4.20}$$

The full conditionals are given by

$$p(x_d = c^*|\boldsymbol{z}, \boldsymbol{\theta}) = \mathcal{S}(\{b_d^c + \sum_k z_k w_{dk}^c\}_{c=1}^C))[c^*] \tag{4.21}$$

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

4.2.2.3 Gaussian RBMs

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

$$\mathcal{E}(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta}) = -\sum_{d=1}^{D}\sum_{k=1}^{K} w_{dk} z_k x_d - \frac{1}{2}\sum_{d=1}^{D}(x_d - b_d)^2 - \sum_{k=1}^{K} a_k z_k \tag{4.23}$$

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 **??**):

$$\mathcal{N}_c(\boldsymbol{x}|\boldsymbol{\eta}, \boldsymbol{\Lambda}) \propto \exp(\boldsymbol{\eta}^\mathsf{T}\boldsymbol{x} - \frac{1}{2}\boldsymbol{x}^\mathsf{T}\boldsymbol{\Lambda}\boldsymbol{x}) \tag{4.24}$$

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

$$p(x_d|\boldsymbol{z}, \boldsymbol{\theta}) = \mathcal{N}(x_d|b_d + \sum_k w_{dk} z_k, 1) \tag{4.25}$$

$$p(z_k = 1|\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(c_k + \sum_d w_{dk} x_d\right) \tag{4.26}$$

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

4.2.2.4 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.

4.2.3 Feature induction for a maxent spelling model

In some applications, we assume the features $\boldsymbol{\phi}(\boldsymbol{x})$ are known. However, it is possible to learn the features in a maxent model in an unsupervised way; this is known as **feature induction**.

A common approach to feature induction, first proposed in [DDL97; ZWM97], is to start with a base set of features, and then to continually create new feature combinations out of old ones, greedily adding the best ones to the model.

As an example of this approach, [DDL97] describe how to build models to represent English spelling. This can be formalized as a probability distribution over variable length strings, $p(\boldsymbol{x}|\boldsymbol{\theta})$, where x_t is a letter in the English alphabet. Initially the model has no features, which represents the uniform distribution. The algorithm starts by choosing to add the feature

$$\phi_1(\boldsymbol{x}) = \sum_i \mathbb{I}(x_i \in \{a, \ldots, z\}) \tag{4.27}$$

which checks if any letter is lower case or not. After the feature is added, the parameters are (re)-fit by maximum likelihood (a computationally difficult problem, which we discuss in **??**). For this feature, it turns out that $\hat{\theta}_1 = 1.944$, which means that a word with a lowercase letter in any position is about $e^{1.944} \approx 7$ times more likely than the same word without a lowercase letter in that position. Some samples from this model, generated using (annealed) Gibbs sampling (described in **??**), are shown below.[2]

```
m, r, xevo, ijjiir, b, to, jz, gsr, wq, vf, x, ga, msmGh, pcp, d, oziVlal, hzagh, yzop, io,
advzmxnv, ijv_bolft, x, emx, kayerf, mlj, rawzyb, jp, ag, ctdnnnbg, wgdw, t, kguv, cy, spxcq,
uzflbbf, dxtkkn, cxwx, jpd, ztzh, lv, zhpkvnu, l^, r, qee, nynrx, atze4n, ik, se, w, lrh, hp+,
yrqyka'h, zcngotcnx, igcump, zjcjs, lqpWiqu, cefmfhc, o, lb, fdcY, tzby, yopxmvk, by, fz,, t,
govyccm, ijyiduwfzo, 6xr, duh, ejv, pk, pjw, l, fl, w
```

The second feature added by the algorithm checks if two adjacent characters are lower case:

$$\phi_2(\boldsymbol{x}) = \sum_{i \sim j} \mathbb{I}\left(x_i \in \{a, \ldots, z\}, x_j \in \{a, \ldots, z\}\right) \tag{4.28}$$

Now the model has the form

$$p(\boldsymbol{x}) = \frac{1}{Z} \exp(\theta_1 \phi_1(\boldsymbol{x}) + \theta_2 \phi_2(\boldsymbol{x})) \tag{4.29}$$

Continuing in this way, the algorithm adds features for the strings `s>` and `ing>`, where > represents the end of word, and for various regular expressions such as `[0-9]`, etc. Some samples from the model with 1000 features, generated using (annealed) Gibbs sampling, are shown below.

```
was, reaser, in, there, to, will, ,, was, by, homes, thing, be, reloverated, ther, which, conists,
at, fores, anditing, with, Mr., proveral, the, ,, ***, on't, prolling, prothere, ,, mento, at, yaou,
1, chestraing, for, have, to, intrally, of, qut, ., best, compers, ***, cluseliment, uster, of,
is, deveral, this, thise, of, offect, inatever, thifer, constranded, stater, vill, in, thase, in,
youse, menttering, and, ., of, in, verate, of, to
```

If we define a feature for every possible combination of letters, we can represent any probability distribution. However, this will overfit. The power of maxent approach is that we can choose which features matter for the domain.

An alternative approach is to introduce latent variables, that implicitly model correlations amongst the visible nodes, rather than explicitly having to learn feature functions. See **??** for an example of such a model.

4.2.4 Relational UGMs

We can create **relational UGMs** in a manner which is analogous to relational DGMs (**??**). This is particularly useful in the discriminative setting, for the same reasons that undirected CRFs are preferable to conditional DGMs (see **??**).

For example, suppose we are interested in the problem of classifying web pages of a university into types (e.g., student, professor, admin, etc.) Obviously we can do this based on the contents of the page (e.g., words, pictures, layout, etc.) However, we might also suppose there is information in the hyper-link structure itself. For example, it might be likely for students to cite professors, and professors to cite other professors, but there may be no links between admin pages and students/ professors. When faced with a web page whose label is ambiguous, we can bias our estimate based on the estimated labels of its neighbors, as in a CRF. This process is known as **collective classification** (see e.g., [Sen+08]). To specify the CRF structure for a web-graph of arbitrary size and shape, we just specify a template graph and potential functions, and then unroll the template appropriately to match the topology of the web, making use of parameter tying.

[2] We thank John Lafferty for sharing this example.

Fr(A,A)	Fr(B,B)	Fr(B,A)	Fr(A,B)	Sm(A)	Sm(B)	Ca(A)	Ca(B)
1	1	0	1	1	1	1	1
1	1	0	1	1	0	0	0
1	1	0	1	1	1	0	1

Table 4.3: Some possible joint instantiations of the 8 variables in the smoking example.

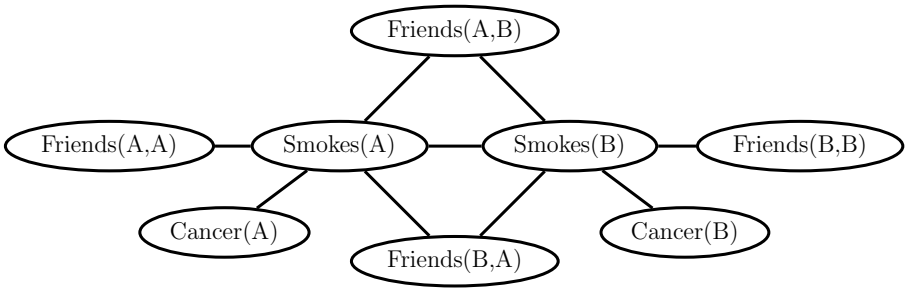


Figure 4.7: An example of a ground Markov logic network represented as a pairwise MRF for 2 people. Adapted from Figure 2.1 from [DL09]. Used with kind permission of Pedro Domingos.

4.2.5 Markov logic networks

One particularly popular way of specifying relational UGMs is to use **first-order logic** rather than a graphical description of the template. The result is known as a **Markov logic network** [RD06; Dom+06; DL09].

For example, consider the sentences "Smoking causes cancer" and "If two people are friends, and one smokes, then so does the other". We can write these sentences in first-order logic as follows:

$$\forall x.Sm(x) \implies Ca(x) \tag{4.30}$$

$$\forall x.\forall y.Fr(x,y) \wedge Sm(x) \implies Sm(y) \tag{4.31}$$

where Sm and Ca are predicates, and Fr is a relation.

It is convenient to write all formulas in **conjunctive normal form** (CNF), also known as **clausal form**. In this case, we get

$$\neg Sm(x) \vee Ca(x) \tag{4.32}$$

$$\neg Fr(x,y) \vee \neg Sm(x) \vee Sm(y) \tag{4.33}$$

The first clause can be read as "Either x does not smoke or he has cancer", which is logically equivalent to Equation (4.30). (Note that in a clause, any unbound variable, such as x, is assumed to be universally quantified.)

Suppose there are just two objects (people) in the world, Anna and Bob, which we will denote by **constant symbols** A and B. We can then create 8 binary random variables $Sm(x)$, $Ca(x)$, and $Fr(x,y)$ for $x, y \in \{A, B\}$. This defines 2^8 **possible worlds**, some of which are shown in Table 4.3.[3]

Our goal is to define a probability distribution over these joint assignments. We can do this by creating a UGM with these variables, and adding a potential function to capture each logical rule or constraint. For example, we can encode the rule $\neg Sm(x) \vee Ca(x)$ by creating a potential function $\Psi(Sm(x), Ca(x))$, where

[3]Note that we have not encoded the fact that Fr is a symmetric relation, so $Fr(A,B)$ and $Fr(B,A)$ might have different values. Similarly, we have the "degenerate" nodes $Fr(A)$ and $Fr(B)$, since we did not enforce $x \neq y$ in Equation (4.31). (If we add such constraints, then the model compiler, which generates the ground network, should avoid creating redundant nodes.)

we define

$$\Psi(Sm(x), Ca(x)) = \begin{cases} 1 & \text{if } \neg Sm(x) \vee Ca(x) = T \\ 0 & \text{if } \neg Sm(x) \vee Ca(x) = F \end{cases} \tag{4.34}$$

The result is the UGM in Figure 4.7.

The above approach will assign non-zero probability to all logically valid worlds. However, logical rules may not always be true. For example, smoking does not always cause cancer. We can relax the hard constraints by using non-zero potential functions. In particular, we can associate a weight with each rule, and thus get potentials such as

$$\Psi(Sm(x), Ca(x)) = \begin{cases} e^w & \text{if } \neg Sm(x) \vee Ca(x) = T \\ e^0 & \text{if } \neg Sm(x) \vee Ca(x) = F \end{cases} \tag{4.35}$$

where the value of $w > 0$ controls strongly we want to enforce the corresponding rule.

The overall joint distribution has the form

$$p(\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{w})} \exp(\sum_i w_i n_i(\boldsymbol{x})) \tag{4.36}$$

where $n_i(\boldsymbol{x})$ is the number of instances of clause i which evaluate to true in assignment $\boldsymbol{x}$.

Given a grounded MLN model, we can then perform inference using standard methods. Of course, the ground models are often extremely large, so more efficient inference methods, which avoid creating the full ground model (known as **lifted inference**), must be used. See [DL09; KNP11] for details.

One way to gain tractability is to relax the discrete problem to a continuous one. This is the basic idea behind **hinge-loss MRFs** [Bac+15], which support exact inference using scalable convex optimization. There is a template language for this model family known as **probabilistic soft logic**, which has a similar "flavor" to MLN, although it is not quite as expressive.

Recently MLNs have been combined with DL in various ways. For example, [Zha+20] uses graph neural networks for inference. And [WP18] uses MLNs for evidence fusion, where the noisy predictions come from DNNs trained using weak supervision.

Finally, it is worth noting one subtlety which arises with undirected models, namely that the size of the unrolled model, which depends on the number of objects in the universe, can affect the results of inference, even if we have no data about the new objects. For example, consider an undirected chain of length T, with T hidden nodes z_t and T observed nodes y_t; call this model M_1. Now suppose we double the length of the chain to $2T$, without adding more evidence; call this model M_2. We find that $p(z_t|y_{1:T}, M_1) \neq p(z_t|y_{1:T}, M_2)$, for $t = 1 : T$, even though we have not added new information, due to the different partition functions. This does not happen with a directed chain, because the newly added nodes can be marginalized out without affecting the original nodes, since the model is locally normalized and therefore modular. See [JBB09; Poo+12] for further discussion.

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.

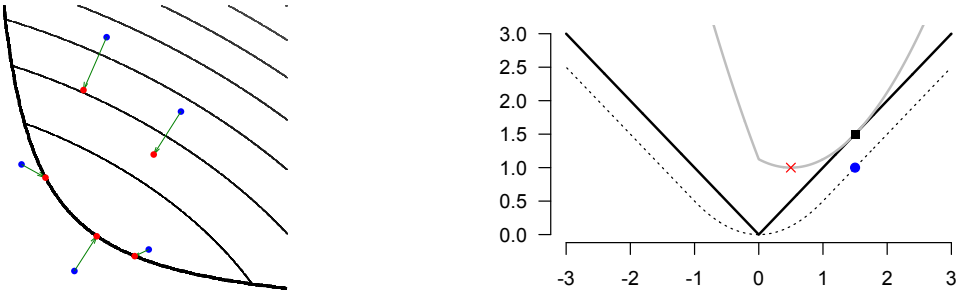
For example, suppose f is the indicator function for the convex set $\mathcal{C}$, i.e.,

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

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 6.5).

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

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

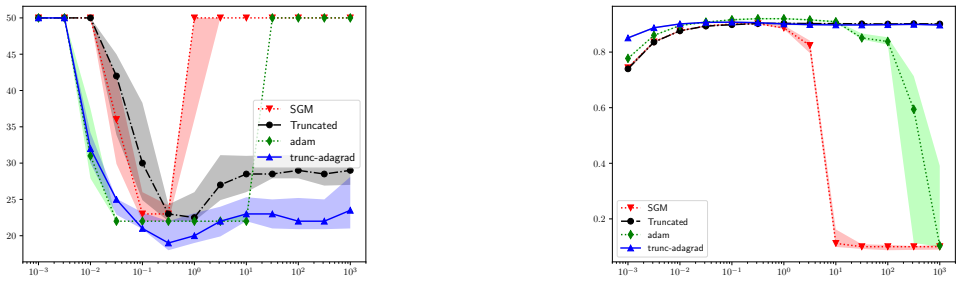
where $\boldsymbol{g}_t = \nabla_{\boldsymbol{\theta}} \ell_t(\boldsymbol{\theta}_t)$. Now we compute a proximal update to this approximate objective:

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

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 Mirror descent

We can extend the proximal point update in Equation (6.34) by replacing the Euclidean distance term $||\boldsymbol{\theta} - \boldsymbol{\theta}_t||_2^2$ by a more general Bregman divergence (**??**),

$$D_h(\boldsymbol{x}, \boldsymbol{y}) = h(\boldsymbol{x}) - \left[h(\boldsymbol{y}) + \nabla h(\boldsymbol{y})^\mathsf{T}(\boldsymbol{x} - \boldsymbol{y})\right] \tag{6.42}$$

where $h(\boldsymbol{x})$ is a strongly convex function. This gives the following update:

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

Suppose we make a linear approximation to $\mathcal{L}(\boldsymbol{\theta})$.

$$\hat{\mathcal{L}}_t(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}_t) + \boldsymbol{g}_t^\mathsf{T}(\boldsymbol{\theta} - \boldsymbol{\theta}_t) \tag{6.44}$$

where $\boldsymbol{g}_t = \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t)$. If we perform a proximal update with this linear approximation using Euclidean distance, we get a standard gradient update, as we showed in Section 6.1.3.2, However, combining this with the Bregman divergence gives the following update:

$$\boldsymbol{\theta}_{t+1} = \operatorname*{argmin}_{\boldsymbol{\theta}} \eta_t \boldsymbol{g}_t^\mathsf{T} \boldsymbol{\theta} + D_h(\boldsymbol{\theta}, \boldsymbol{\theta}_t) \tag{6.45}$$

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.

6.1.5 Proximal gradient method

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

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_s(\boldsymbol{\theta}) + \mathcal{L}_r(\boldsymbol{\theta}) \tag{6.46}$$

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

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

$$\boldsymbol{\theta}_{t+1} = \text{prox}_{\eta_t \mathcal{L}_r}(\boldsymbol{\theta}_t - \eta_t \nabla \mathcal{L}_s(\boldsymbol{\theta}_t)) \tag{6.47}$$

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:

$$\tilde{\boldsymbol{\theta}}_{t+1} = \boldsymbol{\theta}_t + \beta_t(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}) \tag{6.48}$$

$$\boldsymbol{\theta}_{t+1} = \text{prox}_{\eta_t \mathcal{L}_r}(\tilde{\boldsymbol{\theta}}_{t+1} - \eta_t \nabla \mathcal{L}_s(\tilde{\boldsymbol{\theta}}_{t+1})) \tag{6.49}$$

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:

$$\boldsymbol{\theta}_{t+1} = \text{prox}_{\eta_t \mathcal{L}_r}(\boldsymbol{\theta}_t - \eta_t \nabla \mathcal{L}_s(\boldsymbol{\theta}_t, \boldsymbol{z}_t)) \tag{6.50}$$

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].

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

Suppose we are interested in fitting a linear regression model with a sparsity-promoting prior on the weights, as in the lasso model (**??**). 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

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_s(\boldsymbol{\theta}) + \mathcal{L}_r(\boldsymbol{\theta}) = \frac{1}{2}||\mathbf{X}\boldsymbol{\theta} - \boldsymbol{y}||_2^2 + \lambda||\boldsymbol{\theta}||_1 \tag{6.51}$$

The proximal gradient descent update can be written as

$$\boldsymbol{\theta}_{t+1} = \text{SoftThreshold}_{\eta_t\lambda}(\boldsymbol{\theta}_t - \eta_t \nabla \mathcal{L}_s(\boldsymbol{\theta}_t)) \tag{6.52}$$

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.

6.1.6 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:

$$\text{minimize} \;\; \mathcal{L}_s(\boldsymbol{x}) + \mathcal{L}_r(\boldsymbol{z}) \;\; \text{s.t.} \;\; \boldsymbol{x} - \boldsymbol{z} = \mathbf{0} \tag{6.53}$$

This is called **consensus form**.

The corresponding **augmented Langragian** is given by

$$L_\rho(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y}) = \mathcal{L}_s(\boldsymbol{x}) + \mathcal{L}_r(\boldsymbol{z}) + \boldsymbol{y}^\mathsf{T}(\boldsymbol{x} - \boldsymbol{z}) + \frac{\rho}{2}||\boldsymbol{x} - \boldsymbol{z}||_2^2 \tag{6.54}$$

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:

$$\boldsymbol{x}_{t+1} = \underset{\boldsymbol{x}}{\text{argmin}}\, L_\rho(\boldsymbol{x}, \boldsymbol{z}_t, \boldsymbol{y}_t) \tag{6.55}$$

$$\boldsymbol{z}_{t+1} = \underset{\boldsymbol{z}}{\text{argmin}}\, L_\rho(\boldsymbol{x}_{t+1}, \boldsymbol{z}, \boldsymbol{y}_t) \tag{6.56}$$

$$\boldsymbol{y}_{t+1} = \boldsymbol{y}_t + \rho(\boldsymbol{x}_{t+1} - \boldsymbol{z}_{t+1}) \tag{6.57}$$

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:

$$\boldsymbol{x}_{t+1} = \underset{\boldsymbol{x}}{\text{argmin}} \left(\mathcal{L}_s(\boldsymbol{x}) + \boldsymbol{y}_t^\mathsf{T}\boldsymbol{x} + \frac{\rho}{2}||\boldsymbol{x} - \boldsymbol{z}_t||_2^2 \right) \tag{6.58}$$

$$\boldsymbol{z}_{t+1} = \underset{\boldsymbol{z}}{\text{argmin}} \left(\mathcal{L}_r(\boldsymbol{z}) - \boldsymbol{y}_t^\mathsf{T}\boldsymbol{z} + \frac{\rho}{2}||\boldsymbol{x}_{t+1} - \boldsymbol{z}||_2^2 \right) \tag{6.59}$$

If we combine the linear and quadratic terms, we get

$$\boldsymbol{x}_{t+1} = \underset{\boldsymbol{x}}{\text{argmin}} \left(\mathcal{L}_s(\boldsymbol{x}) + \frac{\rho}{2}||\boldsymbol{x} - \boldsymbol{z}_t + (1/\rho)\boldsymbol{y}_t||_2^2 \right) \tag{6.60}$$

$$\boldsymbol{z}_{t+1} = \underset{\boldsymbol{z}}{\text{argmin}} \left(\mathcal{L}_r(\boldsymbol{z}) + \frac{\rho}{2}||\boldsymbol{x}_{t+1} - \boldsymbol{z} - (1/\rho)\boldsymbol{y}_t||_2^2 \right) \tag{6.61}$$

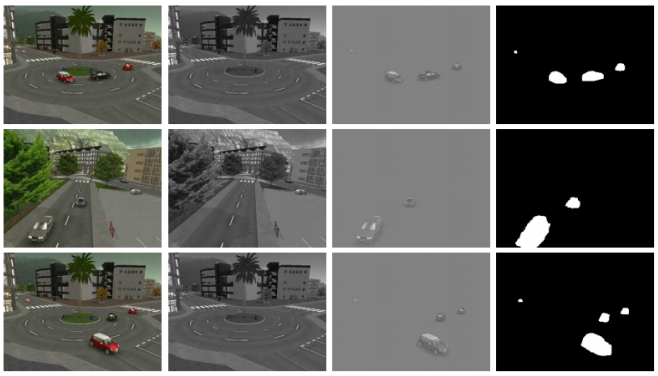


Figure 6.3: Robust PCA applied to some frames from a surveillance video. First column is input image. Second column is low-rank background model. Third model is sparse foreground model. Last column is derived foreground mask. From Figure 1 of [Bou+17]. Used with kind permission of Thierry Bouwmans.

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:

$$\boldsymbol{x}_{t+1} = \text{prox}_{\lambda \mathcal{L}_s}(\boldsymbol{z}_t - \boldsymbol{u}_t) \tag{6.62}$$

$$\boldsymbol{z}_{t+1} = \text{prox}_{\lambda \mathcal{L}_r}(\boldsymbol{x}_{t+1} + \boldsymbol{u}_t) \tag{6.63}$$

$$\boldsymbol{u}_{t+1} = \boldsymbol{u}_t + \boldsymbol{x}_{t+1} - \boldsymbol{z}_{t+1} \tag{6.64}$$

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].

6.1.6.1 Example: robust PCA

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

Consider the following **matrix decomposition problem**:

$$\underset{\mathbf{X}_{1:J}}{\text{minimize}} \sum_{j=1}^{J} \gamma_j \phi_j(\mathbf{X}_j) \quad \text{s.t.} \quad \sum_{j=1}^{J} \mathbf{X}_j = \mathbf{A} \tag{6.65}$$

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:

$$\text{minimize} \quad ||\mathbf{A} - (\mathbf{L} + \mathbf{S})||_F^2 + \gamma_L ||\mathbf{L}||_* + \gamma_S ||\mathbf{S}||_1 \tag{6.66}$$

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

$$\phi_1(\mathbf{X}_1) = ||\mathbf{X}_1||_*, \quad \phi_2(\mathbf{X}_2) = ||\mathbf{X}_2||_1, \quad \phi_3(\mathbf{X}_3) = ||\mathbf{X}_3||_F^2 \tag{6.67}$$

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

$$\mathbf{X}_{j,t+1} = \text{prox}_{\eta_t \phi_j}(\mathbf{X}_{j,t} - \overline{\mathbf{X}_t} + \frac{1}{N}\mathbf{A} - \mathbf{U}_t) \tag{6.68}$$

$$\mathbf{U}_{t+1} = \mathbf{U}_t + \overline{\mathbf{X}}_{t+1} - \frac{1}{J}\mathbf{A} \tag{6.69}$$

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:

$$\text{proj}_{\mathcal{C}}(\mathbf{X}_1, \ldots, \mathbf{X}_J) = (\mathbf{X}_1, \ldots, \mathbf{X}_J) - \overline{\mathbf{X}} + \frac{1}{J}\mathbf{A} \tag{6.70}$$

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 Local search

In this section, we discuss heuristic optimization algorithms that try to find the global maximum in a discrete, unstructured search space. These algorithms replace the local gradient based update, which has the form $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta_t \boldsymbol{d}_t$, with the following discrete analog:

$$\boldsymbol{x}_{t+1} = \underset{\boldsymbol{x} \in \text{nbr}(\boldsymbol{x}_t)}{\text{argmax}}\ \mathcal{L}(\boldsymbol{x}) \tag{6.71}$$

where $\text{nbr}(\boldsymbol{x}_t) \subseteq \mathcal{X}$ is the set of **neighbors** of $\boldsymbol{x}_t$. This is called **hill climbing**, **steepest ascent**, or **greedy search**.

If the "neighborhood" of a point contains the entire space, Equation (6.71) will return the global optimum in one step, but usually such a global neighborhood is too large to search exhaustively. Consequently we usually define local neighborhoods. For example, consider the **8-queens problem**. Here the goal is to place queens on an 8×8 chessboard so that they don't attack each other (see Figure 6.6). The state space has the form $\mathcal{X} = 64^8$, since we have to specify the location of each queen on the grid. However, due to the constraints, there are only $8^8 \approx 17M$ feasible states. We define the neighbors of a state to be all possible states generated by moving a single queen to another square in the same column, so each node has $8 \times 7 = 56$ neighbors. According to [RN10, p.123], if we start at a randomly generated 8-queens state, steepest ascent gets stuck at a local maximum 86% of the time, so it only solves 14% of problem instances. However, it is fast, taking an average of 4 steps when it succeeds and 3 when it gets stuck.

In the sections below, we discuss slightly smarter algorithms that are less likely to get stuck in local maxima.

6.2.1 Stochastic local search

Hill climbing is greedy, since it picks the best point in its local neighborhood, by solving Equation (6.71) exactly. One way to reduce the chance of getting stuck in local maxima is to approximately maximize this objective at each step. For example, we can define a probability distribution over the uphill neighbors, proportional to how much they improve, and then sample one at random. This is called **stochastic hill**

climbing. If we gradually decrease the entropy of this probability distribution (so we become greedier over time), we get a method called simulated annealing, which we discuss in **??**.

Another simple technique is to use greedy hill climbing, but then whenever we reach a local maximum, we start again from a different random starting point. This is called **random restart hill climbing**. To see the benefit of this, consider again the 8-queens problem. If each hill-climbing search has a probability of $p \approx 0.14$ of success, then we expect to need $R = 1/p \approx 7$ restarts until we find a valid solution. The expected number of total steps can be computed as follows. Let $N_1 = 4$ be the average number of steps for successful trials, and $N_0 = 3$ be the average number of steps for failures. Then the total number of steps on average is $N_1 + (R-1)N_0 = 4 + 6 \times 3 = 22$. Since each step is quick, the overall method is very fast. For example, it can solve an n-queens problem with n =1M in under a minute.

Of course, solving the n-queens problem is not the most useful task in practice. However, it is typical of several real-world **boolean satisfiability problems**, which arise in problems ranging from AI planning to model checking (see e.g., [SLM92]). In such problems, simple **stochastic local search** (**SLS**) algorithms of the kind we have discussed work surprisingly well (see e.g., [HS05]).

6.2.2 Tabu search

Algorithm 1: Tabu search.

```
t := 0 // counts iterations ;
c := 0 // counts number of steps with no progress;
Initialize x_0 ;
x* := x_0 // current best incumbent;
while c < c_max do
    x_{t+1} = argmax_{x ∈ nbr(x_t) \ {x_{t-τ}, ..., x_{t-1}}} f(x);
    if f(x_t) > f(x*) then
        x* := x_t ;
        c := 0
    else
        c := c + 1
    t := t + 1
return x*
```

Hill climbing will stop as soon as it reaches a local maximum or a plateau. Obviously one can perform a random restart, but this would ignore all the information that had been gained up to this point. A more intelligent alternative is called **tabu search** [GL97]. This is like hill climbing, except it allows moves that decrease (or at least do not increase) the scoring function, provided the move is to a new state that has not been seen before. We can enforce this by keeping a tabu list which tracks the τ most recently visited states. This forces the algorithm to explore new states, and increases the chances of escaping from local maxima. We continue to do this for up to $c_{\max}$ steps (known as the "tabu tenure"). The pseudocode can be found in Algorithm 1. (If we set $c_{\max} = 1$, we get greedy hill climbing.)

For example, consider what happens when tabu search reaches a hill top, $\boldsymbol{x}_t$. At the next step, it will move to one of the neighbors of the peak, $\boldsymbol{x}_{t+1} \in \text{nbr}(\boldsymbol{x}_t)$, which will have a lower score. At the next step, it will move to the neighbor of the previous step, $\boldsymbol{x}_{t+2} \in \text{nbr}(\boldsymbol{x}_{t+1})$; the tabu list prevents it cycling back to $\boldsymbol{x}_t$ (the peak), so it will be forced to pick a neighboring point at the same height or lower. It continues in this way, "circling" the peak, possibly being forced downhill to a lower level-set (an inverse **basin flooding** operation), until it finds a ridge that leads to a new peak, or until it exceeds a maximum number of non-improving moves.

According to [RN10, p.123], tabu search increases the percentage of 8-queens problems that can be solved from 14% to 94%, although this variant takes an average of 21 steps for each successful instance and 64 steps for each failed instance.

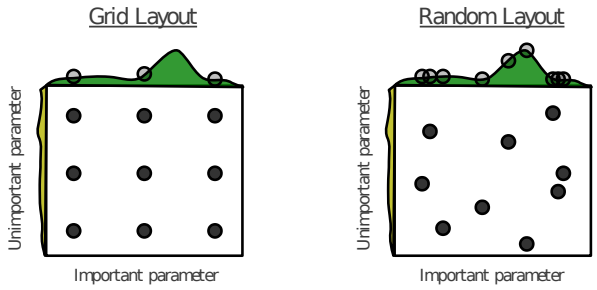


Figure 6.4: Illustration of grid search (left) vs random search (right). From Figure 1 of [BB12]. Used with kind permission of James Bergstra.

6.2.3 Random search

A surprisingly effective strategy in problems where we know nothing about the objective is to use **random search**. In this approach, each iterate $\boldsymbol{x}_{t+1}$ is chosen uniformly at random from $\mathcal{X}$. This should always be tried as a baseline.

In [BB12], they applied this technique to the problem of hyper-parameter optimization for some ML models, where the objective is performance on a validation set. In their examples, the search space is continuous, $\Theta = [0,1]^D$. It is easy to sample from this at random. The standard alternative approach is to quantize the space into a fixed set of values, and then to evaluate them all; this is known as **grid search**. (Of course, this is only feasible if the number of dimensions D is small.) They found that random search outperformed grid search. The intuitive reason for this is that many hyper-parameters do not make much difference to the objective function, as illustrated in Figure 6.4. Consequently it is a waste of time to place a fine grid along such unimportant dimensions.

RS has also been used to optimize the parameters of MDP policies, where the objective has the form $f(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{\tau} \sim \pi_{\boldsymbol{x}}}[R(\boldsymbol{\tau})]$ is the expected reward of trajectories generated by using a policy with parameters $\boldsymbol{x}$. For policies with few free parameters, RS can outperform more sophisticated reinforcement learning methods described in **??**, as shown in [MGR18]. In cases where the policy has a large number of parameters, it is sometimes possible to project them to a lower dimensional random subspace, and perform optimization (either grid search or random search) in this subspace [Li+18].

6.3 Population-based optimization

Stochastic local search (SLS) maintains a single "best guess" at each step, $\boldsymbol{x}_t$. If we run this for T steps, and restart K times, the total cost is TK. A natural alternative is to maintain a set or **population** of K good candidates, $\mathcal{S}_t$, which we try to improve at each step. This is called an an **evolutionary algorithm** (**EA**). If we run this for T steps, it also takes TK time; however, it can often get better results than multi-restart SLS, since the search procedure explores more of the space in parallel, and information from different members of the population can be shared. Many versions of EA are possible, depending on how we update the population at each step, as we discuss in Section 6.3.1.

An alternative to maintaining an explicit set of promising candidates is to maintain a probability distribution over promising candidates. We call this **distribution-based optimization** (**DBO**).[2] We can think of EAs as using a nonparametric representation of this distribution in terms of a "bag of points". We

[2]Note that the terms "**probabilistic optimization**" and "**stochastic optimization**" have already been "taken". (Probabilistic optimization is often used to refer to Bayesian optimization (**??**), and stochastic optimization refers to any optimization problem in which the objective is stochastic.) Another term that is used for DBO is "model based optimization" (see e.g. [BBZ17]). However, this term is ambiguous, because it could either refer to a model of good candidates (i.e., a probability distribution over $\mathcal{X}$) or a cheap (possibly differentiable) approximation to the objective (i.e., a regression function $\mathcal{X} \to \mathbb{R}$), as discussed in **??**.

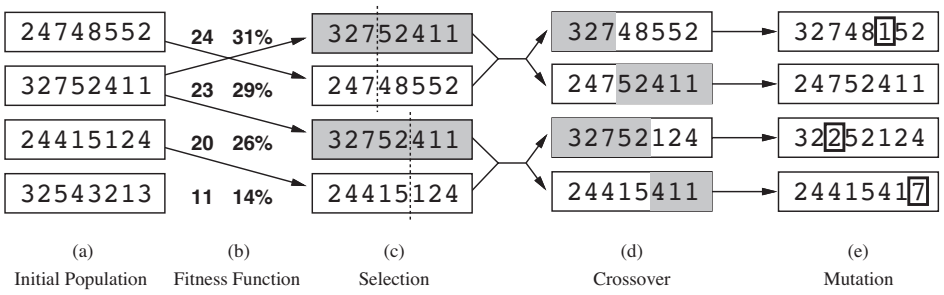


Figure 6.5: Illustration of a genetic algorithm applied to the 8-queens problem. (a) Initial population of 4 strings. (b) We rank the members of the population by fitness, and then compute their probability of mating. Here the integer numbers represent the number of nonattacking pairs of queens, so the global maximum has a value of 28. We pick an individual $\boldsymbol{\theta}$ with probability $p(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta})/Z$, where $Z = \sum_{\boldsymbol{\theta} \in \mathcal{P}} \mathcal{L}(\boldsymbol{\theta})$ sums the total fitness of the population. For example, we pick the first individual with probability $24/78 = 0.31$, the second with probability $23/78 = 0.29$, etc. In this example, we pick the first individual once, the second twice, the third one once, and the last one does not get to breed. (c) A split point on the "chromosome" of each parent is chosen at random. (d) The two parents swap their chromosome halves. (e) We can optionally apply pointwise mutation. From Figure 4.6 of [RN10]. Used with kind permission of Peter Norvig.

can sometimes get better performance by using a parametric distribution, with suitable inductive bias. We discuss some examples in Section 6.3.3.

6.3.1 Evolutionary algorithms

Since EA algorithms draw inspiration from the biological process of evolution, they also borrow a lot of its terminology. The **fitness** of a member of the population is the value of the objective function (possibly normalized across population members). The members of the population at step $t+1$ are called the **offspring**. These can be created by randomly choosing a **parent** from $\mathcal{S}_t$ and applying a random **mutation** to it. This is like asexual reproduction. Alternatively we can create an offspring by choosing two parents from $\mathcal{S}_t$, and then combining them in some way to make a child, as in sexual reproduction; combining the parents is called **recombination**. (It is often followed by mutation.)

The procedure by which parents are chosen is called the **selection function**. In **truncation selection**, each parent is chosen from the fittest K members of the population (known as the **elite set**). In **tournament selection**, each parent is the fittest out of K randomly chosen members. In **fitness proportionate selection**, also called **roulette wheel selection**, each parent is chosen with probability proportional to its fitness relative to the others. We can also "kill off" the oldest members of the population, and then select parents based on their fitness; this is called **regularized evolution** [Rea+19]).

In addition to the selection rule for patents, we need to specify the recombination and mutation rules. There are many possible choices for these heuristics. We briefly mention a few of them below.

- In a **genetic algorithm** (**GA**) [Gol89; Hol92], we use mutation and a particular recombination method based on **crossover**. To implement crossover, we assume each individual is represented as a vector of integers or binary numbers, by analogy to **chromosomes**. We pick a split point along the chromosome for each of the two chosen parents, and then swap the strings, as illustrated in Figure 6.5.
- In **genetic programming** [Koz92], we use use a tree-structured representation of individuals, instead of a bit string. This representation ensures that all crossovers result in valid children, as illustrated in Figure 6.7. Genetic programming can be useful for finding good programs as well as other structured objects, such as neural networks. In **evolutionary programming**, the structure of the tree is fixed and only the numerical parameters are evolved.
- In **surrogate assisted EA**, a surrogate function $\hat{f}(s)$ is used instead of the true objective function $f(s)$ in order to speed up the evaluation of members of the population (see [Jin11] for a survey). This

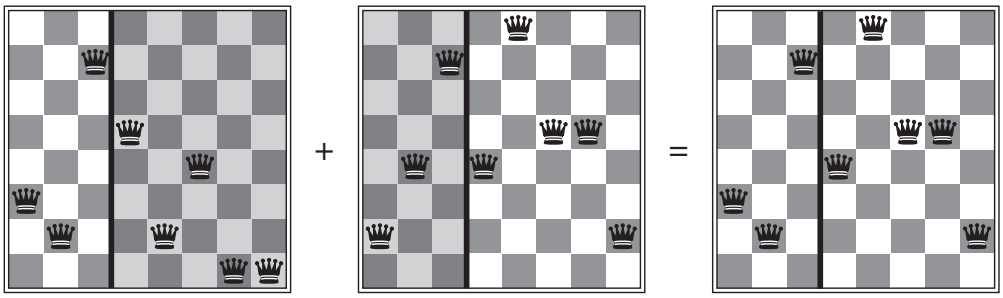


Figure 6.6: The 8-queens states corresponding to the first two parents in Figure 6.5(c) and their first child in Figure 6.5(d). We see that the encoding 32752411 *means that the first queen is in row 3 (counting from the bottom left), the second queen is in row 2, etc. The shaded columns are lost in the crossover, but the unshaded columns are kept. From Figure 4.7 of [RN10]. Used with kind permission of Peter Norvig.*

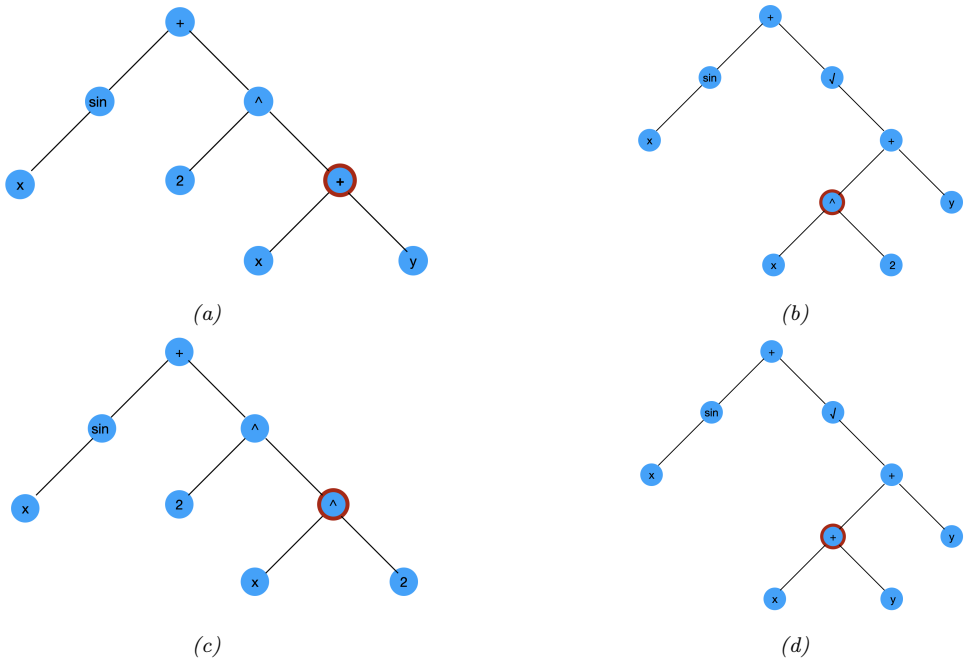


Figure 6.7: Illustration of crossover operator in a genetic program. (a-b) the two parents, representing $\sin(x)+(x+y)^2$ *and* $\sin(x)+\sqrt{x^2+y}$*. The red circles denote the two crossover points. (c-d) the two children, representing* $\sin(x)+(x^2)^2$ *and* $\sin(x)+\sqrt{x+y+y}$*. Adapted from Figure 9.2 of [Mit97]*

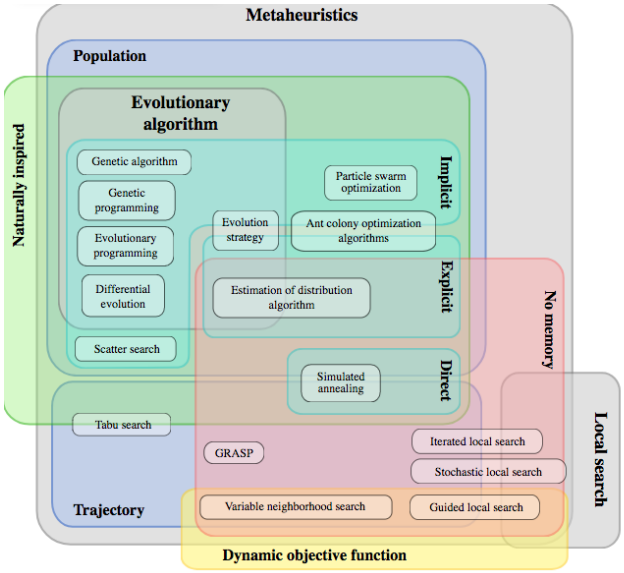


Figure 6.8: A taxonomy of various metaheuristic optimization algorithms. From https://en.wikipedia.org/wiki/Metaheuristic. Used with kind permission of Wikipedia authors Johann Dreo and Caner Candan.

is similar to the use of response surface models in Bayesian optimization (**??**), except it does not deal with the explore-exploit tradeoff.

- In a **memetic algorithm** [MC03], we combine mutation and recombination with standard local search.

Evolutionary algorithms have been applied to a large number of applications, including training neural networks (this combination is known as **neuroevolution** [Sta+19]). An efficient JAX-based library for (neuro)-evolution can be found at https://github.com/google/evojax.

6.3.2 Metaheuristic algorithms

The term "**metaheuristics**" is often used to describe different kinds of local or population-based search algorithms (see e.g., [Luk13]). There are many types of metaheuristic algorithms, and several ways to categorize them, as shown in Figure 6.8. Many of these algorithms are "inspired" by natural phenomena. For example, [AQ+20] proposes to tackle covid19 forecasting using "an improved adaptive neuro-fuzzy inference system (ANFIS) using an enhanced flower pollination algorithm (FPA) by using the salp swarm algorithm (SSA)", and [Mol+18] uses "Chicken Swarm Optimization and Deep Learning for Manufacturing Processes". However, such "inspiration" usually results in ad-hoc algorithms, with no theoretical support (and therefore little reason to use them). As Glover and Sorensen [GS15] memorably put it

> A large (and increasing) number of publications focuses on the development of (supposedly) new metaheuristic frameworks based on metaphors. The list of natural or man-made processes that has been used as the basis for a metaheuristic framework now includes such diverse processes as bacterial foraging, river formation, biogeography, musicians playing together, electromagnetism, gravity, colonization by an empire, mine blasts, league championships, clouds, and so forth. An important subcategory is found in metaheuristics based on animal behavior. Ants, bees, bats, wolves, cats, fireflies, eagles, dolphins, frogs, salmon, vultures, termites, flies, and many others, have all been used to inspire a "novel" metaheuristic. [...] As a general rule, publication of papers on metaphor-based metaheuristics has been limited to second-tier journals and conferences, but some recent exceptions to this rule can be found. Sörensen [Sör15] states that *research in this direction is fundamentally flawed.*

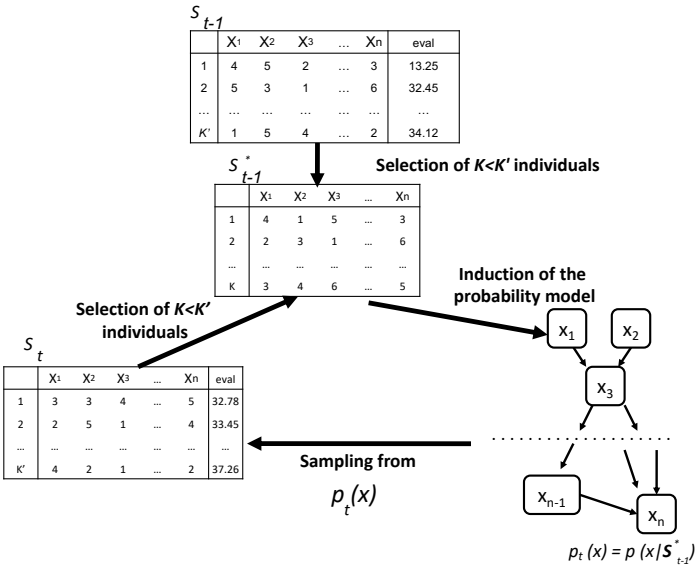


Figure 6.9: Illustration of the BOA algorithm (EDA applied to a generative model structured as a Bayes net). Adapted from Figure 3 of [PHL12].

In view of this assessment, we do not discuss such methods any further.

6.3.3 Estimation of distribution algorithms

EA methods maintain a population of good candidate solutions, which can be thought of as an implicit (nonparametric) density model over states with high fitness. [BC95] proposed to "remove the genetics from GAs", by explicitly learning a probabilistic model over the configuration space that puts its mass on high scoring solutions. Thay is, the population becomes the set of parameters of a generative model, $\boldsymbol{\theta}_t$.

One way to learn such as model is as follows. We start by creating a sample of $K' > K$ candidate solutions from the current model, $\mathcal{S}_t = \{\boldsymbol{x}_k \sim p(\boldsymbol{x}|\boldsymbol{\theta}_t)\}$. We then rank the samples using the fitness function, and then pick the most promising subset $\mathcal{S}_t^*$ of size K using a selection operator (this is known as **truncation selection**). Finally, we fit a new probabilistic model $p(\boldsymbol{x}|\boldsymbol{\theta}_{t+1})$ to $\mathcal{S}_t^*$ using maximum likelihood estimation. This is called the **estimation of distribution** or **EDA** algorithm (see e.g., [LL02; PSCP06; Hau+11; PHL12; Hu+12; San17; Bal17]).

Note that EDA is equivalent to minimizing the cross-entropy between the empirical distribution defined by $\mathcal{S}_t^*$ and the model distribution $p(\boldsymbol{x}|\boldsymbol{\theta}_{t+1})$. Thus EDA is related to the **cross entropy method**, as described in Section 6.3.4, although CEM usually assumes the special case where $p(\boldsymbol{x}|\boldsymbol{w}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. EDA is also closely related to the EM algorithm, as discussed in [Bro+20].

As a simple example, suppose the configuration space is bit strings of length D, and the fitness function is $f(\boldsymbol{x}) = \sum_{d=1}^{D} x_d$, where $x_d \in \{0, 1\}$ (this is called the **one-max** function in the EA literature). A simple probabilistic model for this is a fully factored model of the form $p(\boldsymbol{x}|\boldsymbol{\theta}) = \prod_{d=1}^{D} \text{Ber}(x_d|\theta_d)$. Using this model inside of DBO results in a method called univariate marginal distribution algorithm or **UMDA**.

We can estimate the parameters of the Bernoulli model by setting θ_d to the fraction of samples in $\mathcal{S}_t^*$ that have bit d turned on. Alternatively, we can incrementally adjust the parameters. The population-based incremental learning (**PBIL**) algorithm [BC95] applies this idea to the factored Bernoulli model, resulting in the following update:

$$\hat{\theta}_{d,t+1} = (1 - \eta_t)\hat{\theta}_{d,t} + \eta_t \overline{\theta}_{d,t} \tag{6.72}$$

where $\overline{\theta}_{d,t} = \frac{1}{N_t} \sum_{k=1}^{K} \mathbb{I}(x_{k,d} = 1)$ is the MLE estimated from the $K = |\mathcal{S}_t^*|$ samples generated in the current iteration, and η_t is a learning rate.

It is straightforward to use more expressive probability models that capture dependencies between the parameters (these are known as **building blocks** in the EA literature). For example, in the case of real-valued parameters, we can use a multivariate Gaussian, $p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The resulting method is called the **estimation of multivariate normal algorithm** or **EMNA**, [LL02]. (See also Section 6.3.4.)

For discrete random variables, it is natural to use probabilistic graphical models (**??**) to capture dependencies between the variables. [BD97] learn a tree-structured graphical model using the Chow-Liu algorithm (Section 30.1.2); [BJV97] is a special case of this where the graph is a tree. We can also learn more general graphical model structures (see e.g., [LL02]). We typically use a Bayes net (**??**), since we can use ancestral sampling (**??**) to easily generate samples; the resulting method is therefore called the **Bayesian Optimization Algorithm** (**BOA**) [PGCP00].[3] The hierarchical BOA (**hBOA**) algorithm [Pel05] extends this by using decision trees and decision graphs to represent the local CPTs in the Bayes net (as in [CHM97]), rather than using tables. In general, learning the structure of the probability model for use in EDA is called **linkage learning**, by analogy to how genes can be linked together if they can be co-inherited as a building block.

We can also use deep generative models to represent the distribution over good candidates. For example, [CSF16] use denoising autoencoders and NADE models (**??**), [Bal17] uses a DNN regressor which is then inverted using gradient descent on the inputs, [PRG17] uses RBMs (**??**), [GSM18] uses VAEs (**??**), etc. Such models might take more data to fit (and therefore more function calls), but can potentially model the probability landscape more faithfully. (Whether that translates to better optimization performance is not clear, however.)

6.3.4 Cross-entropy method

The **cross-entropy method** [Rub97; RK04; Boe+05] is a special case of EDA (Section 6.3.3) in which the population is represented by a multivariate Gaussian. In particular, we set $\boldsymbol{\mu}_{t+1}$ and $\boldsymbol{\Sigma}_{t+1}$ to the empirical mean and covariance of $\mathcal{S}^*_{t+1}$, which are the top K samples. This is closely related to the SMC algorithm for sampling rare events discussed in **??**.

The CEM is sometimes used for model-based RL (**??**), since it is simple and can find reasonably good optima of multi-modal objectives. It is also sometimes used inside of Bayesian optimization (**??**), to optimize the multi-modal acquisition function (see [BK10]).

6.3.4.1 Differentiable CEM

The **differentiable CEM** method of [AY19] replaces the top K operator with a soft, differentiable approximation, which allows the optimizer to be used as part of an end-to-end differentiable pipeline. For example, we can use this to create a differentiable model predictive control (MPC) algorithm (**??**), as described in **??**.

The basic idea is as follows. Let $\mathcal{S}_t = \{\boldsymbol{x}_{t,i} \sim p(\boldsymbol{x}|\boldsymbol{\theta}_t) : i = 1 : K'\}$ represent the current population, with fitness values $v_{t,i} = f(\boldsymbol{x}_{t,i})$. Let $v^*_{t,K}$ be the K'th smallest value. In CEM, we compute the set of top K samples, $\mathcal{S}^*_t = \{i : v_{t,i} \geq v^*_{t,K}\}$, and then update the model based on these: $\boldsymbol{\theta}_{t+1} = \text{argmax}_{\boldsymbol{\theta}} \sum_{i \in \mathcal{S}_t} p_t(i) \log p(\boldsymbol{x}_{t,i}|\boldsymbol{\theta})$, where $p_t(i) = \mathbb{I}\left(i \in \mathcal{S}^*_t\right) / |\mathcal{S}^*_t|$. In the differentiable version, we replace the sparse distribution $\boldsymbol{p}_t$ with the "soft" dense distribution $\boldsymbol{q}_t = \Pi(\boldsymbol{p}_t; \tau, K)$, where

$$\Pi(\boldsymbol{p}; \tau, K) = \underset{\mathbf{0} \leq \boldsymbol{q} \leq \mathbf{1}}{\text{argmin}} -\boldsymbol{p}^\mathsf{T}\boldsymbol{q} - \tau\, \mathbb{H}(\boldsymbol{q}) \;\; \text{s.t.} \;\; \mathbf{1}^\mathsf{T}\boldsymbol{q} = K \tag{6.73}$$

projects the distribution $\boldsymbol{p}$ onto the polytope of distributions which sum to K. (Here $\mathbb{H}(\boldsymbol{q}) = -\sum_i q_i \log(q_i) + (1 - q_i) \log(1 - q_i)$ is the entropy, and $\tau > 0$ is a temperature parameter.) This projection operator (and hence the whole DCEM algorithm) can be backpropagated through using implicit differentiation [AKZK19].

[3] This should not be confused with the Bayesian optimization methods we discuss in **??**, that uses response surface modeling to model $p(f(\boldsymbol{x}))$ rather than $p(\boldsymbol{x}^*)$.

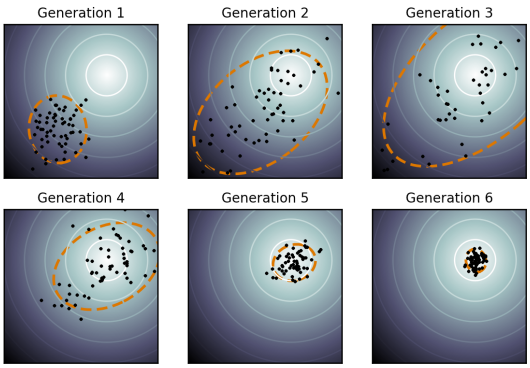


Figure 6.10: Illustration of the CMA-ES method applied to a simple 2d function. The dots represent members of the population, and the dashed orange ellipse represents the multivariate Gaussian. From https://en.wikipedia.org/wiki/CMA-ES. Used with kind permission of Wikipedia author Sentewolf.

6.3.5 Natural evolutionary strategies

Evolution strategies [Wie+14] are a form of distribution-based optimization in which the distribution over the population is represented by a Gaussian, $p(\boldsymbol{x}|\boldsymbol{\theta}_t)$ (see e.g., [Sal+17]). Unlike CEM, the parameters are updated using gradient ascent applied to the expected value of the objective, rather than using MLE on a set of elite samples. More precisely, consider the smoothed objective $\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{p(\boldsymbol{x}|\boldsymbol{\theta})}[f(\boldsymbol{x})]$. We can use the REINFORCE estimator (**??**) to compute the gradient of this objective as follows:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{p(\boldsymbol{x}|\boldsymbol{\theta})}[f(\boldsymbol{x}) \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}|\boldsymbol{\theta})] \tag{6.74}$$

This can be approximated by drawing Monte Carlo samples. If the probability model is in the exponential family, we can compute the natural gradient (**??**), rather than the "vanilla" gradient; such methods are called **natural evolution strategies** [Wie+14].

6.3.5.1 CMA-ES

The **CMA-ES** method of [Han16], which stands for "covariance matrix adaptation evolution strategy" is a kind of NES. It is very similar to CEM except it updates the parameters in a special way. In particular, instead of computing the new mean and covariance using unweighted MLE on the elite set, we attach weights to the elite samples based on their rank. We then set the new mean to the weighted MLE of the elite set.

The update equations for the covariance are more complex. In particular, "evolutionary paths" are also used to accumulate the search directions across successive generations, and these are used to update the covariance. It can be show that the resulting updates approximate the natural gradient of $\mathcal{L}(\boldsymbol{\theta})$ without explicitly modeling the Fisher information matrix [Oll+17].

Figure 6.10 illustrates the method in action.

6.4 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.

6.4.1 Example: computing Fibonnaci numbers

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

$$F_i = F_{i-1} + F_{i-2} \tag{6.75}$$

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 2. Unfortunately, this takes exponential time. For example, evaluating fib(5) proceeds as follows:

$$\begin{aligned} F_5 &= F_4 + F_3 && (6.76)\\ &= (F_3 + F_2) + (F_2 + F_1) && (6.77)\\ &= ((F_2 + F_1) + (F_1 + F_0)) + ((F_1 + F_0) + F_1) && (6.78)\\ &= (((F_1 + F_0) + F_1) + (F_1 + F_0))((F_1 + F_0) + F_1) && (6.79) \end{aligned}$$

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 3.

Algorithm 2: 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 3: 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
```

6.4.2 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.

6.5 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].

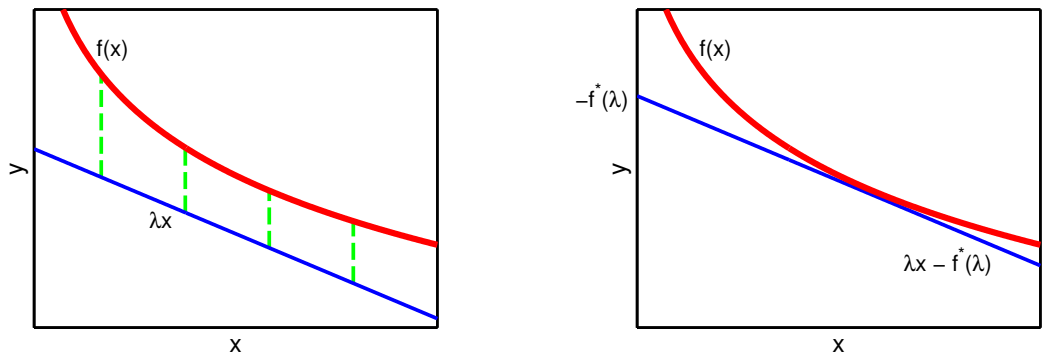


Figure 6.11: Illustration of a conjugate functon. Red line is original function $f(x)$, and the blue line is a linear lower bound λx. To make the bound tight, we find the x where $\nabla f(x)$ is parallel to λ, and slide the line up to touch there; the amount we slide up is given by $f^*(\lambda)$. Adapted from Figure 10.11 of [Bis06].

6.5.1 Introduction

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

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) \triangleq \boldsymbol{\lambda}^{\mathsf{T}} \boldsymbol{x} - f^*(\boldsymbol{\lambda}) \leq f(\boldsymbol{x}) \tag{6.80}$$

where $\boldsymbol{\lambda}$ is the slope, which we choose, and $f^*(\boldsymbol{\lambda})$ is the intercept, which we solve for below. See Figure 6.11(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 6.11(b). At $\boldsymbol{x}_\lambda$, we minimize the distance between the function and the lower bound:

$$\boldsymbol{x}_\lambda \triangleq \operatorname*{argmin}_{\boldsymbol{x}} f(x) - \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = \operatorname*{argmin}_{\boldsymbol{x}} f(x) - \boldsymbol{\lambda}^{\mathsf{T}} \boldsymbol{x} \tag{6.81}$$

Since the bound is tight at this point, we have

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

and hence

$$f^*(\boldsymbol{\lambda}) = \boldsymbol{\lambda}^{\mathsf{T}} \boldsymbol{x}_\lambda - f(\boldsymbol{x}_\lambda) = \max_{\boldsymbol{x}} \boldsymbol{\lambda}^{\mathsf{T}} \boldsymbol{x} - f(\boldsymbol{x}) \tag{6.83}$$

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 **??**.

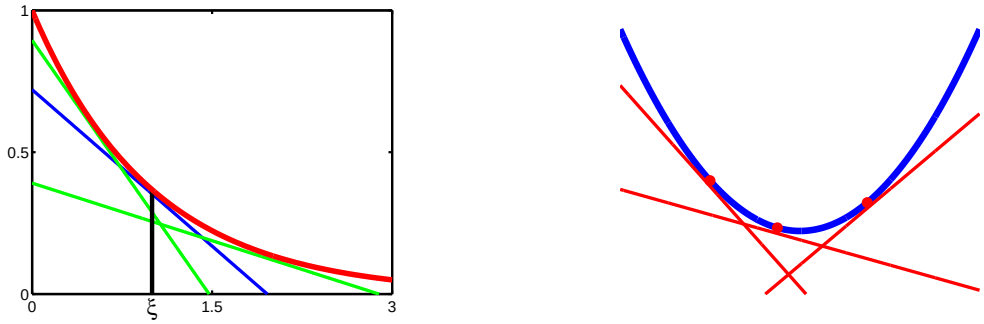
6.5.2 Example: exponential function

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

$$\mathcal{L}(x, \lambda) = \lambda x - f^\dagger(\lambda) \tag{6.84}$$

where the conjugate function is given by

$$f^\dagger(\lambda) = \max_x \lambda x - f(x) = -\lambda \log(-\lambda) + \lambda \tag{6.85}$$



(a) (b)

Figure 6.12: (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 6.12(a).

To see this, define

$$J(x, \lambda) = \lambda x - f(x) \tag{6.86}$$

We have

$$\frac{\partial J}{\partial x} = \lambda x - f'(x) = \lambda + e^{-x} \tag{6.87}$$

Setting the derivative to zero gives

$$x_\lambda = \arg\max_x J(x, \lambda) = -\log(-\lambda) \tag{6.88}$$

Hence

$$f^\dagger(\lambda) = J(x_\lambda, \lambda) = \lambda(-\log(-\lambda)) - e^{\log(-\lambda)} = -\lambda \log(-\lambda) + \lambda \tag{6.89}$$

6.5.3 Conjugate of a conjugate

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

$$f^{**}(\boldsymbol{x}) = \max_{\boldsymbol{\lambda}} \boldsymbol{\lambda}^\mathsf{T} \boldsymbol{x} - f^\dagger(\boldsymbol{\lambda}) \tag{6.90}$$

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

$$f^{**}(\boldsymbol{x}) = \max_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) \leq f(\boldsymbol{x}) \tag{6.91}$$

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 6.12(b).

Let us demonstrate this using the example from Section 6.5.2. We have

$$f^{**}(x) = \max_\lambda \lambda x - f^\dagger(\lambda) = \max_\lambda \lambda x + \lambda \log(-\lambda) - \lambda \tag{6.92}$$

Define

$$J^*(x, \lambda) = \lambda x - f^\dagger(x) = \lambda x + \lambda \log(-\lambda) - \lambda \tag{6.93}$$

We have

$$\frac{\partial}{\partial \lambda} J^*(x,\lambda) = x + \log(-\lambda) + \lambda\left(\frac{-1}{-\lambda}\right) - 1 = 0 \tag{6.94}$$

$$x = -\log(-\lambda) \tag{6.95}$$

$$\lambda_x = -e^{-x} \tag{6.96}$$

Substituting back we find

$$f^{**}(x) = J^*(x,\lambda_x) = (-e^{-x})x + (-e^{-x})(-x) - (-e^{-x}) = e^{-x} = f(x) \tag{6.97}$$

6.5.4 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, $\sigma(x) = \frac{1}{1+e^{-x}}$.

6.5.4.1 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

$$f(x) = \min_\eta \eta x - f^\dagger(\eta) \tag{6.98}$$

where

$$f^\dagger(\eta) = \min_x \eta x - f(x) \tag{6.99}$$

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

$$f^\dagger(\eta) = -\eta \ln \eta - (1-\eta)\ln(1-\eta) \tag{6.100}$$

which is the binary entropy function. Hence

$$\log \sigma(x) \leq \eta x - f^\dagger(\eta) \tag{6.101}$$

$$\sigma(x) \leq \exp(\eta x - f^\dagger(\eta)) \tag{6.102}$$

This exponential upper bound on $\sigma(x)$ is illustrated in Figure 6.13(a).

6.5.4.2 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

$$\log \sigma(x) = -\log(+e^{-x}) = -\log\left(e^{-x/2}(e^{x/2} + e^{-x/2})\right) \tag{6.103}$$

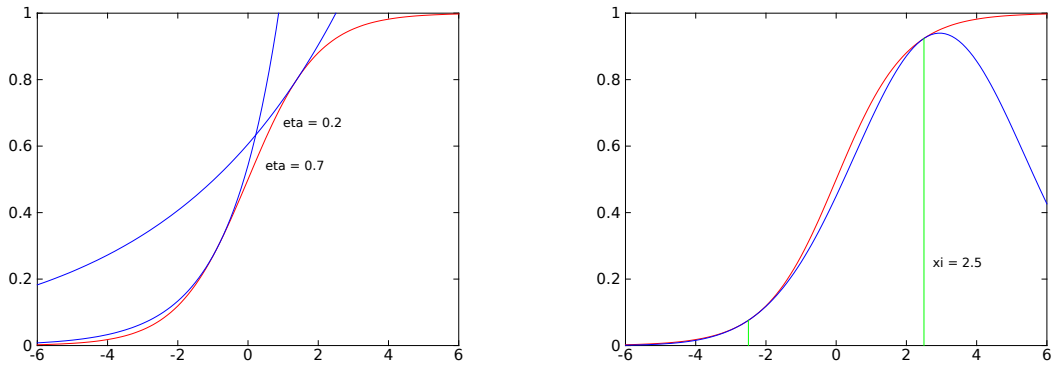
$$= x/2 - \log(e^{x/2} + e^{-x/2}) \tag{6.104}$$

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

$$f^\dagger(\eta) = \max_{x^2} \eta x^2 - f(\sqrt{x^2}) \tag{6.105}$$

We have

$$0 = \eta - \frac{dx}{dx^2}\frac{d}{dx} f(x) = \eta + \frac{1}{4x}\tanh(\frac{x}{2}) \tag{6.106}$$



(a) *(b)*

Figure 6.13: 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

$$\eta = -\frac{1}{4\xi}\tanh(\frac{\xi}{2}) = -\frac{1}{2\xi}\left[\sigma(\xi) - \frac{1}{2}\right] = -\lambda(\xi) \tag{6.107}$$

The conjugate function can be rewritten as

$$f^\dagger(\lambda(\xi)) = -\lambda(\xi)\xi^2 - f(\xi) = \lambda(\xi)\xi^2 + \log(e^{\xi/2} + e^{-\xi/2}) \tag{6.108}$$

So the lower bound on f becomes

$$f(x) \geq -\lambda(\xi)x^2 - g(\lambda(\xi)) = -\lambda(\xi)x^2 + \lambda(\xi)\xi^2 - \log(e^{\xi/2} + e^{-\xi/2}) \tag{6.109}$$

and the lower bound on the sigmoid function becomes

$$\sigma(x) \geq \sigma(\xi)\exp\left[(x-\xi)/2 - \lambda(\xi)(x^2 - \xi^2)\right] \tag{6.110}$$

This is illustrated in Figure 6.13(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

$$\mathbb{E}\left[\tilde{\boldsymbol{f}}(\boldsymbol{z})\right] \approx \begin{pmatrix}\boldsymbol{\mu}\\ \boldsymbol{f}(\boldsymbol{\mu})\end{pmatrix} \tag{8.34}$$

$$\mathrm{Cov}\left[\tilde{\boldsymbol{f}}(\boldsymbol{z})\right] \approx \begin{pmatrix}\mathbf{I}\\ \mathbf{F}(\boldsymbol{\mu})\end{pmatrix}\boldsymbol{\Sigma}\begin{pmatrix}\mathbf{I}\\ \mathbf{F}(\boldsymbol{\mu})\end{pmatrix}^\mathsf{T} = \begin{pmatrix}\boldsymbol{\Sigma} & \boldsymbol{\Sigma}\mathbf{F}(\boldsymbol{\mu})^\mathsf{T}\\ \mathbf{F}(\boldsymbol{\mu})\boldsymbol{\Sigma} & \mathbf{F}(\boldsymbol{\mu})\boldsymbol{\Sigma}\mathbf{F}(\boldsymbol{\mu})^\mathsf{T}\end{pmatrix} \tag{8.35}$$

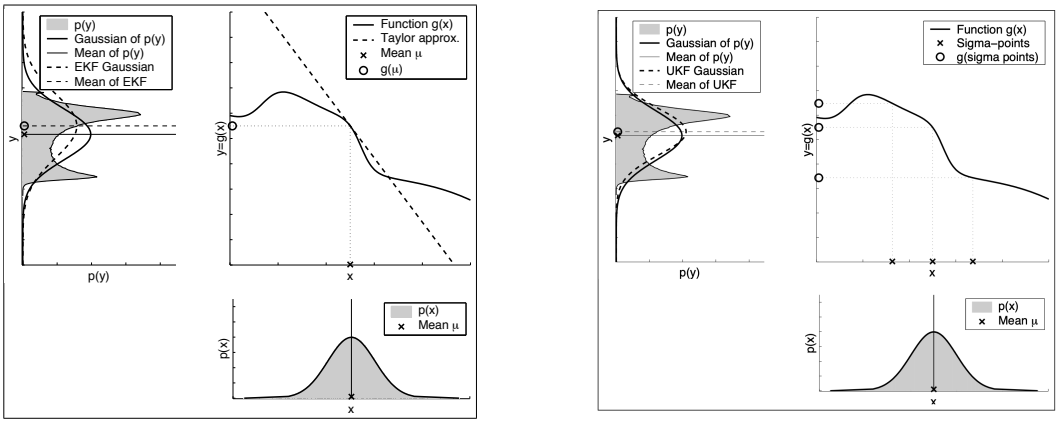
When deriving the EKF, we need the following slightly more general version:

$$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}),\ \boldsymbol{y} = \boldsymbol{f}(\boldsymbol{z}) + \boldsymbol{q},\ \boldsymbol{q} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}) \tag{8.36}$$

The resulting linear approximation to the joint is

$$\begin{aligned}
\begin{pmatrix}\boldsymbol{z}\\ \boldsymbol{y}\end{pmatrix} &\sim \mathcal{N}\left(\begin{pmatrix}\boldsymbol{\mu}\\ \boldsymbol{\mu}_L\end{pmatrix}, \begin{pmatrix}\boldsymbol{\Sigma} & \mathbf{C}_L\\ \mathbf{C}_L^\mathsf{T} & \mathbf{S}_L\end{pmatrix}\right) && (8.37)\\
\boldsymbol{\mu}_L &= \boldsymbol{f}(\boldsymbol{\mu}) && (8.38)\\
\mathbf{S}_L &= \mathbf{F}(\boldsymbol{\mu})\boldsymbol{\Sigma}\mathbf{F}(\boldsymbol{\mu})^\mathsf{T} + \mathbf{Q} && (8.39)\\
\mathbf{C}_L &= \boldsymbol{\Sigma}\mathbf{F}(\boldsymbol{\mu})^\mathsf{T} && (8.40)
\end{aligned}$$

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 **??** 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.

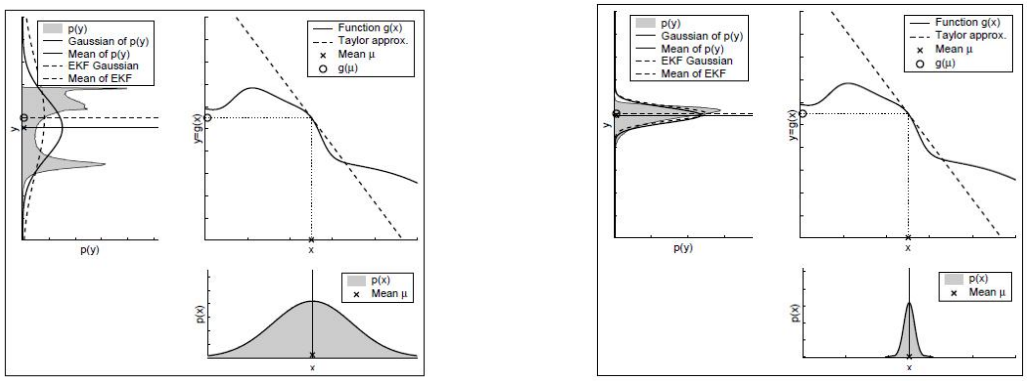
8.2.2.3 Accuracy

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**.

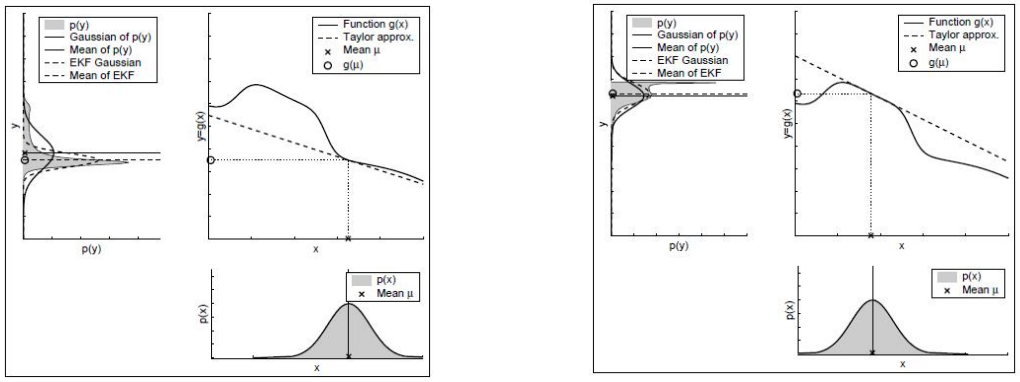
8.2.2.4 Diagonal approximation

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

$$\mathcal{T}(\boldsymbol{y}) = [\mathbb{I}(y = 1), \ldots, \mathbb{I}(y = C - 1)] \tag{8.84}$$

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

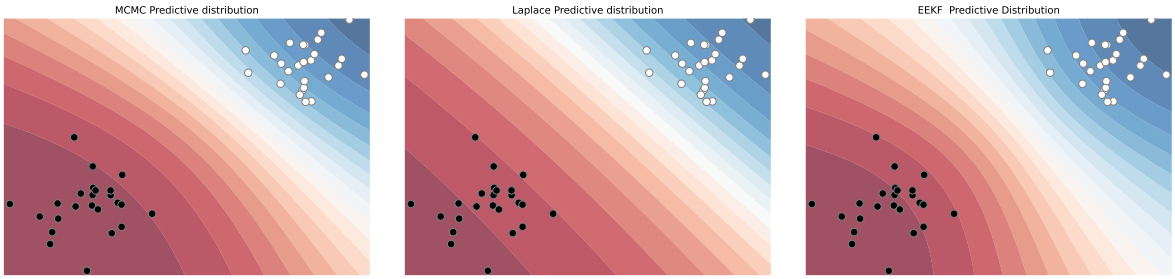
$$R_{ij} = \mathrm{diag}(p_i) - p_i p_j \tag{8.85}$$

Now consider the simpler case where we have two class labels, so $C = 2$. In this case, $\mathcal{T}(\boldsymbol{y}) = \mathbb{I}(\boldsymbol{y} = 1)$, 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:

$$p(y_t|\boldsymbol{z}_t, \boldsymbol{u}_t) = \mathrm{Ber}(y_t|\sigma(\boldsymbol{z}_t^\mathsf{T} \boldsymbol{u}_t)) \tag{8.86}$$

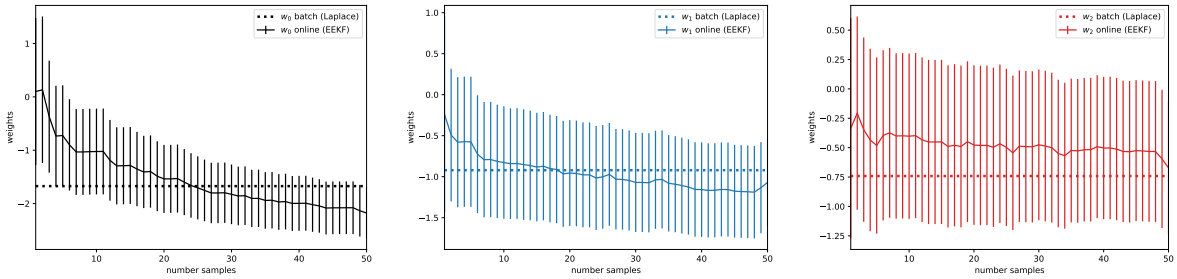
We can use the exponential family representation of the output discussed in Section 8.2.4.3.

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.

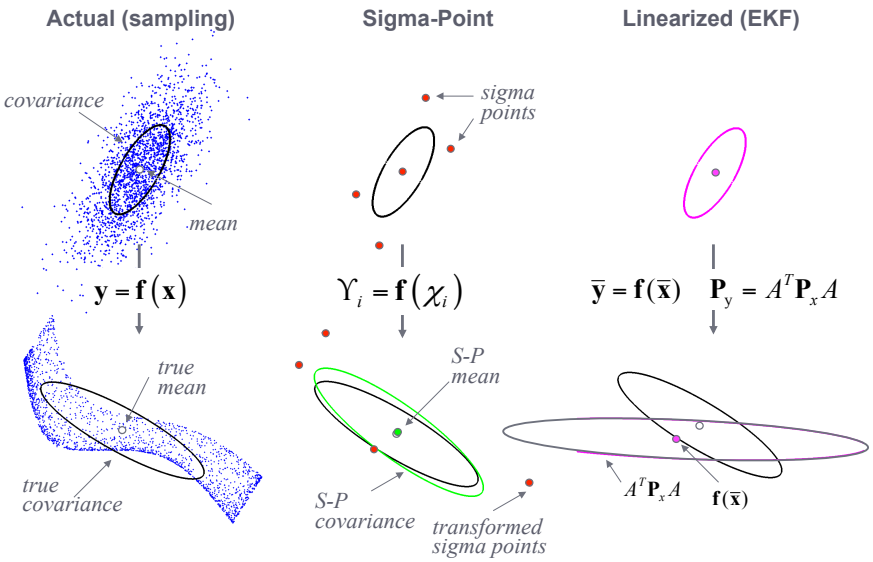


Figure 8.6: An example of the unscented transform in two dimensions. From [WM01]. Used with kind permission of Eric Wan.

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:

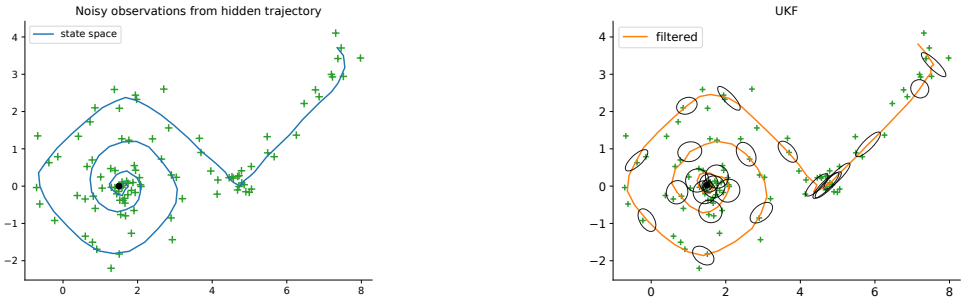
$$\hat{\boldsymbol{z}}_{t,i} = \boldsymbol{f}(\boldsymbol{z}_{t,i}) \tag{8.110}$$

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:

$$q(\boldsymbol{z}|\boldsymbol{y}) = \mathcal{N}\left(\boldsymbol{z}| \underbrace{\boldsymbol{\mu}_z + \boldsymbol{\Sigma}_{zy}\boldsymbol{\Sigma}_y^{-1}(\boldsymbol{y} - \boldsymbol{\mu}_y)}_{\boldsymbol{\mu}_{z|y}}, \underbrace{\boldsymbol{\Sigma}_z - \boldsymbol{\Sigma}_{zy}\boldsymbol{\Sigma}_y^{-1}\boldsymbol{\Sigma}_{zy}^{\mathsf{T}}}_{\boldsymbol{\Sigma}_{z|y}}\right) \tag{8.134}$$

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 $\boldsymbol{\Sigma}_{z|x}$ 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**.

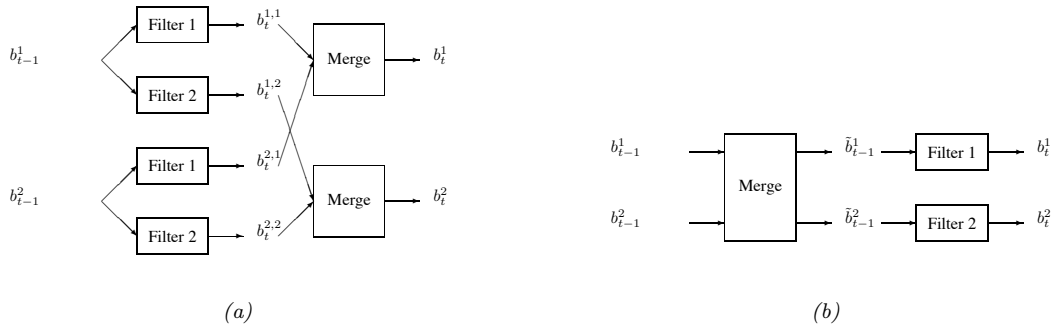


Figure 8.8: ADF for a switching linear dynamical system with 2 discrete states. (a) GPB2 method. (b) IMM method.

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.

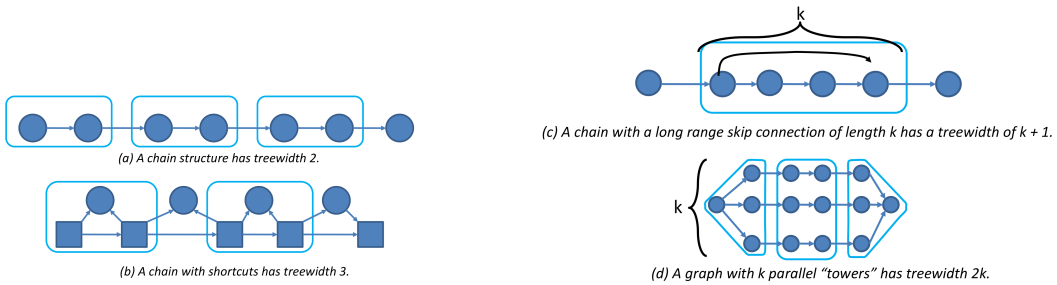


Figure 8.9: Examples of optimal tree decompositions for some common graph structures. Adapted from `https://bit.ly/2m5vauG`.

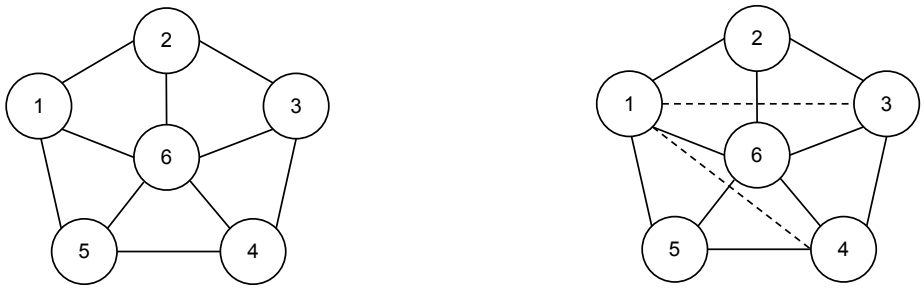


Figure 8.10: A triangulated graph is not just composed of little triangles. Left: this graph is not triangulated, despite appearances, since it contains a chordless 5-cycle 1-2-3-4-5-1. Right: one possible triangulation, by adding the 1-3 and 1-4 fill-in edges. Adapted from [Arm05, p46]

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

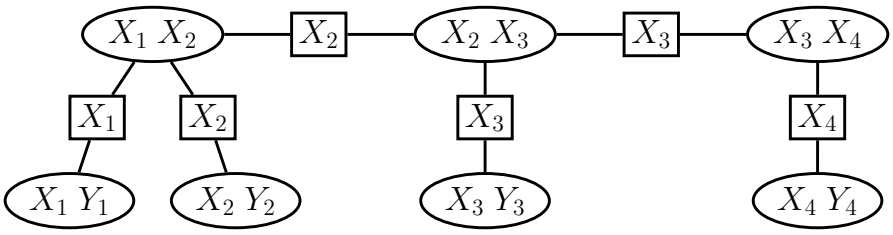


Figure 8.11: The junction tree derived from an HMM of length $T = 4$.

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

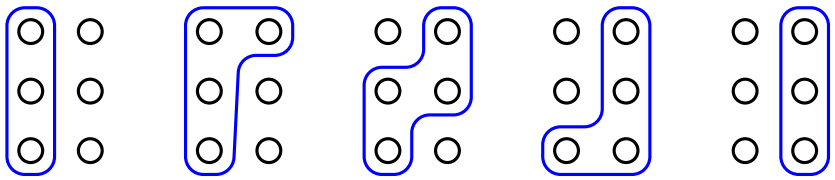


Figure 8.12: The cliques in the junction tree decomposition as we advance the frontier from one time-slice to the next in a 3-chain factorial HMM model.

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 **??**, 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

$$\mathbb{M}(G) = \text{conv}\{(0,0,0), (1,0,0), (0,1,0), (1,1,1)\} \tag{8.177}$$

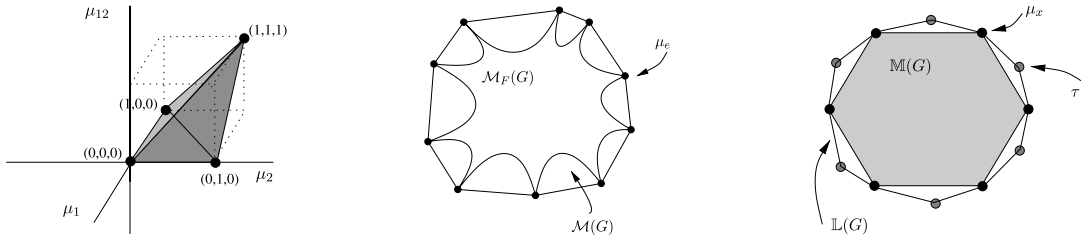
These are the four black dots in Figure 8.13(a). We see that the convex hull defines the same volume as the intersection of half-spaces.

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.



(a) (b) (c)

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

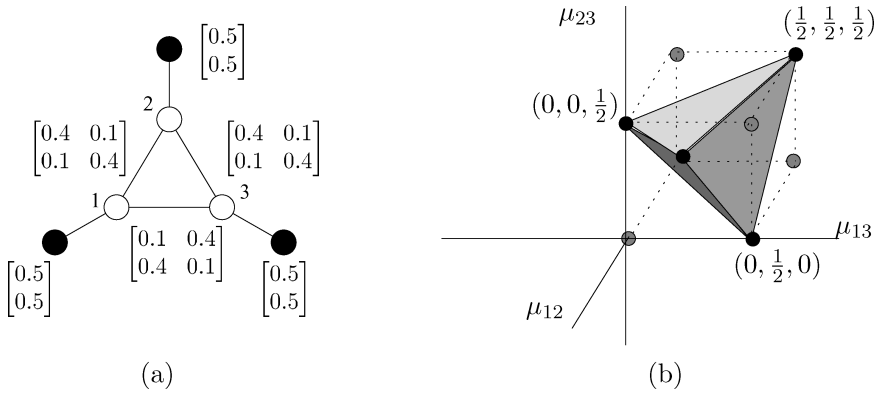


Figure 8.14: (a) Illustration of pairwise UGM on binary nodes, together with a set of pseudo marginals that are not globally consistent. (b) A slice of the marginal polytope illustrating the set of feasible edge marginals, assuming the node marginals are clamped at $\mu_1 = \mu_2 = \mu_3 = 0.5$. From 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.

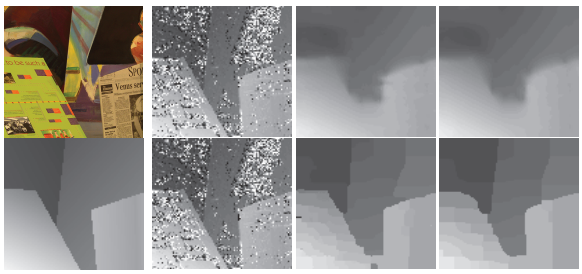


Figure 8.15: Illustration of belief propagation for stereo depth estimation applied to the Venus image from the Middlebury stereo benchmark dataset [SS02]. Left column: image and true disparities. Remaining columns: initial estimate, estimate after 1 iteration, and estimate at convergence. Top row: Gaussian 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:

$$\text{cost}(\mathcal{X}_s, \mathcal{X}_t) = \sum_{x_u \in \mathcal{X}_s, x_v \in \mathcal{X}_t} \text{cost}(x_u, x_v) \tag{8.186}$$

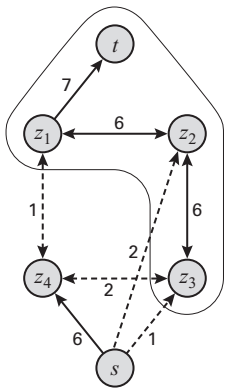


Figure 8.16: Illustration of graphcuts applied to an MRF with 4 nodes. Dashed lines are ones which contribute to the cost of the cut (for bidirected edges, we only count one of the costs). Here the min cut has cost 6. From Figure 13.5 from [KF09]. Used with kind permission of Daphne Koller.

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].

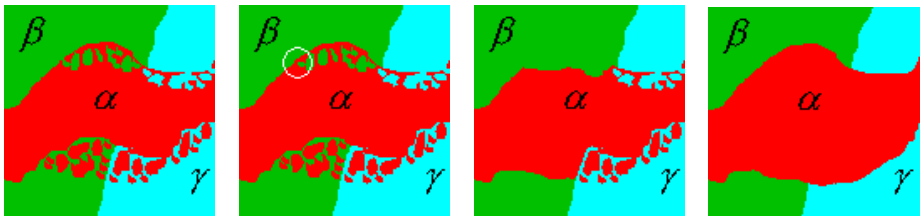
8.7.4.3 Graphcuts for nonbinary metric MRFs

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



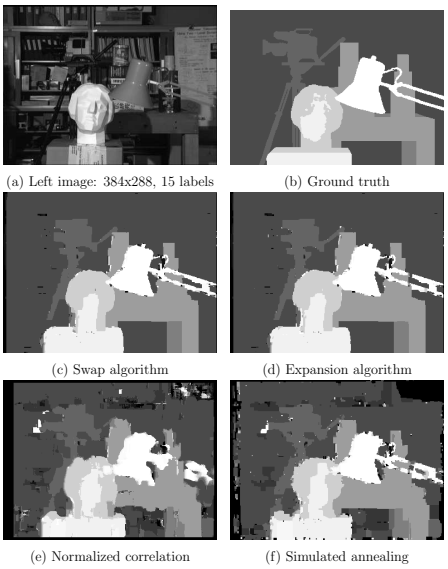
(a) initial labeling (b) standard move (c) α-β-swap (d) α-expansion

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

9.1 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

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) = \frac{1}{Z} \exp\left\{ \sum_{s \in \mathcal{V}} \theta_s(z_s) + \sum_{(s,t) \in \mathcal{E}} \theta_{st}(z_s, z_t) \right\} \tag{9.1}$$

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:

$$\mathcal{L}(q) = \mathbb{E}_{q(\boldsymbol{z})}[\log \tilde{p}(\boldsymbol{z})] + \mathbb{H}(q) \leq \log Z \tag{9.2}$$

Let $\boldsymbol{\mu} = \mathbb{E}_q[\mathcal{T}(\boldsymbol{z})]$ be the mean parameters of the variational distribution. Then we can rewrite this as

$$\mathcal{L}(\boldsymbol{\mu}) = \boldsymbol{\theta}^\mathsf{T}\boldsymbol{\mu} + \mathbb{H}(\boldsymbol{\mu}) \leq \log Z \tag{9.3}$$

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

$$\max_{\boldsymbol{\mu} \in \mathbb{M}(G)} \boldsymbol{\theta}^T\boldsymbol{\mu} + \mathbb{H}(\boldsymbol{\mu}) = \log Z \tag{9.4}$$

Equation (9.4) 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	$\max_{\boldsymbol{\mu} \in \mathbb{M}_F(G)} \boldsymbol{\theta}^T \boldsymbol{\mu} + \mathbb{H}_{\text{MF}}(\boldsymbol{\mu}) \leq \log Z$	Concave	Nonconvex inner approx.	Section 9.1.2
Loopy BP	$\max_{\boldsymbol{\tau} \in \mathbb{L}(G)} \boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{\text{Bethe}}(\boldsymbol{\tau}) \approx \log Z$	Non-concave	Convex outer approx.	Section 9.1.3
TRBP	$\max_{\boldsymbol{\tau} \in \mathbb{L}(G)} \boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{\text{TRBP}}(\boldsymbol{\tau}) \geq \log Z$	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

$$\mathbb{H}_{\text{MF}}(\boldsymbol{\mu}) = \sum_s \mathbb{H}(\boldsymbol{\mu}_s) \tag{9.5}$$

which follows from the factorization assumption. Thus the mean field objective is

$$\mathcal{L}_{\text{MF}}(\boldsymbol{\mu}) = \boldsymbol{\theta}^{\mathsf{T}} \boldsymbol{\mu} + \mathbb{H}_{\text{MF}}(\boldsymbol{\mu}) \leq \log Z \tag{9.6}$$

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:

$$\Omega(F) \triangleq \{\boldsymbol{\theta} \in \Omega : \boldsymbol{\theta}_\alpha = 0 \; \forall \alpha \in \mathcal{I} \setminus \mathcal{I}(F)\} \tag{9.7}$$

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

$$\Omega(F_0) \triangleq \{\boldsymbol{\theta} \in \Omega : \boldsymbol{\theta}_{st} = 0 \; \forall (s,t) \in E\} \tag{9.8}$$

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:

$$\mathbb{M}_F(G) \triangleq \{\boldsymbol{\mu} \in \mathbb{R}^d : \boldsymbol{\mu} = \mathbb{E}_{\boldsymbol{\theta}}[\mathcal{T}(\boldsymbol{z})] \quad \text{for some } \boldsymbol{\theta} \in \Omega(F)\} \tag{9.9}$$

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

$$\max_{\boldsymbol{\mu} \in \mathbb{M}_F(G)} \boldsymbol{\theta}^{\mathsf{T}} \boldsymbol{\mu} + \mathbb{H}_{\text{MF}}(\boldsymbol{\mu}) \tag{9.10}$$

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:

$$p_{\boldsymbol{\mu}}(\boldsymbol{x}) = \prod_{s \in V} \mu_s(x_s) \prod_{(s,t) \in E} \frac{\mu_{st}(x_s, x_t)}{\mu_s(x_s)\mu_t(x_t)} \tag{9.11}$$

This satisfies the normalization and pairwise marginalization constraints of the outer approximation by construction.

From Equation 9.11, we can write the exact entropy of any tree structured distribution $\boldsymbol{\mu} \in \mathbb{M}(T)$ as follows:

$$\mathbb{H}(\boldsymbol{\mu}) = \sum_{s \in V} H_s(\mu_s) - \sum_{(s,t) \in E} I_{st}(\mu_{st}) \tag{9.12}$$

$$H_s(\mu_s) = -\sum_{x_s \in \mathcal{X}_s} \mu_s(x_s) \log \mu_s(x_s) \tag{9.13}$$

$$I_{st}(\mu_{st}) = \sum_{(x_s, x_t) \in \mathcal{X}_s \times \mathcal{X}_t} \mu_{st}(x_s, x_t) \log \frac{\mu_{st}(x_s, x_t)}{\mu_s(x_s)\mu_t(x_t)} \tag{9.14}$$

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:

$$\mathbb{H}(\boldsymbol{\mu}) = -\sum_{s \in V} (d_s - 1) H_s(\mu_s) + \sum_{(s,t) \in E} H_{st}(\mu_{st}) \tag{9.15}$$

where d_s is the degree (number of neighbors) for node s.

The **Bethe**[1] approximation to the entropy is simply the use of Equation 9.12 even when we don't have a tree:

$$\mathbb{H}_{\text{Bethe}}(\boldsymbol{\tau}) = \sum_{s \in V} H_s(\tau_s) - \sum_{(s,t) \in E} I_{st}(\tau_{st}) \tag{9.16}$$

We define the **Bethe free energy** as the expected energy minus approximate entropy:

$$\mathcal{F}_{\text{Bethe}}(\boldsymbol{\tau}) \triangleq -\left[\boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{\text{Bethe}}(\boldsymbol{\tau})\right] \approx -\log Z \tag{9.17}$$

Thus our final objective becomes

$$\max_{\boldsymbol{\tau} \in \mathbb{L}(G)} \boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{\text{Bethe}}(\boldsymbol{\tau}) \tag{9.18}$$

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.

[1] 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

$$\begin{aligned}\mathcal{L}(\boldsymbol{\tau}, \boldsymbol{\lambda}; \boldsymbol{\theta}) \triangleq \boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{\text{Bethe}}(\boldsymbol{\tau}) + \sum_s \lambda_{ss} C_{ss}(\boldsymbol{\tau}) \\ + \sum_{s,t} \left[\sum_{x_s} \lambda_{ts}(x_s) C_{ts}(x_s; \boldsymbol{\tau}) + \sum_{x_t} \lambda_{st}(x_t) C_{st}(x_t; \boldsymbol{\tau}) \right] \end{aligned} \tag{9.19}$$

(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

$$\log \tau_s(x_s) = \lambda_{ss} + \theta_s(x_s) + \sum_{t \in \text{nbr}(s)} \lambda_{ts}(x_s) \tag{9.20}$$

$$\log \frac{\tau_{st}(x_s, x_t)}{\tilde{\tau}_s(x_s) \tilde{\tau}_t(x_t)} = \theta_{st}(x_s, x_t) - \lambda_{ts}(x_s) - \lambda_{st}(x_t) \tag{9.21}$$

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

$$\begin{aligned}\log \tau_{st}(x_s, x_t) = \lambda_{ss} + \lambda_{tt} + \theta_{st}(x_s, x_t) + \theta_s(x_s) + \theta_t(x_t) \\ + \sum_{u \in \text{nbr}(s) \setminus t} \lambda_{us}(x_s) + \sum_{u \in \text{nbr}(t) \setminus s} \lambda_{ut}(x_t) \end{aligned} \tag{9.22}$$

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:

$$\tau_s(x_s) \propto \exp(\theta_s(x_s)) \prod_{t \in \text{nbr}(s)} m_{t \to s}(x_s) \tag{9.23}$$

$$\begin{aligned}\tau_{st}(x_s, x_t) \propto \exp\left(\theta_{st}(x_s, x_t) + \theta_s(x_s) + \theta_t(x_t)\right) \\ \times \prod_{u \in \text{nbr}(s) \setminus t} m_{u \to s}(x_s) \prod_{u \in \text{nbr}(t) \setminus s} m_{u \to t}(x_t) \end{aligned} \tag{9.24}$$

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

$$m_{t \to s}(x_s) \propto \sum_{x_t} \left[\exp\left\{\theta_{st}(x_s, x_t) + \theta_t(x_t)\right\} \prod_{u \in \text{nbr}(t) \setminus s} m_{u \to t}(x_t) \right] \tag{9.25}$$

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

$$\sum_{x_1,x_2} \tau_{1245}(x_1,x_2,x_4,x_5) = \tau_{45}(x_4,x_5), \quad \sum_{x_6} \tau_{56}(x_5,x_6) = \tau_5(x_5), \ldots \tag{9.26}$$

Furthermore, we approximate the entropy as follows:

$$\mathbb{H}_{\text{Kikuchi}}(\boldsymbol{\tau}) \triangleq \sum_{g \in E} c(g) H_g(\tau_g) \tag{9.27}$$

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

$$\mathbb{H}_{\text{Kikuchi}}(\boldsymbol{\tau}) = -[H_{1245} + H_{2356} + H_{4578} + H_{5689}] - [H_{25} + H_{45} + H_{56} + H_{58}] + H_5 \tag{9.28}$$

Putting these two approximations together, we can define the **Kikuchi free energy**[2] as follows:

$$\mathcal{F}_{\text{Kikuchi}}(\boldsymbol{\tau}) \triangleq -\left[\boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{\text{Kikuchi}}(\boldsymbol{\tau})\right] \approx -\log Z \tag{9.29}$$

Our variational problem becomes

$$\max_{\boldsymbol{\tau} \in \mathbb{L}(G)} \boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{\text{Kikuchi}}(\boldsymbol{\tau}) \tag{9.30}$$

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 (9.4) 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:

$$\mathbb{H}(\boldsymbol{\mu}(G)) \leq \sum_{F \in \mathcal{F}} \rho(F) \mathbb{H}(\boldsymbol{\mu}(F)) \triangleq \mathbb{H}(\boldsymbol{\mu}, \rho) \tag{9.31}$$

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:

$$\mathbb{L}(G; \mathcal{F}) \triangleq \{\boldsymbol{\tau} \in \mathbb{R}^d : \boldsymbol{\tau}(F) \in \mathbb{M}(F) \; \forall F \in \mathcal{F}\} \tag{9.32}$$

This is a convex set since each $\mathbb{M}(F)$ is a projection of a convex set. Hence we define our problem as

$$\max_{\boldsymbol{\tau} \in \mathbb{L}(G;\mathcal{F})} \boldsymbol{\tau}^T \boldsymbol{\theta} + \mathbb{H}(\boldsymbol{\tau}, \rho) \tag{9.33}$$

[2] Ryoichi Kikuchi is a Japanese physicist.

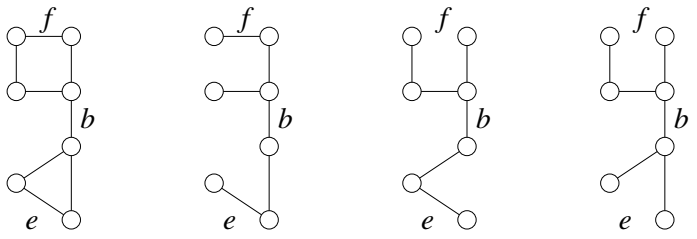


Figure 9.1: (a) A graph. (b-d) Some of its spanning trees. From Figure 7.1 of [WJ08]. Used with kind permission of Martin Wainwright.

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 9.12. 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:

$$\mathbb{H}(\boldsymbol{\mu}) \leq \sum_{s \in V} H_s(\mu_s) - \sum_{(s,t) \in E} \rho_{st} I_{st}(\mu_{st}) \triangleq \mathbb{H}_{\text{TRBP}}(\boldsymbol{\mu}) \tag{9.34}$$

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:

$$\max_{\boldsymbol{\tau} \in \mathbb{L}(G)} \boldsymbol{\tau}^T \boldsymbol{\theta} + \mathbb{H}_{\text{TRBP}}(\boldsymbol{\tau}) \geq \log Z \tag{9.35}$$

9.1.5.2 Message passing implementation

The simplest way to minimize Equation (9.35) 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]:

$$m_{t\to s}(x_s) \propto \sum_{x_t} \exp\left(\frac{1}{\rho_{st}}\theta_{st}(x_s,x_t) + \theta_t(x_t)\right) \frac{\prod_{v\in \text{nbr}(t)\setminus s}[m_{v\to t}(x_t)]^{\rho_{vt}}}{[m_{s\to t}(x_t)]^{1-\rho_{ts}}} \tag{9.36}$$

At convergence, the node and edge pseudo marginals are given by

$$\tau_s(x_s) \propto \exp(\theta_s(x_s)) \prod_{v\in \text{nbr}(s)} [m_{v\to s}(x_s)]^{\rho_{vs}} \tag{9.37}$$

$$\tau_{st}(x_s,x_t) \propto \varphi_{st}(x_s,x_t) \frac{\prod_{v\in \text{nbr}(s)\setminus t}[m_{v\to s}(x_s)]^{\rho_{vs}}}{[m_{t\to s}(x_s)]^{1-\rho_{st}}} \frac{\prod_{v\in \text{nbr}(t)\setminus s}[m_{v\to t}(x_t)]^{\rho_{vt}}}{[m_{s\to t}(x_t)]^{1-\rho_{ts}}} \tag{9.38}$$

$$\varphi_{st}(x_s,x_t) \triangleq \exp\left(\frac{1}{\rho_{st}}\theta_{st}(x_s,x_t) + \theta_s(x_s) + \theta_t(x_t)\right) \tag{9.39}$$

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 (9.36) (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 [Sch10b] 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 6.5.4.2 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:

$$Ł(\boldsymbol{\psi}) \triangleq -D_{\mathbb{KL}}\left(q(\boldsymbol{w}) \parallel p(\boldsymbol{w})\right) + \mathbb{E}_q\left[\sum_{i=1}^N \log p(y_i|\boldsymbol{x}_i, \boldsymbol{w})\right] \tag{14.30}$$

$$= -D_{\mathbb{KL}}\left(q(\boldsymbol{w}) \parallel p(\boldsymbol{w})\right) + \mathbb{E}_q\left[\sum_{i=1}^N \boldsymbol{y}_i^\mathsf{T}\boldsymbol{\eta}_i - \mathrm{lse}(\boldsymbol{\eta}_i)\right] \tag{14.31}$$

$$= -D_{\mathbb{KL}}\left(q(\boldsymbol{w}) \parallel p(\boldsymbol{w})\right) + \sum_{i=1}^N \boldsymbol{y}_i^\mathsf{T}\mathbb{E}_q\left[\boldsymbol{\eta}_i\right] - \sum_{i=1}^N \mathbb{E}_q\left[\mathrm{lse}(\boldsymbol{\eta}_i)\right] \tag{14.32}$$

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

$$\begin{aligned}-D_{\mathbb{KL}}\left(\mathcal{N}(\boldsymbol{m}_N, \mathbf{V}_N) \parallel \mathcal{N}(\boldsymbol{m}_0, \mathbf{V}_0)\right) = -\frac{1}{2}\big[&\mathrm{tr}(\mathbf{V}_N\mathbf{V}_0^{-1}) - \log|\mathbf{V}_N\mathbf{V}_0^{-1}| \\ &+(\boldsymbol{m}_N - \boldsymbol{m}_0)^\mathsf{T}\mathbf{V}_0^{-1}(\boldsymbol{m}_N - \boldsymbol{m}_0) - DM\big]\end{aligned} \tag{14.33}$$

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:

$$-\sum_{i=1}^N \mathbb{E}_q\left[\mathrm{lse}(\boldsymbol{\eta}_i)\right] \geq -\frac{1}{2}\mathrm{tr}(\mathbf{A}_i\tilde{\mathbf{V}}_i) - \frac{1}{2}\tilde{\boldsymbol{m}}_i\mathbf{A}_i\tilde{\boldsymbol{m}}_i + \boldsymbol{b}_i^\mathsf{T}\tilde{\boldsymbol{m}}_i - c_i \tag{14.35}$$

where $\tilde{\mathbf{V}}_i \triangleq \mathbf{X}_i\mathbf{V}_N\mathbf{X}_i^\mathsf{T}$. Hence we have

$$\begin{aligned}Ł(\boldsymbol{\psi}) \geq &-\frac{1}{2}\left[\mathrm{tr}(\mathbf{V}_N\mathbf{V}_0^{-1}) - \log|\mathbf{V}_N\mathbf{V}_0^{-1}| + (\boldsymbol{m}_N - \boldsymbol{m}_0)^\mathsf{T}\mathbf{V}_0^{-1}(\boldsymbol{m}_N - \boldsymbol{m}_0)\right] \\ &- \frac{1}{2}DM + \sum_{i=1}^N \boldsymbol{y}_i^\mathsf{T}\tilde{\boldsymbol{m}}_i - \frac{1}{2}\mathrm{tr}(\mathbf{A}_i\tilde{\mathbf{V}}_i) - \frac{1}{2}\tilde{\boldsymbol{m}}_i\mathbf{A}_i\tilde{\boldsymbol{m}}_i + \boldsymbol{b}_i^\mathsf{T}\tilde{\boldsymbol{m}}_i - c_i\end{aligned} \tag{14.36}$$

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 6.5.4.2. 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.

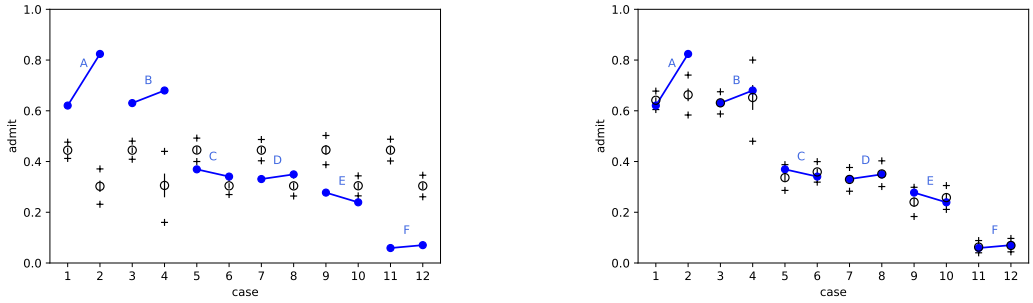
We can perform posterior inference using a variety of methods (see **??**). Here we use HMC (**??**). We find the 89% credible interval for α_1 is $[-0.29, 0.16]$ and for α_2 is $[-0.91, 0.75]$.[1] The corresponding distribution for the difference in probability, $\sigma(\alpha_1) - \sigma(\alpha_2)$, 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 **??**). 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:

$$A_i \sim \text{BetaBinom}(N_i, \pi_i, \kappa) \tag{14.75}$$

$$\text{logit}(\pi_i) = \alpha_{\text{GENDER}[i]} \tag{14.76}$$

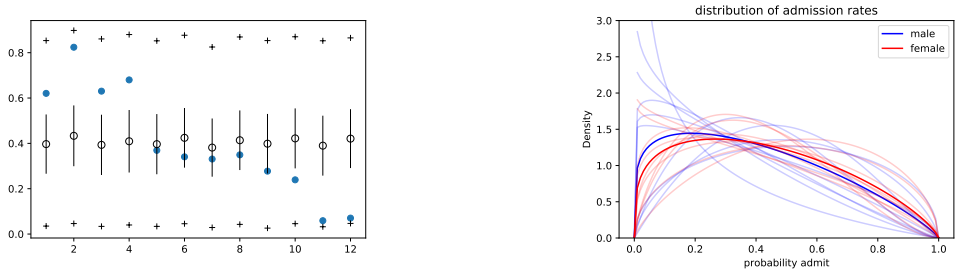
$$\alpha_j \sim \mathcal{N}(0, 1.5) \tag{14.77}$$

$$\kappa = \phi + 2 \tag{14.78}$$

$$\phi \sim \text{Expon}(1) \tag{14.79}$$

(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

$$\boldsymbol{w}_j = \boldsymbol{u} + \sigma \mathbf{L} \boldsymbol{z}_j \tag{14.98}$$

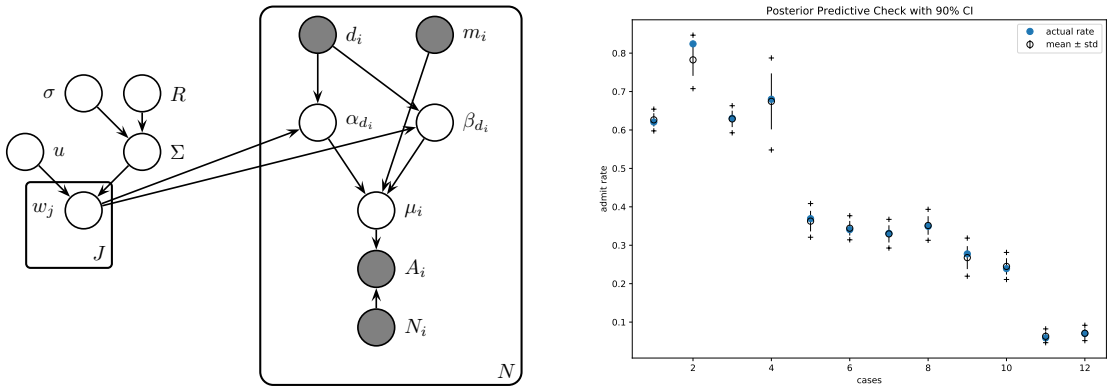
where $\mathbf{L} = \text{chol}(\mathbf{R})$ is the Cholesy factor for the correlation matrix $\mathbf{R}$, and $\boldsymbol{z}_j \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_2)$. Thus the model becomes the following:[3].

$$\boldsymbol{z}_j \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_2) \tag{14.99}$$
$$\boldsymbol{v}_j = \text{diag}(\boldsymbol{\sigma}) \mathbf{L} \boldsymbol{z}_j \tag{14.100}$$
$$\boldsymbol{u} \sim \mathcal{N}(\mathbf{0}, \text{diag}(4, 1)) \tag{14.101}$$
$$\log(\mu_i) = u[0] + v[\text{DEPT}[i], 0] + (u[1] + v[\text{DEPT}[i], 1]) \times \text{MALE}[i] \tag{14.102}$$

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 variable models

27.1 Topic models

In this section, we show how to modify the multinomial PCA model of **??** to handle variable-sized observations, such as text documents. The basic idea is to assume each observation $\boldsymbol{x}_l$ is generated independently of all the others conditional on the shared latent factors $\boldsymbol{z}$; the likelihood is categorical, with shared parameters, and the prior is Dirichlet distribution. This is called a **topic model**, since the $\boldsymbol{z}$ variables can be interpreted as a mixture of different topics that are present in document $\boldsymbol{x}$.

27.1.1 Latent Dirichlet Allocation (LDA)

The most common kind of topic model is called **latent Dirichlet allocation** (**LDA**) [BNJ03; Ble12; BGHM17]. (This usage of the term "LDA" is not to be confused with linear discriminant analysis.)

27.1.1.1 Model definition

We can define the LDA model as follows. Let $x_{nl} \in \{1, \ldots, V\}$ be the identity of the l'th word in document n, where l can now range from 1 to L_n, the length of the document, and V is the size of the vocabulary. The probability of word v at location l is given by

$$p(x_{nl} = v|\boldsymbol{z}_n) = \sum_k z_{nk} w_{kv} \tag{27.1}$$

where $0 \leq z_{nk} \leq 1$ is the proportion of "topic" k in document n, and $\boldsymbol{z}_n \sim \text{Dir}(\boldsymbol{\alpha})$.

We can rewrite this model by associating a discrete latent variable $c_{nl} \in \{1, \ldots, N_z\}$ with each word in each document, with distribution $p(c_{nl}|\boldsymbol{z}_n) = \text{Cat}(c_{nl}|\boldsymbol{z}_n)$. Thus c_{nl} specifies the topic to use for word l in document n. The full joint model becomes

$$p(\boldsymbol{x}_n, \boldsymbol{z}_n, \boldsymbol{c}_n) = \text{Dir}(\boldsymbol{z}_n|\boldsymbol{\alpha}) \prod_{l=1}^{L_n} \text{Cat}(c_{nl}|\boldsymbol{z}_n)\text{Cat}(x_{nl}|\mathbf{W}[c_{nl}, :]) \tag{27.2}$$

where $\mathbf{W}[k, :] = \boldsymbol{w}_k$ is the distribution over words for the k'th topic. See Figure 27.1 for the corresponding PGM-D.

We typically use a Dirichlet prior the topic parameters, $p(\boldsymbol{w}_k) = \text{Dir}(\boldsymbol{w}_k|\beta \mathbf{1}_V)$; by setting β small enough, we can encourage these topics to be sparse, so that each topic only predicts a subset of the words. In addition, we use a Dirichlet prior on the latent factors, $p(\boldsymbol{z}_n) = \text{Dir}(\boldsymbol{z}_n|\alpha \mathbf{1}_{N_z})$. If we set α small enough, we can encourage the topic distribution for each document to be sparse, so that each document only contains a subset of the topics. See Figure 27.2 for an illustration.

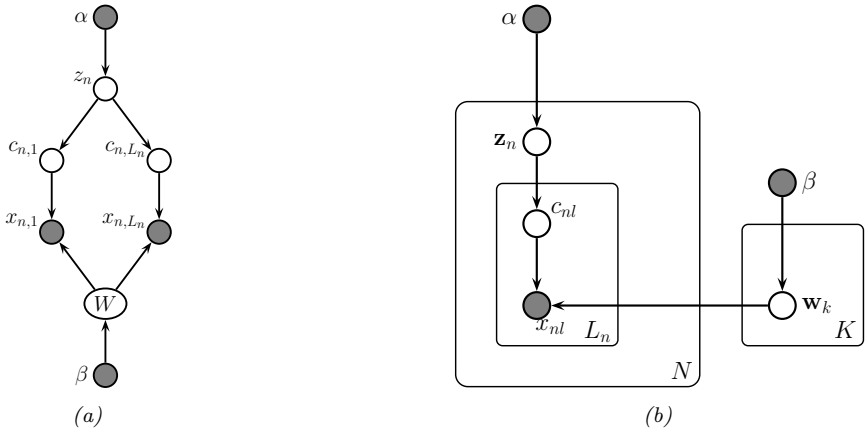


Figure 27.1: Latent Dirichlet Allocation (LDA) as a PGM-D. (a) Unrolled form. (b) Plate form.

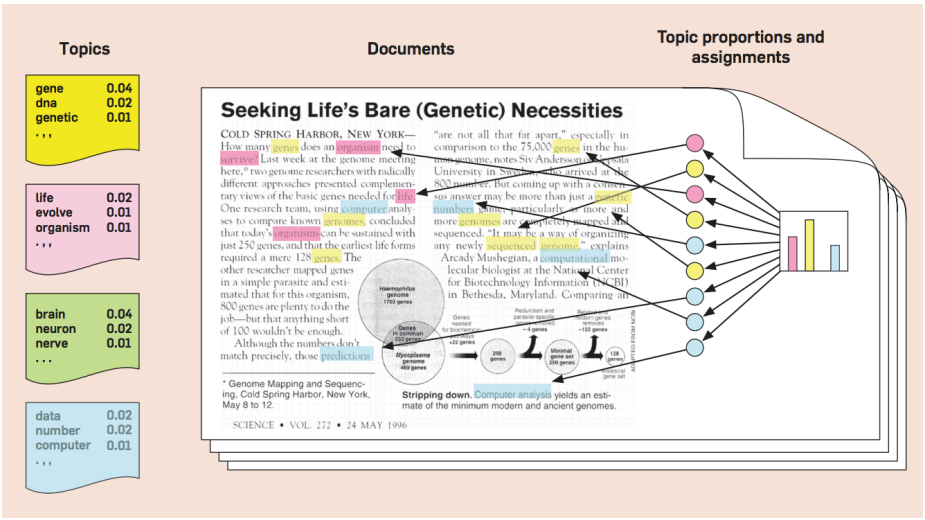


Figure 27.2: Illustration of latent Dirichlet allocation (LDA). We have color coded certain words by the topic they have been assigned to: yellow represents the genetics cluster, pink represents the evolution cluster, blue represent the data analysis cluster, and green represents the neuroscience cluster. Each topic is in turn defined as a sparse distribution over words. This article is not related to neuroscience, so no words are assigned to the green topic. The overall distribution over topic assignments for this document is shown in the right as a sparse histogram. Adapted from Figure 1 of [Ble12]. Used with kind permission of David Blei.

Topic 77

word	prob.
MUSIC	.090
DANCE	.034
SONG	.033
PLAY	.030
SING	.026
SINGING	.026
BAND	.026
PLAYED	.023
SANG	.022
SONGS	.021
DANCING	.020
PIANO	.017
PLAYING	.016
RHYTHM	.015
ALBERT	.013
MUSICAL	.013

Topic 82

word	prob.
LITERATURE	.031
POEM	.028
POETRY	.027
POET	.020
PLAYS	.019
POEMS	.019
PLAY	.015
LITERARY	.013
WRITERS	.013
DRAMA	.012
WROTE	.012
POETS	.011
WRITER	.011
SHAKESPEARE	.010
WRITTEN	.009
STAGE	.009

Topic 166

word	prob.
PLAY	.136
BALL	.129
GAME	.065
PLAYING	.042
HIT	.032
PLAYED	.031
BASEBALL	.027
GAMES	.025
BAT	.019
RUN	.019
THROW	.016
BALLS	.015
TENNIS	.011
HOME	.010
CATCH	.010
FIELD	.010

Figure 27.3: Three topics related to the word play*. From Figure 9 of [SG07]. Used with kind permission of Tom Griffiths.*

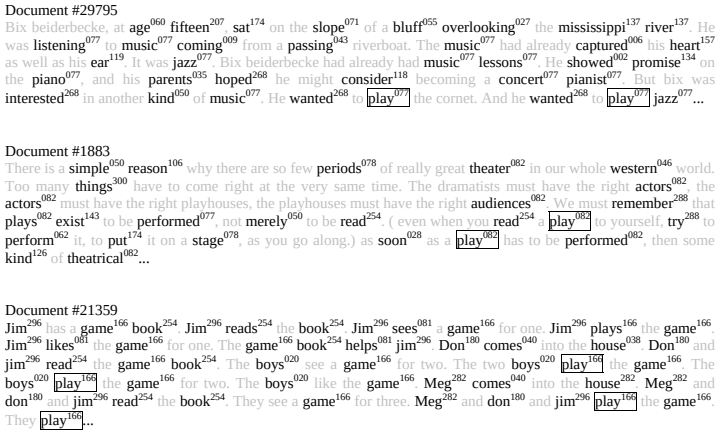


Figure 27.4: Three documents from the TASA corpus containing different senses of the word play*. Grayed out words were ignored by the model, because they correspond to uninteresting stop words (such as "and", "the", etc.) or very low frequency words. From Figure 10 of [SG07]. Used with kind permission of Tom Griffiths.*

Note that an earlier version of LDA, known as **probabilistic LSA**, was proposed in [Hof99]. (LSA stands for "**latent semantic analysis**", and refers to the application of PCA to text data; see [Mur22, Sec 20.5.1.2] for details.) The likelihood function, $p(\boldsymbol{x}|\boldsymbol{z})$, is the same as in LDA, but pLSA does not specify a prior for $\boldsymbol{z}$, since it is designed for posterior analysis of a fixed corpus (similar to LSA), rather than being a true generative model.

27.1.1.2 Polysemy

Each topic is a distribution over words that co-occur together, and which are therefore semantically related. For example, Figure 27.3 shows 3 topics which were learned from an LDA model fit to the **TASA corpus**[1]. These seem to correspond to 3 different senses of the word "play": playing an instrument, a theatrical play, and playing a sports game.

We can use the inferred document-level topic distribution to overcome **polysemy**, i.e., to disambiguate

[1]The TASA corpus is an untagged collection of educational materials consisting of 37,651 documents and 12,190,931 word tokens. Words appearing in fewer than 5 documents were replaced with an asterisk, but punctuation was included. The combined vocabulary was of size 37,202 unique words.

the meaning of a particular word. This is illustrated in Figure 27.4, where a subset of the words are annotated with the topic to which they were assigned (i.e., we show $\text{argmax}_k \, p(c_{nl} = k|\boldsymbol{x}_n)$. In the first document, the word "music" makes it clear that the musical topic (number 77) is present in the document, which in turn makes it more likely that $c_{nl} = 77$ where l is the index corresponding to the word "play".

27.1.1.3 Posterior inference

Many algorithms have been proposed to perform approximate posterior inference in the LDA model. In the original LDA paper, [BNJ03], they use variational mean field inference (see **??**), and in [HBB10], they use stochastic VI (see Section 27.1.6). In Section 27.1.5 we discuss the collapsed Gibbs sampler, which marginalizes out the discrete latents. In [MB16; SS17] they discuss how to learned amortized inference networks to perform VI for the collapsed model.

Recently, there has been considerable interest in spectral methods for fitting LDA-like models which are fast and which come with provable guarantees about the quality of the solution they obtain (unlike MCMC and variational methods, where the solution is just an approximation of unknown quality). These methods make certain (reasonable) assumptions beyond the basic model, such as the existence of some anchor words, which uniquely the topic for a document. See [Aro+13] for details.

27.1.1.4 Determining the number of topics

Choosing N_z, the number of topics, is a standard model selection problem. Here are some approaches that have been taken:

- Use annealed importance sampling (**??**) to approximate the evidence [Wal+09].
- Cross validation, using the log likelihood on a test set.
- Use the variational lower bound as a proxy for $\log p(\mathcal{D}|N_z)$.
- Use non-parametric Bayesian methods [Teh+06].

27.1.2 Correlated topic model

One weakness of LDA is that it cannot capture correlation between topics. For example, if a document has the "business" topic, it is reasonable to expect the "finance" topic to co-occcur. The source of the problem is the use of a Dirichlet prior for $\boldsymbol{z}_n$. The problem with the Dirichlet it that it is characterized by just a mean vector $\boldsymbol{\alpha}$, but its covariance is fixed ($\Sigma_{ij} = -\alpha_i \alpha_j$), rather than being a free parameter.

One way around this is to replace the Dirichlet prior with the **logistic normal** distribution, which is defined as follows:

$$p(\boldsymbol{z}) = \int \text{Cat}(\boldsymbol{z}|\mathcal{S}(\boldsymbol{\epsilon}))\mathcal{N}(\boldsymbol{\epsilon}|\boldsymbol{\mu}, \boldsymbol{\Sigma})d\boldsymbol{\epsilon} \tag{27.3}$$

This is known as the **correlated topic model** [BL07];

The difference from categorical PCA discussed in **??** is that CTM uses a logistic normal to model the mean parameters, so $\boldsymbol{z}_n$ is sparse and non-negative, whereas CatPCA uses a normal to model the natural parameters, so $\boldsymbol{z}_n$ is dense and can be negative. More precisely, the CTM defines $x_{nl} \sim \text{Cat}(\mathbf{W}\mathcal{S}(\boldsymbol{\epsilon}_n))$, but CatPCA defines $\boldsymbol{x}_{nd} \sim \text{Cat}(\mathcal{S}(\mathbf{W}_d \boldsymbol{z}_n))$.

Fitting the CTM model is tricky, since the prior for $\boldsymbol{\epsilon}_n$ is no longer conjugate to the multinomial likelihood for c_{nl}. However, we can derive a variational mean field approximation, as described in [BL07].

Having fit the model, one can then convert $\hat{\boldsymbol{\Sigma}}$ to a sparse precision matrix $\hat{\boldsymbol{\Sigma}}^{-1}$ by pruning low-strength edges, to get a sparse Gaussian graphical model. This allows you to visualize the correlation between topics. Figure 27.5 shows the result of applying this procedure to articles from *Science* magazine, from 1990-1999.

27.1.3 Dynamic topic model

In LDA, the topics (distributions over words) are assumed to be static. In some cases, it makes sense to allow these distributions to evolve smoothly over time. For example, an article might use the topic "neuroscience", but if it was written in the 1900s, it is more likely to use words like "nerve", whereas if it was written in the

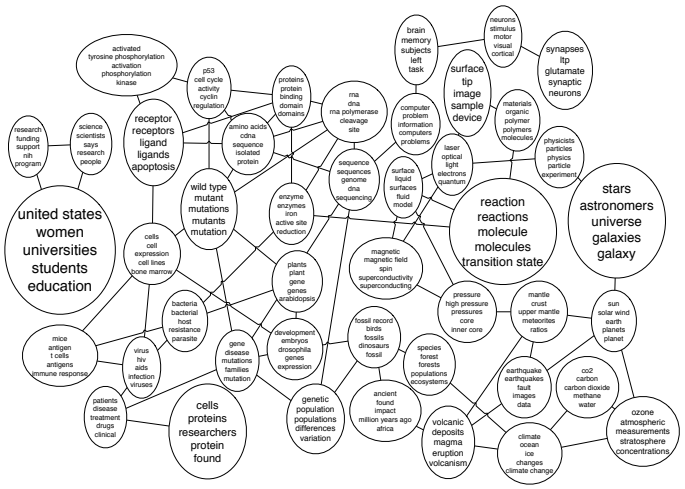


Figure 27.5: Output of the correlated topic model (with $K = 50$ topics) when applied to articles from Science. *Nodes represent topics, with the 5 most probable phrases from each topic shown inside. Font size reflects overall prevalence of the topic. See* http://www.cs.cmu.edu/~lemur/science/ *for an interactive version of this model with 100 topics. Used with kind permission of Figure 2 of [BL07]. Used with kind permission of David Blei.*

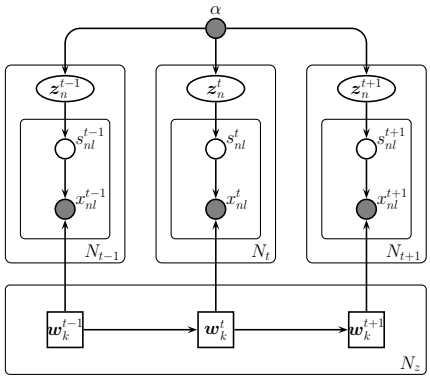


Figure 27.6: The dynamic topic model as a PGM-D.

2000s, it is more likely to use words like "calcium receptor" (this reflects the general trend of neuroscience towards molecular biology).

One way to model this is to assume the topic distributions evolve according to a Gaussian random walk, as in a state space mdoel (see **??**). We can map these Gaussian vectors to probabilities via the softmax function, resulting in the following model:

$$\boldsymbol{w}_k^t | \boldsymbol{w}_k^{t-1} \sim \mathcal{N}(\boldsymbol{w}_{t-1,k}, \sigma^2 \mathbf{1}_{N_w}) \tag{27.4}$$

$$\boldsymbol{z}_n^t \sim \text{Dir}(\alpha \mathbf{1}_{N_z}) \tag{27.5}$$

$$c_{nl}^t | \boldsymbol{z}_n^t \sim \text{Cat}(\boldsymbol{z}_n^t) \tag{27.6}$$

$$x_{nl}^t | c_{nl}^t = k, \mathbf{W}^t \sim \text{Cat}(\mathcal{S}(\boldsymbol{w}_k^t)) \tag{27.7}$$

This is known as a **dynamic topic model** [BL06a]. See Figure 27.6 for the PGM-D.

One can perform approximate inference in this model using a structured mean field method (**??**), that exploits the Kalman smoothing algorithm (**??**) to perform exact inference on the linear-Gaussian chain between the $\boldsymbol{w}_k^t$ nodes (see [BL06a] for details). See the main text for an example of this model applied to 100 years of articles from *Science*.

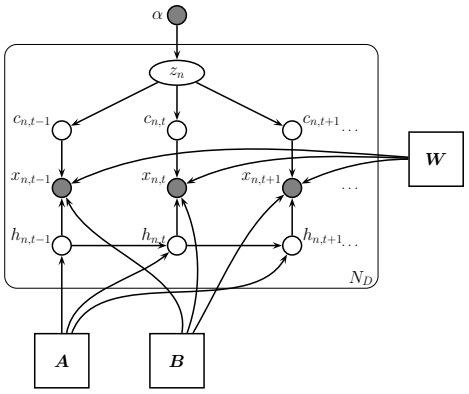


Figure 27.7: LDA-HMM model as a PGM-D.

It is also possible to use amortized inference, and to learn embeddings for each word, which works much better with rare words. This is called the **dynamic embedded topic model** [DRB19].

27.1.4 LDA-HMM

The Latent Dirichlet Allocation (LDA) model of Section 27.1.1 assumes words are exchangeable, and thus ignores word order. A simple way to model sequential dependence between words is to use an HMM. The trouble with HMMs is that they can only model short-range dependencies, so they cannot capture the overall gist of a document. Hence they can generate syntactically correct sentences, but not semantically plausible ones.

It is possible to combine LDA with HMM to create a model called **LDA-HMM** [Gri+04]. This model uses the HMM states to model function or syntactic words, such as "and" or "however", and uses the LDA to model content or semantic words, which are harder to predict. There is a distinguished HMM state which specifies when the LDA model should be used to generate the word; the rest of the time, the HMM generates the word.

More formally, for each document n, the model defines an HMM with states $h_{nl} \in \{0, \ldots, H\}$. In addition, each document has an LDA model associated with it. If $h_{nl} = 0$, we generate word x_{nl} from the semantic LDA model, with topic specified by c_{nl}; otherwise we generate word x_{nl} from the syntactic HMM model. The PGM-D is shown in Figure 27.7. The CPDs are as follows:

$$p(\boldsymbol{z}_n) = \text{Dir}(\boldsymbol{z}_n|\alpha \mathbf{1}_{N_z}) \tag{27.8}$$

$$p(c_{nl} = k|\boldsymbol{z}_n) = z_{nk} \tag{27.9}$$

$$p(h_{n,l} = j|h_{n,l-1} = i) = A_{ij} \tag{27.10}$$

$$p(x_{nl} = d|c_{nl} = k, h_{nl} = j) = \begin{cases} W_{kd} & \text{if } j = 0 \\ B_{jd} & \text{if } j > 0 \end{cases} \tag{27.11}$$

where $\mathbf{W}$ is the usual topic-word matrix, $\mathbf{B}$ is the state-word HMM emission matrix and $\mathbf{A}$ is the state-state HMM transition matrix.

Inference in this model can be done with collapsed Gibbs sampling, analytically integrating out all the continuous quantities. See [Gri+04] for the details.

The results of applying this model (with $N_z = 200$ LDA topics and $H = 20$ HMM states) to the combined Brown and TASA corpora[2] are shown in Table 27.1. We see that the HMM generally is responsible for

[2]The Brown corpus consists of 500 documents and 1,137,466 word tokens, with part-of-speech tags for each token. The TASA corpus is an untagged collection of educational materials consisting of 37,651 documents and 12,190,931 word tokens. Words appearing in fewer than 5 documents were replaced with an asterisk, but punctuation was included. The combined vocabulary was of size 37,202 unique words.

the	the	the	the	the	a	the	the	the
blood	,	,	of	a	the	,	,	,
,	and	and	,	of	of	of	a	a
of	of	of	to	,	,	a	of	in
body	a	in	in	in	in	and	and	game
heart	in	land	and	to	water	in	drink	ball
and	trees	to	classes	picture	is	story	alcohol	and
in	tree	farmers	government	film	and	is	to	team
to	with	for	a	image	matter	to	bottle	to
is	on	farm	state	lens	are	as	in	play
blood	forest	farmers	government	light	water	story	drugs	ball
heart	trees	land	state	eye	matter	stories	drug	game
pressure	forests	crops	federal	lens	molecules	poem	alcohol	team
body	land	farm	public	image	liquid	characters	people	*
lungs	soil	food	local	mirror	particles	poetry	drinking	baseball
oxygen	areas	people	act	eyes	gas	character	person	players
vessels	park	farming	states	glass	solid	author	effects	football
arteries	wildlife	wheat	national	object	substance	poems	marijuana	player
*	area	farms	laws	objects	temperature	life	body	field
breathing	rain	corn	department	lenses	changes	poet	use	basketball
the	in	he	*	be	said	can	time	,
a	for	it	new	have	made	would	way	;
his	to	you	other	see	used	will	years	(
this	on	they	first	make	came	could	day	:
their	with	i	same	do	went	may	part	)
these	at	she	great	know	found	had	number	
your	by	we	good	get	called	must	kind	
her	from	there	small	go		do	place	
my	as	this	little	take		have		
some	into	who	old	find		did		

Table 27.1: Upper row: Topics extracted by the LDA model when trained on the combined Brown and TASA corpora. Middle row: topics extracted by LDA part of LDA-HMM model. Bottom row: topics extracted by HMM part of LDA-HMM model. Each column represents a single topic/class, and words appear in order of probability in that topic/class. Since some classes give almost all probability to only a few words, a list is terminated when the words account for 90% of the probability mass. From Figure 2 of [Gri+04]. Used with kind permission of Tom Griffiths.

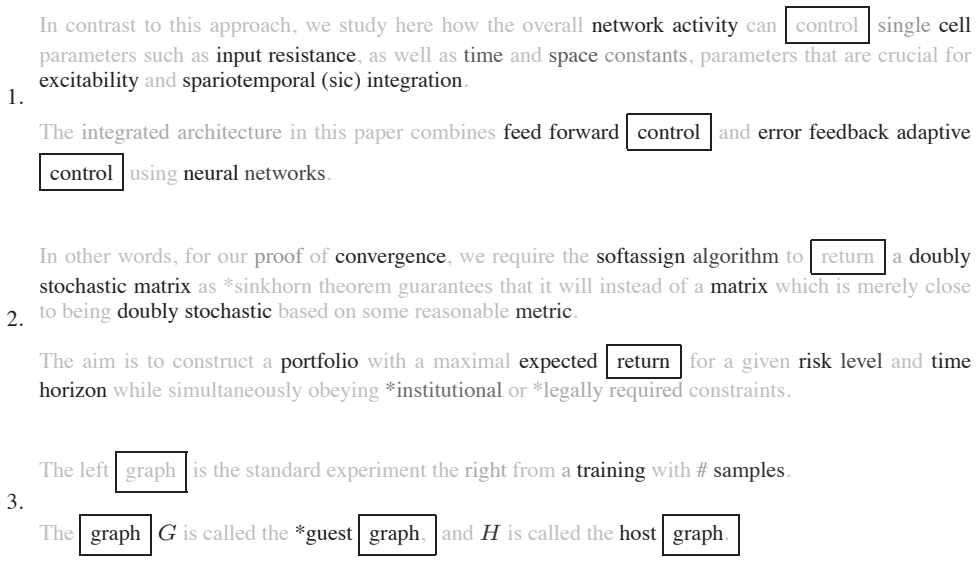


Figure 27.8: Function and content words in the NIPS corpus, as distinguished by the LDA-HMM model. Graylevel indicates posterior probability of assignment to LDA component, with black being highest. The boxed word appears as a function word in one sentence, and as a content word in another sentence. Asterisked words had low frequency, and were treated as a single word type by the model. From Figure 4 of [Gri+04]. Used with kind permission of Tom Griffiths.

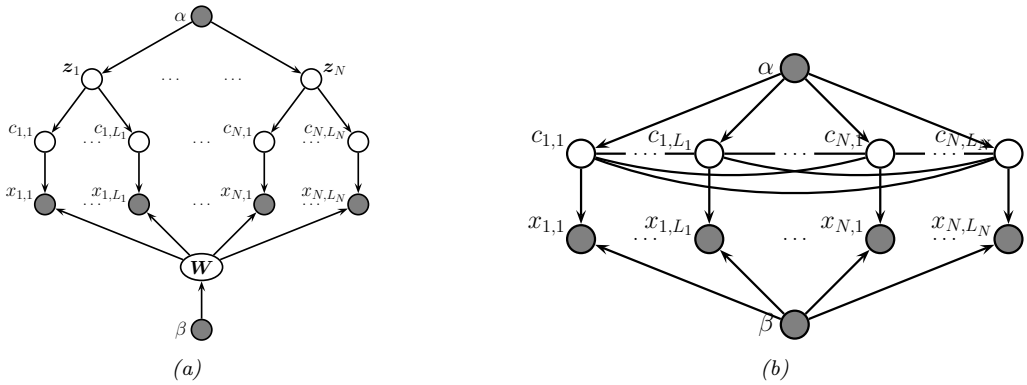


Figure 27.9: (a) LDA unrolled for N documents. (b) Collapsed LDA, where we integrate out the continuous latents $\boldsymbol{z}_n$ and the continuous topic parameters $\mathbf{W}$.

syntactic words, and the LDA for semantics words. If we did not have the HMM, the LDA topics would get "polluted" by function words (see top of figure), which is why such words are normally removed during preprocessing.

The model can also help disambiguate when the same word is being used syntactically or semantically. Figure 27.8 shows some examples when the model was applied to the NIPS corpus.[3] We see that the roles of words are distinguished, e.g., "we require the algorithm to *return* a matrix" (verb) vs "the maximal expected *return*" (noun). In principle, a part of speech tagger could disambiguate these two uses, but note that (1) the LDA-HMM method is fully unsupervised (no POS tags were used), and (2) sometimes a word can have the same POS tag, but different senses, e.g., "the left graph" (a synactic role) vs "the graph G" (a semantic role).

More recently, [Die+17] proposed topic-RNN, which is similar to LDA-HMM, but replaces the HMM model with an RNN, which is a much more powerful model.

27.1.5 Collapsed Gibbs sampling for LDA

In this section, we discuss how to perform inference using MCMC. Vanilla Gibbs sampling samples from the following full conditionals:

$$p(c_{il} = k|\cdot) \propto \exp[\log \pi_{ik} + \log w_{k,x_{il}}] \tag{27.12}$$

$$p(\boldsymbol{\pi}_i|\cdot) = \text{Dir}(\{\alpha_k + \sum_l \mathbb{I}\left(c_{il} = k\right)\}) \tag{27.13}$$

$$p(\boldsymbol{w}_k|\cdot) = \text{Dir}(\{\gamma_v + \sum_i \sum_l \mathbb{I}\left(x_{il} = v, c_{il} = k\right)\}) \tag{27.14}$$

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 27.9(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 $N_{ivk} = \sum_{l=1}^{L_i} \mathbb{I}\left(c_{il} = k, x_{il} = v\right)$ be the number of times word v is assigned to topic k in document i. Let $N_{ik} = \sum_v N_{ivk}$ be the number of times any word from document i has been assigned to topic k. Let $N_{vk} = \sum_i N_{ivk}$ be the number of times word v has been assigned to topic k in any document. Let $N_k = \sum_v N_{vk}$ be the number of words assigned to topic k. Finally, let $L_i = \sum_k N_{ik}$ be the number of words in document i; this is observed.

[3] NIPS stands for "Neural Information Processing Systems". It is one of the top machine learning conferences. The NIPS corpus volumes 1–12 contains 1713 documents.

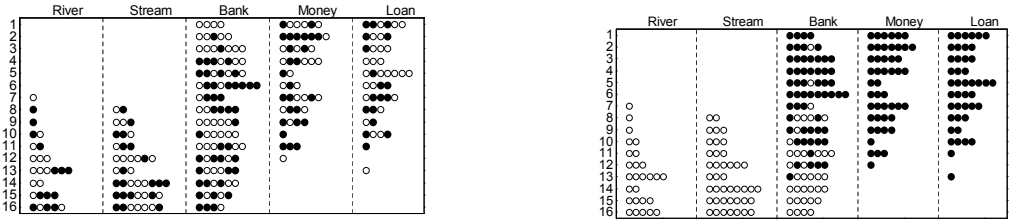


Figure 27.10: Illustration of (collapsed) Gibbs sampling applied to a small LDA example. There are $N = 16$ documents, each containing a variable number of words drawn from a vocabulary of $V = 5$ words, There are two topics. A white dot means word the word is assigned to topic 1, a black dot means the 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.

We can now derive the marginal prior. By applying **??**, one can show that

$$p(\boldsymbol{c}|\alpha) = \prod_i \int \left[\prod_{l=1}^{L_i} \text{Cat}(c_{il}|\boldsymbol{\pi}_i)\right] \text{Dir}(\boldsymbol{\pi}_i|\boldsymbol{\alpha}\mathbf{1}_K) d\boldsymbol{\pi}_i \tag{27.15}$$

$$= \left(\frac{\Gamma(K\alpha)}{\Gamma(\alpha)^K}\right)^N \prod_{i=1}^N \frac{\prod_{k=1}^K \Gamma(N_{ik} + \alpha)}{\Gamma(L_i + K\alpha)} \tag{27.16}$$

By similar reasoning, one can show

$$p(\boldsymbol{x}|\boldsymbol{c}, \beta) = \prod_k \int \left[\prod_{il:c_{il}=k} \text{Cat}(x_{il}|\boldsymbol{w}_k)\right] \text{Dir}(\boldsymbol{w}_k|\beta\mathbf{1}_V) d\boldsymbol{w}_k \tag{27.17}$$

$$= \left(\frac{\Gamma(V\beta)}{\Gamma(\beta)^V}\right)^K \prod_{k=1}^K \frac{\prod_{v=1}^V \Gamma(N_{vk} + \beta)}{\Gamma(N_k + V\beta)} \tag{27.18}$$

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

$$p(c_{i,l} = k|\boldsymbol{c}_{-i,l}, \boldsymbol{y}, \alpha, \beta) \propto \frac{N^-_{v,k} + \beta}{N^-_k + V\beta} \; \frac{N^-_{i,k} + \alpha}{L_i + K\alpha} \tag{27.19}$$

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 (27.19), 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 (27.19), 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 27.10 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.

27.1.6 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.

27.1.6.1 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

$$q(\boldsymbol{z}_n, \boldsymbol{s}_n) = \text{Dir}(\boldsymbol{z}_n|\tilde{\boldsymbol{z}}_n) \prod_l \text{Cat}(s_{nl}|\tilde{\mathbf{N}}_{nl}) \tag{27.20}$$

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:

$$\tilde{N}_{nlk} \propto w_{d,k} \exp(\mathbb{E}[\log z_{nk}]) \tag{27.21}$$

where $d = x_{nl}$, and

$$\mathbb{E}[\log z_{nk}] = \psi_k(\tilde{\boldsymbol{z}}_n) \triangleq \Psi(\tilde{z}_{nk}) - \psi(\sum_{k'} \tilde{z}_{nk'}) \tag{27.22}$$

where ψ is the digamma function. The update for $q(\boldsymbol{z}_n)$ is obtained by adding up the expected counts:

$$\tilde{z}_{nk} = \alpha_k + \sum_l \tilde{N}_{nlk} \tag{27.23}$$

The M step is obtained by adding up the expected counts and normalizing:

$$\hat{w}_{dk} \propto \beta_d + \sum_{n=1}^{N} \sum_{l=1}^{L_n} \tilde{N}_{nlk} \mathbb{I}(x_{nl} = d) \tag{27.24}$$

27.1.6.2 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

$$q(\boldsymbol{z}_n, \mathbf{N}_n) = \text{Dir}(\boldsymbol{z}_n|\tilde{\boldsymbol{z}}_n) \prod_d \mathcal{M}(\mathbf{N}_{nd}|x_{nd}, \tilde{N}_{nd}) \tag{27.25}$$

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

$$\tilde{z}_{nk} = \alpha_k + \sum_d x_{nd} \tilde{N}_{ndk} \tag{27.26}$$

$$\tilde{N}_{ndk} \propto w_{dk} \exp(\mathbb{E}[\log z_{nk}]) \tag{27.27}$$

The M step becomes

$$\hat{w}_{dk} \propto \beta_d + \sum_n x_{nd} \tilde{N}_{ndk} \tag{27.28}$$

27.1.6.3 Bayesian version

Algorithm 4: Batch VB for LDA

```
Input: {x_nd}, N_z, α, β;
Estimate w̃_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
        (z̃_n, Ñ_n) = VB-Estep(x_n, W̃, α);
        a_dk += x_nd Ñ_ndk;
    // M step ;
    for each topic k = 1 : N_z do
        w̃_dk = β_d + a_dk;
function (z̃_n, Ñ_n) = VB-Estep(x_n, W̃, α);
Initialize z̃_nk = α_k;
repeat
    z̃_n^old = z̃_n, z̃_nk = α_k;
    for each word d = 1 : N_w do
        for each topic k = 1 : N_z do
            Ñ_ndk = exp(ψ_k(w̃_d) + ψ_k(z̃_n^old));
        Ñ_nd = normalize(Ñ_nd);
        z̃_n += x_nd Ñ_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

$$q(\boldsymbol{z}_n, \mathbf{N}_n, \mathbf{W}) = \text{Dir}(\boldsymbol{z}_n|\tilde{\boldsymbol{z}}_n) \prod_d \mathcal{M}(\mathbf{N}_{nd}|x_{nd}, \tilde{N}_{nd}) \prod_k \text{Dir}(\boldsymbol{w}_k|\tilde{\boldsymbol{w}}_k) \tag{27.29}$$

The update for $\tilde{N}_{ndk}$ changes, to the following:

$$\tilde{N}_{ndk} \propto \exp\left(\mathbb{E}[\log w_{dk}] + \mathbb{E}[\log z_{nk}]\right) \tag{27.30}$$

The M step is the same as before:

$$\hat{w}_{dk} \propto \beta_d + \sum_n x_{nd} \tilde{N}_{ndk} \tag{27.31}$$

No normalization is required, since we are just updating the pseudcounts. The overall algorithm is summarized in Algorithm 4.

Algorithm 5: 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; ;
  $(\tilde{\boldsymbol{z}}_n, \tilde{\mathbf{N}}_n) = \text{VB-Estep}(\boldsymbol{x}_n, \tilde{\mathbf{W}}, \boldsymbol{\alpha})$;
  $\tilde{w}_{dk}^{new} = \beta_d + N x_{nd} \tilde{N}_{ndk}$;
  $\tilde{w}_{dk} = (1 - \eta_t)\tilde{w}_{dk} + \eta_t \tilde{w}_{dk}^{new}$;

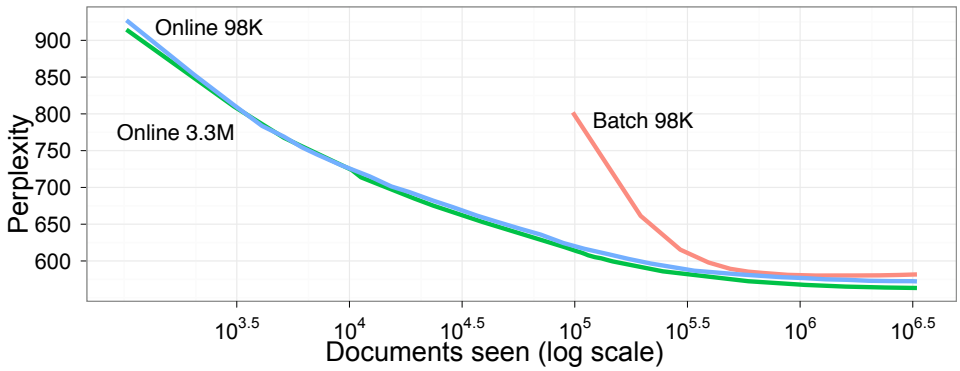


Figure 27.11: Test perplexity vs number of training documents for batch and online VB-LDA. From Figure 1 of [HBB10]. Used with kind permission of David Blei.

27.1.6.4 Online (SVI) version

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 5. In practice, we should use mini-batches, as explained in **??**. In [HBB10], they used a batch of size 256–4096.

Figure 27.11 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

Hidden Markov models

Chapter 29

State-space models

29.1 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 $\boldsymbol{z}(t) \in \mathbb{R}^N$:

$$\dot{\boldsymbol{z}}(t) = \mathbf{A}\boldsymbol{z}(t) + \mathbf{B}\boldsymbol{u}(t) \tag{29.1}$$

$$\boldsymbol{y}(t) = \mathbf{C}\boldsymbol{z}(t) + \mathbf{D}\boldsymbol{u}(t) \tag{29.2}$$

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 **??**. If we set $\alpha = \frac{1}{2}$ in **??** we get the **bilinear method**, which preserves the stability of the system [ZCC07]. The result is

$$\boldsymbol{z}_t = \overline{\mathbf{A}}\boldsymbol{z}_{t-1} + \overline{\mathbf{B}}\boldsymbol{u}_t \tag{29.3}$$

$$\boldsymbol{y}_t = \overline{\mathbf{C}}\boldsymbol{z}_t \tag{29.4}$$

$$\overline{\mathbf{A}} = (\mathbf{I} - \Delta/2 \cdot \mathbf{A})^{-1}(\mathbf{I} + \Delta/ \cdot \mathbf{A}) \in \mathbb{R}^{N \times N} \tag{29.5}$$

$$\overline{\mathbf{B}} = (\mathbf{I} - \Delta/2 \cdot \mathbf{A})^{-1}\Delta\mathbf{B} \in \mathbb{R}^{N} \tag{29.6}$$

$$\overline{\mathbf{C}} = \mathbf{C} \in \mathbb{R}^{N \times 1} \tag{29.7}$$

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

$$\boldsymbol{z}_0 = \overline{\mathbf{B}}u_0,\ \boldsymbol{z}_1 = \overline{\mathbf{A}\mathbf{B}}u_0 + \overline{\mathbf{B}}u_1,\ \boldsymbol{z}_2 = \overline{\mathbf{A}}^2\overline{\mathbf{B}}u_0 + \overline{\mathbf{A}\mathbf{B}}u_1 + \overline{\mathbf{B}}u_2,\ \cdots \tag{29.8}$$

$$y_0 = \overline{\mathbf{C}\mathbf{B}}u_0,\ y_1 = \overline{\mathbf{C}\mathbf{A}\mathbf{B}}u_0 + \overline{\mathbf{C}\mathbf{B}}u_1,\ y_2 = \overline{\mathbf{C}\mathbf{A}}^2\overline{\mathbf{B}}u_0 + \overline{\mathbf{C}\mathbf{A}\mathbf{B}}u_1 + \overline{\mathbf{C}\mathbf{B}}u_2, \cdots \tag{29.9}$$

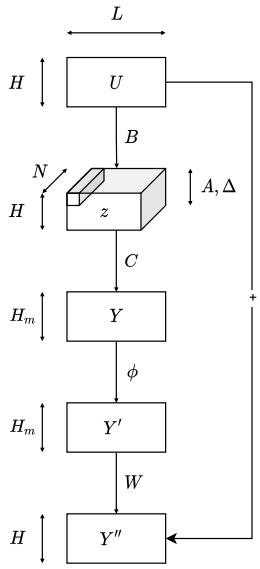


Figure 29.1: Illustration of one S4 block. The input $\mathbf{X}$ *has* H *sequences, each of length* L*. These get mapped (in parallel for each of the sequences) to the output* $\mathbf{Y}$ *by a (noiseless) linear SSM with state size* N *and parameters* $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$*. 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 **??**.) 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:

$$y_k = \overline{\mathbf{C}}\overline{\mathbf{A}}^k\overline{\mathbf{B}}u_0 + \overline{\mathbf{C}}\overline{\mathbf{A}}^{k-1}\overline{\mathbf{B}}u_1 + \cdots + \overline{\mathbf{C}}\overline{\mathbf{B}}u_k \tag{29.10}$$

$$\boldsymbol{y} = \overline{\mathbf{K}} \odot \boldsymbol{u} \tag{29.11}$$

$$\overline{\mathbf{K}} = (\overline{\mathbf{C}}\overline{\mathbf{B}}, \overline{\mathbf{C}}\overline{\mathbf{A}}\overline{\mathbf{B}}, \cdots, \overline{\mathbf{C}}\overline{\mathbf{A}}^{L-1}\overline{\mathbf{B}}) \tag{29.12}$$

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 29.1. 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 30

Graph learning

30.1 Learning tree structures

Since the problem of structure learning for general graphs is NP-hard [Chi96], we start by considering the special case of trees. Trees are special because we can learn their structure efficiently, as we discuss below, and because, once we have learned the tree, we can use them for efficient exact inference, as discussed in **??**.

30.1.1 Directed or undirected tree?

Before continuing, we need to discuss the issue of whether we should use directed or undirected trees. A directed tree, with a single root node r, defines a joint distribution as follows:

$$p(\boldsymbol{x}|T) = \prod_{t \in V} p(x_t|x_{\text{pa}(t)}) \tag{30.1}$$

where we define $\text{pa}(r) = \emptyset$. For example, in Figure 30.1(b-c), we have

$$p(x_1, x_2, x_3, x_4|T) = p(x_1)p(x_2|x_1)p(x_3|x_2)p(x_4|x_2) \tag{30.2}$$

$$= p(x_2)p(x_1|x_2)p(x_3|x_2)p(x_4|x_2) \tag{30.3}$$

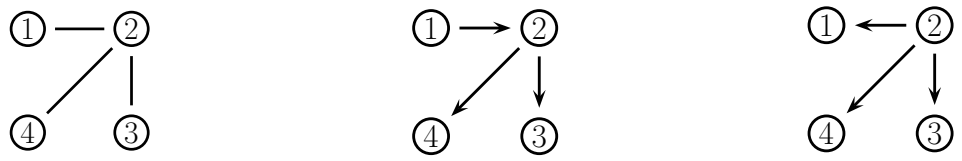
We see that the choice of root does not matter: both of these models are equivalent.

To make the model more symmetric, it is preferable to use an undirected tree. This can be represented as follows:

$$p(\boldsymbol{x}|T) = \prod_{t \in V} p(x_t) \prod_{(s,t) \in E} \frac{p(x_s, x_t)}{p(x_s)p(x_t)} \tag{30.4}$$

where $p(x_s, x_t)$ is an edge marginal and $p(x_t)$ is a node marginal. For example, in Figure 30.1(a) we have

$$p(x_1, x_2, x_3, x_4|T) = p(x_1)p(x_2)p(x_3)p(x_4)\frac{p(x_1, x_2)p(x_2, x_3)p(x_2, x_4)}{p(x_1)p(x_2)p(x_2)p(x_3)p(x_2)p(x_4)} \tag{30.5}$$



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

Figure 30.1: An undirected tree and two equivalent directed trees.

To see the equivalence with the directed representation, let us cancel terms to get

$$p(x_1, x_2, x_3, x_4|T) = p(x_1, x_2)\frac{p(x_2, x_3)}{p(x_2)}\frac{p(x_2, x_4)}{p(x_2)} \tag{30.6}$$

$$= p(x_1)p(x_2|x_1)p(x_3|x_2)p(x_4|x_2) \tag{30.7}$$

$$= p(x_2)p(x_1|x_2)p(x_3|x_2)p(x_4|x_2) \tag{30.8}$$

where $p(x_t|x_s) = p(x_s, x_t)/p(x_s)$.

Thus a tree can be represented as either an undirected or directed graph: the number of parameters is the same, and hence the complexity of learning is the same. And of course, inference is the same in both representations, too. The undirected representation, which is symmetric, is useful for structure learning, but the directed representation is more convenient for parameter learning.

30.1.2 Chow-Liu algorithm

Using Equation (30.4), we can write the log-likelihood for a tree as follows:

$$\log p(\mathcal{D}|\boldsymbol{\theta}, T) = \sum_t \sum_k N_{tk} \log p(x_t = k|\boldsymbol{\theta}) + \sum_{s,t} \sum_{j,k} N_{stjk} \log \frac{p(x_s = j, x_t = k|\boldsymbol{\theta})}{p(x_s = j|\boldsymbol{\theta})p(x_t = k|\boldsymbol{\theta})} \tag{30.9}$$

where N_{stjk} is the number of times node s is in state j and node t is in state k, and N_{tk} is the number of times node t is in state k. We can rewrite these counts in terms of the empirical distribution: $N_{stjk} = Np_{\mathcal{D}}(x_s = j, x_t = k)$ and $N_{tk} = Np_{\mathcal{D}}(x_t = k)$. Setting $\boldsymbol{\theta}$ to the MLEs, this becomes

$$\frac{\log p(\mathcal{D}|\boldsymbol{\theta}, T)}{N} = \sum_{t \in \mathcal{V}} \sum_k p_{\mathcal{D}}(x_t = k) \log p_{\mathcal{D}}(x_t = k) \tag{30.10}$$

$$+ \sum_{(s,t) \in \mathcal{E}(T)} \mathbb{I}(x_s, x_t|\hat{\boldsymbol{\theta}}_{st}) \tag{30.11}$$

where $\mathbb{I}(x_s, x_t|\hat{\boldsymbol{\theta}}_{st}) \geq 0$ is the mutual information between x_s and x_t given the empirical distribution:

$$\mathbb{I}(x_s, x_t|\hat{\boldsymbol{\theta}}_{st}) = \sum_j \sum_k p_{\mathcal{D}}(x_s = j, x_t = k) \log \frac{p_{\mathcal{D}}(x_s = j, x_t = k)}{p_{\mathcal{D}}(x_s = j)p_{\mathcal{D}}(x_t = k)} \tag{30.12}$$

Since the first term in Equation (30.11) is independent of the topology T, we can ignore it when learning structure. Thus the tree topology that maximizes the likelihood can be found by computing the maximum weight spanning tree, where the edge weights are the pairwise mutual informations, $\mathbb{I}(y_s, y_t|\hat{\boldsymbol{\theta}}_{st})$. This is called the **Chow-Liu algorithm** [CL68].

There are several algorithms for finding a max spanning tree (MST). The two best known are Prim's algorithm and Kruskal's algorithm. Both can be implemented to run in $O(E \log V)$ time, where $E = V^2$ is the number of edges and V is the number of nodes. See e.g., [SW11, Sec 4.3] for details. Thus the overall running time is $O(NV^2 + V^2 \log V)$, where the first term is the cost of computing the sufficient statistics.

Figure 30.2 gives an example of the method in action, applied to the binary 20 newsgroups data shown in **??**. The tree has been arbitrarily rooted at the node representing "email". The connections that are learned seem intuitively reasonable.

30.1.3 Finding the MAP forest

Since all trees have the same number of parameters, we can safely use the maximum likelihood score as a model selection criterion without worrying about overfitting. However, sometimes we may want to fit a

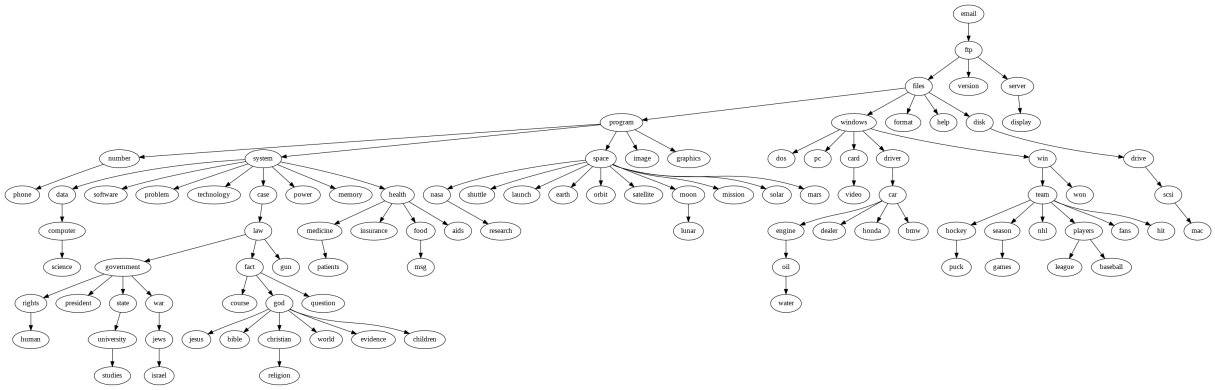


Figure 30.2: The MLE tree estimated from the 20-newsgroup data. Generated by chow_liu_tree_demo.py.

forest rather than a single tree, since inference in a forest is much faster than in a tree (we can run belief propagation in each tree in the forest in parallel). The MLE criterion will never choose to omit an edge. However, if we use the marginal likelihood or a penalized likelihood (such as BIC), the optimal solution may be a forest. Below we give the details for the marginal likelihood case.

In Section 30.2.3.2, we explain how to compute the marginal likelihood of any DAG using a Dirichlet prior for the CPTs. The resulting expression can be written as follows:

$$\log p(\mathcal{D}|T) = \sum_{t \in \mathcal{V}} \log \int \prod_{i=1}^{N} p(x_{it}|\boldsymbol{x}_{i,\text{pa}(t)}|\boldsymbol{\theta}_t) p(\boldsymbol{\theta}_t) d\boldsymbol{\theta}_t = \sum_t \text{score}(\mathbf{N}_{t,\text{pa}(t)}) \tag{30.13}$$

where $\mathbf{N}_{t,\text{pa}(t)}$ are the counts (sufficient statistics) for node t and its parents, and score is defined in Equation (30.26).

Now suppose we only allow DAGs with at most one parent. Following [HGC95, p227], let us associate a weight with each $s \to t$ edge, $w_{s,t} \triangleq \text{score}(t|s) - \text{score}(t|0)$, where $\text{score}(t|0)$ is the score when t has no parents. Note that the weights might be negative (unlike the MLE case, where edge weights are aways non-negative because they correspond to mutual information). Then we can rewrite the objective as follows:

$$\log p(\mathcal{D}|T) = \sum_t \text{score}(t|\text{pa}(t)) = \sum_t w_{\text{pa}(t),t} + \sum_t \text{score}(t|0) \tag{30.14}$$

The last term is the same for all trees T, so we can ignore it. Thus finding the most probable tree amounts to finding a **maximal branching** in the corresponding weighted directed graph. This can be found using the algorithm in [GGS84].

If the scoring function is prior and likelihood equivalent (these terms are explained in Section 30.2.3.3), we have

$$\text{score}(s|t) + \text{score}(t|0) = \text{score}(t|s) + \text{score}(s|0) \tag{30.15}$$

and hence the weight matrix is symmetric. In this case, the maximal branching is the same as the maximal weight forest. We can apply a slightly modified version of the MST algorithm to find this [EAL10]. To see this, let $G = (V, E)$ be a graph with both positive and negative edge weights. Now let G' be a graph obtained by omitting all the negative edges from G. This cannot reduce the total weight, so we can find the maximum weight forest of G by finding the MST for each connected component of G'. We can do this by running Kruskal's algorithm directly on G': there is no need to find the connected components explicitly.

30.1.4 Mixtures of trees

A single tree is rather limited in its expressive power. Later in this chapter we discuss ways to learn more general graphs. However, the resulting graphs can be expensive to do inference in. An interesting alternative

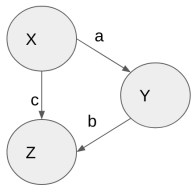


Figure 30.3: A simple linear Gaussian model. .

is to learn a **mixture of trees** [MJ00], where each mixture component may have a different tree topology. This is like an unsupervised version of the TAN classifier discussed in **??**. We can fit a mixture of trees by using EM: in the E step, we compute the responsibilities of each cluster for each data point, and in the M step, we use a weighted version of the Chow-Liu algorithm. See [MJ00] for details.

In fact, it is possible to create an "infinite mixture of trees", by integrating out over all possible trees. Remarkably, this can be done in V^3 time using the matrix tree theorem. This allows us to perform exact Bayesian inference of posterior edge marginals etc. However, it is not tractable to use this infinite mixture for inference of hidden nodes. See [MJ06] for details.

30.2 Learning DAG structures

In this section, we discuss how to estimate the structure of directed graphical models from observational data. This is often called **Bayes net structure learning**. We can only do this if we make the faithfulness assumption, which we explain in Section 30.2.1. Furthermore our output will be a set of equivalent DAGs, rather than a single unique DAG, as we explain in Section 30.2.2. After introducing these restrictions, we discuss some statistical and algorithmic techniques. If the DAG is interpreted causal, these techniques can be used for **causal discovery**, although this relies on additional assumptions about non-confounding. For more details, see e.g., [GZS19].

30.2.1 Faithfulness

The Markov assumption allows us to infer CI properties of a distribution p from a graph G. To go in the opposite direction, we need to assume that the generating distribution p is **faithful** to the generating DAG G. This means that all the conditional indepence (CI) properties of p are exactly captured by the graphical structure, so $I(p) = I(G)$; this means there cannot be any CI properties in p that are due to particular settings of the parameters (such as zeros in a regression matrix) that are not graphically explicit. (For this reason, a faithful distribution is also called a **stable** distribution.)

Let us consider an example of a non-faithful distribution (from [PJS17, Sec 6.5.3]). Consider a linear Gaussian model of the form

$$X = E_X, \; E_X \sim \mathcal{N}(0, \sigma_X^2) \tag{30.16}$$

$$Y = aX + E_Y, \; E_Y \sim \mathcal{N}(0, \sigma_Y^2) \tag{30.17}$$

$$Z = bY + cX + E_Z, \; E_Z \sim \mathcal{N}(0, \sigma_Z^2) \tag{30.18}$$

where the error terms are independent. If $ab + c = 0$, then $X \perp Z$, even though this is not implied by the DAG in Figure 30.3. Fortunately, this kind of accidental cancellation happens with zero probability if the coefficients are drawn randomly from positive densities [SGS00, Thm 3.2].

30.2.2 Markov equivalence

Even with the faithfulness assumption, we cannot always uniquely identify a DAG from a joint distribution. To see this, consider the following 3 DGMs: $X \to Y \to Z$, $X \leftarrow Y \leftarrow Z$ and $X \leftarrow Y \to Z$. These all represent

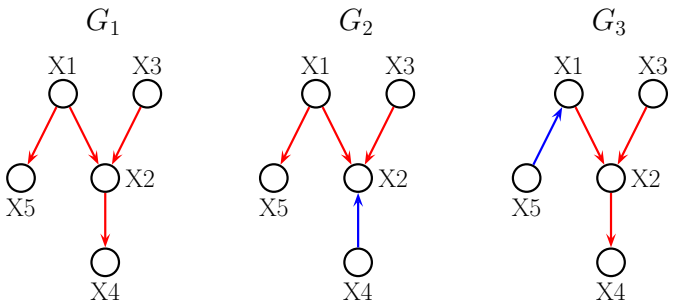


Figure 30.4: Three DAGs. G_1 and G_3 are Markov equivalent, G_2 is not.

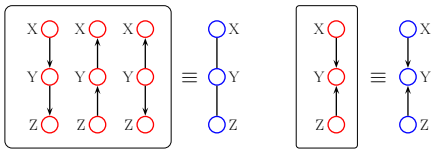


Figure 30.5: PDAG representation of Markov equivalent DAGs.

the same set of CI statements, namely

$$X \perp Z|Y, \quad X \not\perp Z \tag{30.19}$$

We say these graphs are **Markov equivalent**, since they encode the same set of CI assumptions. That is, they all belong to the same Markov **equivalence class**. However, the DAG $X \rightarrow Y \leftarrow Z$ encodes $X \perp Z$ and $X \not\perp Z|Y$, so corresponds to a different distribution.

In [VP90], they prove the following theorem.

Theorem 30.2.1. *Two structures are Markov equivalent iff they have the same* **skeleton***, i.e., the have the same edges (disregarding direction) and they have the same set of v-structures (colliders whose parents are not adjacent).*

For example, referring to Figure 30.4, we see that $G_1 \not\equiv G_2$, since reversing the $2 \rightarrow 4$ arc creates a new v-structure. However, $G_1 \equiv G_3$, since reversing the $1 \rightarrow 5$ arc does not create a new v-structure.

We can represent a Markov equivalence class using a single **partially directed acyclic graph** or **PDAG** (also called an **essential graph** or **pattern**), in which some edges are directed and some undirected (see **??**). The undirected edges represent reversible edges; any combination is possible so long as no new v-structures are created. The directed edges are called **compelled edges**, since changing their orientation would change the v-structures and hence change the equivalence class. For example, the PDAG $X - Y - Z$ represents $\{X \rightarrow Y \rightarrow Z, X \leftarrow Y \leftarrow Z, X \leftarrow Y \rightarrow Z\}$ which encodes $X \not\perp Z$ and $X \perp Z|Y$. See Figure 30.4 for another example.

The significance of the above theorem is that, when we learn the DAG structure from data, we will not be able to uniquely identify all of the edge directions, even given an infinite amount of data. We say that we can learn DAG structure "up to Markov equivalence". This also cautions us not to read too much into the meaning of particular edge orientations, since we can often change them without changing the model in any observable way. (If we want to distinguish between edge orientations within a PDAG (e.g., if we want to imbue a causal interpretation on the edges), we can use interventional data, as we discuss in Section 30.4.2.)

30.2.3 Bayesian model selection: statistical foundations

In this section, we discuss how to compute the exact posterior over graphs, $p(G|\mathcal{D})$, ignoring for now the issue of computational tractability. We assume there is no missing data, and that there are no hidden variables. This is called the **complete data assumption**.

For simplicity, we will focus on the case where all the variables are categorical and all the CPDs are tables. Our presentation is based in part on [HGC95], although we will follow the notation of **??**. In particular, let $x_{it} \in \{1, \ldots, K_t\}$ be the value of node t in case i, where K_t is the number of states for node t. Let $\theta_{tck} \triangleq p(x_t = k|\boldsymbol{x}_{\mathrm{pa}(t)} = c)$, for $k = 1 : K_t$, and $c = 1 : C_t$, where C_t is the number of parent combinations (possible conditioning cases). For notational simplicity, we will often assume $K_t = K$, so all nodes have the same number of states. We will also let $d_t = \dim(\mathrm{pa}(t))$ be the degree or fan-in of node t, so that $C_t = K^{d_t}$.

30.2.3.1 Deriving the likelihood

Assuming there is no missing data, and that all CPDs are tabular, the likelihood can be written as follows:

$$p(\mathcal{D}|G, \boldsymbol{\theta}) = \prod_{i=1}^{N} \prod_{t=1}^{V} \mathrm{Cat}(x_{it}|\boldsymbol{x}_{i,\mathrm{pa}(t)}, \boldsymbol{\theta}_t) \tag{30.20}$$

$$= \prod_{i=1}^{N} \prod_{t=1}^{V} \prod_{c=1}^{C_t} \prod_{k=1}^{K_t} \theta_{tck}^{\mathbb{I}\left(x_{i,t}=k, \boldsymbol{x}_{i,\mathrm{pa}(t)}=c\right)} = \prod_{t=1}^{V} \prod_{c=1}^{C_t} \prod_{k=1}^{K_t} \theta_{tck}^{N_{tck}} \tag{30.21}$$

where N_{tck} is the number of times node t is in state k and its parents are in state c. (Technically these counts depend on the graph structure G, but we drop this from the notation.)

30.2.3.2 Deriving the marginal likelihood

Choosing the graph with the maximum likelihood will always pick a fully connected graph (subject to the acyclicity constraint), since this maximizes the number of parameters. To avoid such overfitting, we will choose the graph with the maximum marginal likelihood, $p(\mathcal{D}|G)$, where we integrate out the parameters; the magic of the Bayesian Occam's razor (**??**) will then penalize overly complex graphs.

To compute the marginal likelihood, we need to specify priors on the parameters. We will make two standard assumptions. First, we assume **global prior parameter independence**, which means $p(\boldsymbol{\theta}) = \prod_{t=1}^{V} p(\boldsymbol{\theta}_t)$. Second, we assume **local prior parameter independence**, which means $p(\boldsymbol{\theta}_t) = \prod_{c=1}^{C_t} p(\boldsymbol{\theta}_{tc})$ for each t. It turns out that these assumtions imply that the prior for each row of each CPT must be a Dirichlet [GH97], that is, $p(\boldsymbol{\theta}_{tc}) = \mathrm{Dir}(\boldsymbol{\theta}_{tc}|\boldsymbol{\alpha}_{tc})$. Given these assumptions, and using the results of **??**, we can write down the marginal likelihood of any DAG as follows:

$$p(\mathcal{D}|G) = \prod_{t=1}^{V} \prod_{c=1}^{C_t} \int \left[\prod_{i:x_{i,\mathrm{pa}(t)}=c} \mathrm{Cat}(x_{it}|\boldsymbol{\theta}_{tc}) \right] \mathrm{Dir}(\boldsymbol{\theta}_{tc}) d\boldsymbol{\theta}_{tc} \tag{30.22}$$

$$= \prod_{t=1}^{V} \prod_{c=1}^{C_t} \frac{B(\mathbf{N}_{tc} + \boldsymbol{\alpha}_{tc})}{B(\boldsymbol{\alpha}_{tc})} \tag{30.23}$$

$$= \prod_{t=1}^{V} \prod_{c=1}^{C_t} \frac{\Gamma(N_{tc})}{\Gamma(N_{tc} + \alpha_{tc})} \prod_{k=1}^{K_t} \frac{\Gamma(N_{tck} + \alpha_{tck}^G)}{\Gamma(\alpha_{tck}^G)} \tag{30.24}$$

$$= \prod_{t=1}^{V} \mathrm{score}(\mathbf{N}_{t,\mathrm{pa}(t)}) \tag{30.25}$$

where $N_{tc} = \sum_k N_{tck}$, $\alpha_{tc} = \sum_k \alpha_{tck}$, $\mathbf{N}_{t,\mathrm{pa}(t)}$ is the vector of counts (sufficient statistics) for node t and its

parents, and score() is a local scoring function defined by

$$\text{score}(\mathbf{N}_{t,\text{pa}(t)}) \triangleq \prod_{c=1}^{C_t} \frac{B(\mathbf{N}_{tc} + \boldsymbol{\alpha}_{tc})}{B(\boldsymbol{\alpha}_{tc})} \tag{30.26}$$

We say that the marginal likelihood **decomposes** or factorizes according to the graph structure.

30.2.3.3 Setting the prior

How should we set the hyper-parameters α_{tck}? It is tempting to use a Jeffreys prior of the form $\alpha_{tck} = \frac{1}{2}$ (**??**). However, it turns out that this violates a property called **likelihood equivalence**, which is sometimes considered desirable. This property says that if G_1 and G_2 are Markov equivalent (Section 30.2.2), they should have the same marginal likelihood, since they are essentially equivalent models. Geiger and Heckerman [GH97] proved that, for complete graphs, the only prior that satisfies likelihood equivalence and parameter independence is the Dirichlet prior, where the pseudo counts have the form

$$\alpha_{tck} = \alpha \; p_0(x_t = k, \boldsymbol{x}_{\text{pa}(t)} = c) \tag{30.27}$$

where $\alpha > 0$ is called the **equivalent sample size**, and p_0 is some prior joint probability distribution. This is called the **BDe** prior, which stands for Bayesian Dirichlet likelihood equivalent.

To derive the hyper-parameters for other graph structures, Geiger and Heckerman [GH97] invoked an additional assumption called **parameter modularity**, which says that if node X_t has the same parents in G_1 and G_2, then $p(\boldsymbol{\theta}_t|G_1) = p(\boldsymbol{\theta}_t|G_2)$. With this assumption, we can always derive $\boldsymbol{\alpha}_t$ for a node t in any other graph by marginalizing the pseudo counts in Equation (30.27).

Typically the prior distribution p_0 is assumed to be uniform over all possible joint configurations. In this case, we have $\alpha_{tck} = \frac{\alpha}{K_t C_t}$, since $p_0(x_t = k, \boldsymbol{x}_{\text{pa}(t)} = c) = \frac{1}{K_t C_t}$. Thus if we sum the pseudo counts over all $C_t \times K_t$ entries in the CPT, we get a total equivalent sample size of α. This is called the **BDeu** prior, where the "u" stands for uniform. This is the most widely used prior for learning Bayes net structures. For advice on setting the global tuning parameter α, see [SKM07].

30.2.3.4 Example: analysis of the college plans dataset

We now consider a larger example from [HMC97], who analyzed a dataset of 5 variables, related to the decision of high school students about whether to attend college. Specifically, the variables are as follows:

- Sex: Male or female
- SES: Socio economic status: low, lower middle, upper middle or high.
- IQ: Intelligence quotient: discretized into low, lower middle, upper middle or high.
- PE: Parental encouragment: low or high
- CP: College plans: yes or no.

These variables were measured for 10,318 Wisconsin high school seniors. There are $2 \times 4 \times 4 \times 2 \times = 128$ possible joint configurations.

Heckerman et al. computed the exact posterior over all 29,281 possible 5 node DAGs, except for ones in which SEX and/or SES have parents, and/or CP have children. (The prior probability of these graphs was set to 0, based on domain knowledge.) They used the BDeu score with $\alpha = 5$, although they said that the results were robust to any α in the range 3 to 40. The top two graphs are shown in Figure 30.6. We see that the most probable one has approximately all of the probability mass, so the posterior is extremely peaked.

It is tempting to interpret this graph in terms of causality (see **??** for a detailed discussion of this topic). In particular, it seems that socio-economic status, IQ and parental encouragment all causally influence the decision about whether to go to college, which makes sense. Also, sex influences college plans only indirectly through parental encouragement, which also makes sense. However, the direct link from socio economic status to IQ seems surprising; this may be due to a hidden common cause. In Section 30.2.8.5 we will re-examine this dataset allowing for the presence of hidden variables.

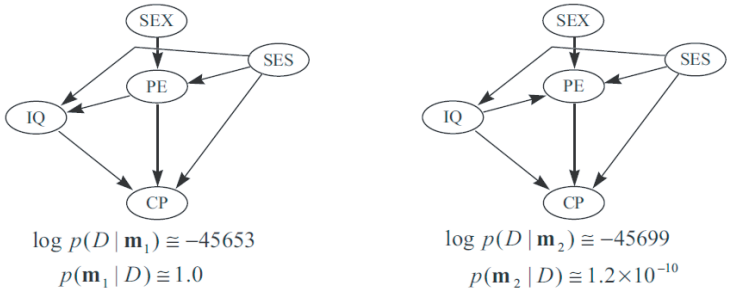


Figure 30.6: The two most probable DAGs learned from the Sewell-Shah data. From [HMC97]. Used with kind permission of David Heckerman

30.2.3.5 Marginal likelihood for non-tabular CPDs

If all CPDs are linear Gaussian, we can replace the Dirichlet-multinomial model with the normal-gamma model, and thus derive a different exact expression for the marginal likelihood. See [GH94] for the details. In fact, we can easily combine discrete nodes and Gaussian nodes, as long as the discrete nodes always have discrete parents; this is called a **conditional Gaussian** DAG. Again, we can compute the marginal likelihood in closed form. See [BD03] for the details.

In the general case (i.e., everything except Gaussians and CPTs), we need to approximate the marginal likelihood. The simplest approach is to use the BIC approximation, which has the form

$$\sum_t \log p(\mathcal{D}_t|\hat{\boldsymbol{\theta}}_t) - \frac{K_t C_t}{2} \log N \tag{30.28}$$

30.2.4 Bayesian model selection: algorithms

In this section, we discuss some algorithms for approximately computing the mode of (or samples from) the posterior $p(G|D)$.

30.2.4.1 The K2 algorithm for known node orderings

Suppose we know a total ordering of the nodes. Then we can compute the distribution over parents for each node independently, without the risk of introducing any directed cycles: we simply enumerate over all possible subsets of ancestors and compute their marginal likelihoods. If we just return the best set of parents for each node, we get the the **K2 algorithm** [CH92]. In this case, we can find the best set of parents for each node using ℓ_1-regularization, as shown in [SNMM07].

30.2.4.2 Dynamic programming algorithms

In general, the ordering of the nodes is not known, so the posterior does not decompose. Nevertheless, we can use dynamic programming to find the globally optimal MAP DAG (up to Markov equivalence), as shown in [KS04; SM06].

If our goal is knowledge discovery, the MAP DAG can be misleading, for reasons we discussed in **??**. A better approach is to compute the marginal probability that each edge is present, $p(G_{st} = 1|\mathcal{D})$. We can also compute these quantities using dynamic programming, as shown in [Koi06; PK11].

Unfortunately, all of these methods take $V 2^V$ time in the general case, making them intractable for graphs with more than about 16 nodes.

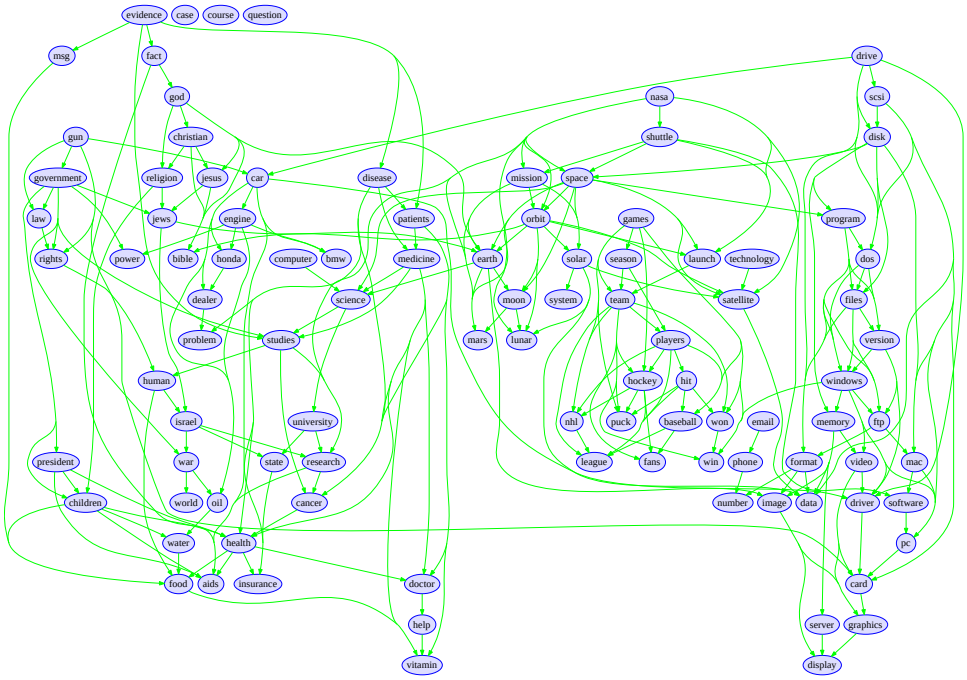


Figure 30.7: A locally optimal DAG learned from the 20-newsgroup data. From Figure 4.10 of [Sch10a]. Used with kind permission of Mark Schmidt.

30.2.4.3 Scaling up to larger graphs

The main challenge in computing the posterior over DAGs is that there are so many possible graphs. More precisely, [Rob73] showed that the number of DAGs on D nodes satisfies the following recurrence:

$$f(D) = \sum_{i=1}^{D} (-1)^{i+1} \binom{D}{i} 2^{i(D-i)} f(D-i) \tag{30.29}$$

for $D > 2$. The base case is $f(1) = 1$. Solving this recurrence yields the following sequence: 1, 3, 25, 543, 29281, 3781503, etc.[1]

Indeed, the general problem of finding the globally optimal MAP DAG is provably NP-complete [Chi96]. In view of the enormous size of the hypothesis space, we are generally forced to use approximate methods, some of which we review below.

30.2.4.4 Hill climbing methods for approximating the mode

A common way to find an approximate MAP graph structure is to use a greedy hill climbing method. At each step, the algorithm proposes small changes to the current graph, such as adding, deleting or reversing a single edge; it then moves to the neighboring graph which most increases the posterior. The method stops when it reaches a local maximum. It is important that the method only proposes local changes to the graph, since this enables the change in marginal likelihood (and hence the posterior) to be computed in constant time (assuming we cache the sufficient statistics). This is because all but one or two of the terms in Equation (30.23) will cancel out when computing the log Bayes factor $\delta(G \rightarrow G') = \log p(G'|\mathcal{D}) - \log p(G|\mathcal{D})$.

We can initialize the search from the best tree, which can be found using exact methods discussed in Section 30.1.2. For speed, we can restrict the search so it only adds edges which are part of the Markov

[1] A longer list of values can be found at http://www.research.att.com/~njas/sequences/A003024. Interestingly, the number of DAGs is equal to the number of (0,1) matrices all of whose eigenvalues are positive real numbers [McK+04].

blankets estimated from a dependency network [Sch10a]. Figure 30.7 gives an example of a DAG learned in this way from the 20-newsgroup data. For binary data, it is possible to use techniques from frequent itemset mining to find good Markov blanket candidates, as described in [GM04].

We can use techniques such as multiple random restarts to increase the chance of finding a good local maximum. We can also use more sophisticated local search methods, such as genetic algorithms or simulated annealing, for structure learning. (See also Section 30.2.6 for gradient based techniques based on continuous relaxations.)

It is also possible to perform the greedy search in the space of PDAGs instead of in the space of DAGs; this is known as the **greedy equivalence search** method [Chi02]. Although each step is somewhat more complicated, the advantage is that the search space is smaller.

30.2.4.5 Sampling methods

If our goal is knowledge discovery, the MAP DAG can be misleading, for reasons we discussed in **??**. A better approach is to compute the probability that each edge is present, $p(G_{st} = 1|\mathcal{D})$. We can do this exactly using dynamic programming [Koi06; PK11], although this can be expensive. An approximate method is to sample DAGs from the posterior, and then to compute the fraction of times there is an $s \to t$ edge or path for each (s, t) pair. The standard way to draw samples is to use the Metropolis Hastings algorithm (**??**), where we use the same local proposal as we did in greedy search [MR94].

A faster-mixing method is to use a collapsed MH sampler, as suggested in [FK03]. This exploits the fact that, if a total ordering of the nodes is known, we can select the parents for each node independently, without worrying about cycles, as discussed in Section 30.2.4.1. By summing over all possible choice of parents, we can marginalize out this part of the problem, and just sample total orders. [EW08] also use order-space (collapsed) MCMC, but this time with a parallel tempering MCMC algorithm.

30.2.5 Constraint-based approach

We now present an approach to learning a DAG structure — up to Markov equivalence (the output of the method is a PDAG) — that uses local conditional independence tests, rather than scoring models globally with a likelihood. The CI tests are combined together to infer the global graph structure, so this approach is called **constraint-based**. The advantage of CI testing is that it is more local and does not require specifying a complete model. (However, the form of the CI test implicitly relies on assumptions, see e.g., [SP18].)

30.2.5.1 IC algorithm

The original algorithm, due to Verma and Pearl [VP90], was called the **IC algorithm**, which stands for "inductive causation". The method is as follows [Pea09, p50]:

1. For each pair of variables a and b, search for a set S_{ab} such that $a \perp b|S_{ab}$. Construct an undirected graph such that a and b are connected iff no such set S_{ab} can be found (i.e., they cannot be made conditionally independent).

2. Orient the edges involved in v-structures as follows: for each pair of nonadjacent nodes a and b with a common neighbor c, check if $c \in S_{ab}$; if it is, the corresponding DAG must be $a \to c \to b$, $a \leftarrow c \to b$ or $a \leftarrow c \leftarrow b$, so we cannot determine the direction; if it is not, the DAG must be $a \to c \leftarrow b$, so add these arrows to the graph.

3. In the partially directed graph that results, orient as many of the undirected edges as possible, subject to two conditions: (1) the orientation should not create a new v-structure (since that would have been detected already if it existed), and (2) the orientation should not create a directed cycle. More precisely, follow the rules shown in Figure 30.8. In the first case, if $X \to Y$ has a known orientation, but $Y - Z$ is unknown, then we must have $Y \to Z$, otherwise we would have created a new v-structure $X \to Y \leftarrow Z$, which is not allowed. The other two cases follow similar reasoning.

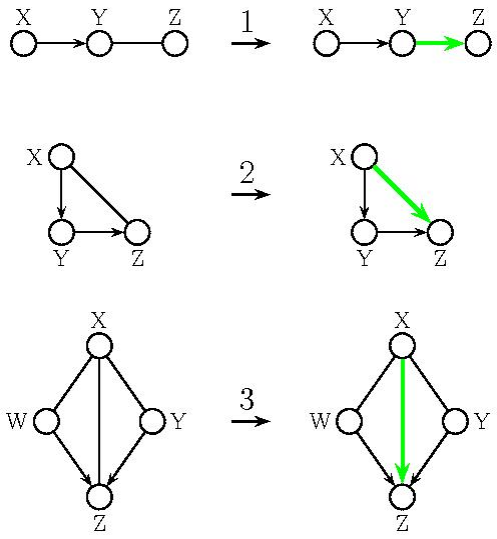


Figure 30.8: The 3 rules for inferring compelled edges in PDAGs. Adapted from [Pe'05].

30.2.5.2 PC algorithm

A significant speedup of IC, known as the **PC algorithm** after is creators Peter Spirtes and Clark Glymour [SG91], can be obtained by ordering the search for separating sets in step 1 in terms of sets of increasing cardinality. We start with a fully connected graph, and then look for sets S_{ab} of size 0, then of size 1, and so on; as soon we find a separating set, we remove the corresponding edge. See Figure 30.9 for an example.

Another variant on the PC algorithm is to learn the original undirected structure (i.e., the Markov blanket of each node) using generic variable selection techniques instead of CI tests. This tends to be more robust, since it avoids issues of statisical significance that can arise with independence tests. See [PE08] for details.

The running time of the PC algorithm is $O(D^{K+1})$ [SGS00, p85], where D is the number of nodes and K is the maximal degree (number of neighbors) of any node in the corresponding undirected graph.

30.2.5.3 Frequentist vs Bayesian methods

The IC/PC algorithm relies on an oracle that can test for conditional independence between any set of variables, $A \perp B|C$. This can be approximated using hypothesis testing methods applied to a finite data set, such as chi-squared tests for discrete data. However, such methods work poorly with small sample sizes, and can run into problems with multiple testing (since so many hypotheses are being compared). In addition, errors made at any given step can lead to an incorrect final result, as erroneous constraints get propagated. In practice it is a common to use a hybrid approach, where we use IC/PC to create an initial structure, and then use this to speed up Bayesian model selection, which tends to be more robust, since it avoids any hard decisions about conditional independence or lack thereof.

30.2.6 Methods based on sparse optimization

There is a 1:1 connection between sparse graphs and sparse adjacency matrices. This suggests that we can perform structure learning by using continuous optimization methods that enforce sparsity, similar to lasso and other ℓ_1 penalty methods (**??**). In the cases of undirected graphs, this is relatively straightforward, and results in a convex objective, as we discuss in Section 30.3.2. However, in the case of DAGs, the problem is harder, because of the acyclicity constraint. Fortunately, [Zhe+18] showed how to encode this constraint as a smooth penalty term. (They call their method "DAGs with no tears", since it is supposed to be painless to

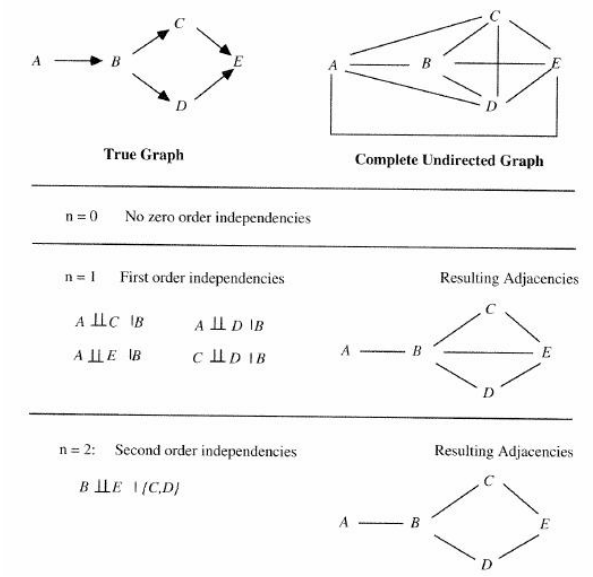


Figure 30.9: Example of step 1 of the PC algorithm. From Figure 5.1 of [SGS00]. Used with kind permission of Peter Spirtes.

use.) In particular, they show how to convert the combinatorial problem into a continuous problem:

$$\min_{\mathbf{W}\in\mathbb{R}^{D\times D}} f(\mathbf{W}) \;\text{ s.t. }\; G(\mathbf{W}) \in \text{DAGs} \iff \min_{\mathbf{W}\in\mathbb{R}^{D\times D}} f(\mathbf{W}) \;\text{ s.t. }\; h(\mathbf{W}) = 0 \tag{30.30}$$

Here $\mathbf{W}$ is a weighted adjacency matrix on D nodes, $G(\mathbf{W})$ is the corresponding graph (obtained by thresholding $\mathbf{W}$ at 0), $f(\mathbf{W})$ is a scoring function (e.g., penalized log likelihood), and $h(\mathbf{W})$ is a constraint function that measures how close $\mathbf{W}$ is to defining a DAG. The constraint is given by

$$h(\mathbf{W}) = \text{tr}((\mathbf{I} + \alpha\mathbf{W})^d) - d \propto \text{tr}(\sum_{k=1}^{d} \alpha^k \mathbf{W}^k) \tag{30.31}$$

where $\mathbf{W}^k = \mathbf{W}\cdots\mathbf{W}$ with k terms, and $\alpha > 0$ is a regularizer. Element (i, j) of $\mathbf{W}^k$ will be non-zero iff theree is a path from j to i made of K educes. Hence the diagonal elements count the number of paths from an edge to itself in k steps. Thus $h(\boldsymbol{w})$ will be 0 if $\mathbf{W}$ defines a valid DAG.

The scoring function considered in [Zhe+18] has the form

$$f(\mathbf{W}) = \frac{1}{2N}||\mathbf{X} - \mathbf{X}\mathbf{W}||_F^2 + \lambda||\mathbf{W}||_1 \tag{30.32}$$

where $\mathbf{X} \in \mathbb{R}^{ND}$ is the data matrix. The show how to find a local optimum of the equality constrained objective using gradient-based methods. The cost per iteration is $O(D^3)$.

Several extensions of this have been proposed. For example, [Yu+19] replace the Gaussian noise assumption with a VAE (variational autoencoder, **??**), and use a graph neural network as the encoder/decoder. And [Lac+20] relax the linearity assumption, and allow for the use of neural network dependencies between variables.

30.2.7 Consistent estimators

A natural question is whether any of the above algorithms can recover the "true" DAG structure G (up to Markov equivalence), in the limit of infinite data. We assume that the data was generated by a distribution p that is faithful to G (see Section 30.2.1).

The posterior mode (MAP) is known to converge to the MLE, which in turn will converge to the true graph G (up to Markov equivalence), so any exact algorithm for Bayesian inference is a consistent estimator. [Chi02] showed that his greedy equivalence search method (which is a form of hill climbing in the space of PDAGs) is a consistent estimator. Similarly, [SGS00; KB07] showed that the PC is a consistent estimator. However, the running time of these algorithms might be exponential in the number of nodes. Also, all of these methods assume that all the variables are fully observed.

30.2.8 Handling latent variables

In general, we will not get to observe the values of all the nodes (i.e., the complete data assumption does not hold), either because we have missing data, and/ or because we have hidden variables. This makes it intractable to compute the marginal likelihood of any given graph structure, as we discuss in Section 30.2.8.1. It also opens up new problems, such as knowing how many hidden variables to add to the model, and how to connect them, as we discuss in Section 30.2.8.7.

30.2.8.1 Approximating the marginal likelihood

If we have hidden or missing variables $\boldsymbol{h}$, the marginal likelihood is given by

$$p(\mathcal{D}|G) = \int \sum_{\boldsymbol{h}} p(\mathcal{D}, \boldsymbol{h}|\boldsymbol{\theta}, G)p(\boldsymbol{\theta}|G)d\boldsymbol{\theta} = \sum_{\boldsymbol{h}} \int p(\mathcal{D}, \boldsymbol{h}|\boldsymbol{\theta}, G)p(\boldsymbol{\theta}|G)d\boldsymbol{\theta} \tag{30.33}$$

In general this is intractable to compute. For example, consider a mixture model, where we don't observe the cluster label. In this case, there are K^N possible completions of the data (assuming we have K clusters); we can evaluate the inner integral for each one of these assignments to $\boldsymbol{h}$, but we cannot afford to evaluate all of the integrals. (Of course, most of these integrals will correspond to hypotheses with little posterior support, such as assigning single data points to isolated clusters, but we don't know ahead of time the relative weight of these assignments.) Below we mention some faster deterministic approximations for the marginal likelihood.

30.2.8.2 BIC approximation

A simple approximation to the marginal likelihood is to use the BIC score (**??**), which is given by

$$\text{BIC}(G) \triangleq \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \frac{\log N}{2} \dim(G) \tag{30.34}$$

where $\dim(G)$ is the number of degrees of freedom in the model and $\hat{\boldsymbol{\theta}}$ is the MAP or ML estimate. However, the BIC score often severely underestimates the true marginal likelihood [CH97], resulting in it selecting overly simple models. We discuss some better approximations below.

30.2.8.3 Cheeseman-Stutz approximation

We now discuss the **Cheeseman-Stutz approximation** (CS) to the marginal likelihood [CS96]. We first compute a MAP estimate of the parameters $\hat{\boldsymbol{\theta}}$ (e.g., using EM). Denote the expected sufficient statistics of the data by $\overline{\mathcal{D}} = \overline{\mathcal{D}}(\hat{\boldsymbol{\theta}})$; in the case of discrete variables, we just "fill in" the hidden variables with their expectation. We then use the exact marginal likelihood equation on this filled-in data:

$$p(\mathcal{D}|G) \approx p(\overline{\mathcal{D}}|G) = \int p(\overline{\mathcal{D}}|\boldsymbol{\theta}, G)p(\boldsymbol{\theta}|G)d\boldsymbol{\theta} \tag{30.35}$$

However, comparing this to Equation (30.33), we can see that the value will be exponentially smaller, since it does not sum over all values of $\boldsymbol{h}$. To correct for this, we first write

$$\log p(\mathcal{D}|G) = \log p(\overline{\mathcal{D}}|G) + \log p(\mathcal{D}|G) - \log p(\overline{\mathcal{D}}|G) \tag{30.36}$$

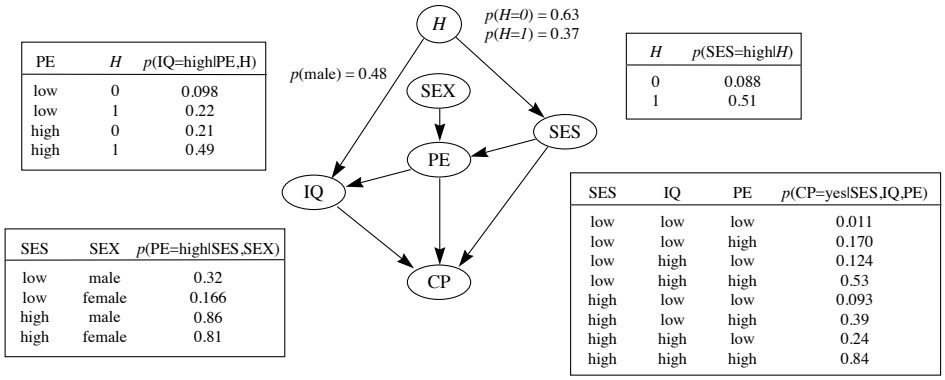


Figure 30.10: The most probable DAG with a single binary hidden variable learned from the Sewell-Shah data. MAP estimates of the CPT entries are shown for some of the nodes. From [HMC97]. Used with kind permission of David Heckerman.

and then we apply a BIC approximation to the last two terms:

$$\log p(\mathcal{D}|G) - \log p(\overline{\mathcal{D}}|G) \approx \left[\log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \frac{\log N}{2}\dim(G)\right] - \left[\log p(\overline{\mathcal{D}}|\hat{\boldsymbol{\theta}}, G) - \frac{\log N}{2}\dim(G)\right] \tag{30.37}$$

$$= \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \log p(\overline{\mathcal{D}}|\hat{\boldsymbol{\theta}}, G) \tag{30.38}$$

Putting it altogether we get

$$\log p(\mathcal{D}|G) \approx \log p(\overline{\mathcal{D}}|G) + \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \log p(\overline{\mathcal{D}}|\hat{\boldsymbol{\theta}}, G) \tag{30.39}$$

The first term $p(\overline{\mathcal{D}}|G)$ can be computed by plugging in the filled-in data into the exact marginal likelihood. The second term $p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G)$, which involves an exponential sum (thus matching the "dimensionality" of the left hand side) can be computed using an inference algorithm. The final term $p(\overline{\mathcal{D}}|\hat{\boldsymbol{\theta}}, G)$ can be computed by plugging in the filled-in data into the regular likelihood.

30.2.8.4 Variational Bayes EM

An even more accurate approach is to use the variational Bayes EM algorithm. Recall from **??** that the key idea is to make the following factorization assumption:

$$p(\boldsymbol{\theta}, \boldsymbol{z}_{1:N}|\mathcal{D}) \approx q(\boldsymbol{\theta})q(\boldsymbol{z}) = q(\boldsymbol{\theta})\prod_i q(\boldsymbol{z}_i) \tag{30.40}$$

where $\boldsymbol{z}_i$ are the hidden variables in case i. In the E step, we update the $q(\boldsymbol{z}_i)$, and in the M step, we update $q(\boldsymbol{\theta})$. The corresponding variational free energy provides a lower bound on the log marginal likelihood. In [BG06], it is shown that this bound is a much better approximation to the true log marginal likelihood (as estimated by a slow annealed importance sampling procedure) than either BIC or CS. In fact, one can prove that the variational bound will always be more accurate than CS (which in turn is always more accurate than BIC).

30.2.8.5 Example: college plans revisited

Let us revisit the college plans dataset from Section 30.2.3.4. Recall that if we ignore the possibility of hidden variables there was a direct link from socio economic status to IQ in the MAP DAG. Heckerman et

al. decided to see what would happen if they introduced a hidden variable H, which they made a parent of both SES and IQ, representing a hidden common cause. They also considered a variant in which H points to SES, IQ and PE. For both such cases, they considered dropping none, one, or both of the SES-PE and PE-IQ edges. They varied the number of states for the hidden node from 2 to 6. Thus they computed the approximate posterior over $8 \times 5 = 40$ different models, using the CS approximation.

The most probable model which they found is shown in Figure 30.10. This is $2 \cdot 10^{10}$ times more likely than the best model containing no hidden variable. It is also $5 \cdot 10^9$ times more likely than the second most probable model with a hidden variable. So again the posterior is very peaked.

These results suggests that there is indeed a hidden common cause underlying both the socio-economic status of the parents and the IQ of the children. By examining the CPT entries, we see that both SES and IQ are more likely to be high when H takes on the value 1. They interpret this to mean that the hidden variable represents "parent quality" (possibly a genetic factor). Note, however, that the arc between H and SES can be reversed without changing the v-structures in the graph, and thus without affecting the likelihood; this underscores the difficulty in interpreting hidden variables.

Interestingly, the hidden variable model has the same conditional independence assumptions amongst the visible variables as the most probable visible variable model. So it is not possible to distinguish between these hypotheses by merely looking at the empirical conditional independencies in the data (which is the basis of the constraint-based approach to structure learning discussed in Section 30.2.5). Instead, by adopting a Bayesian approach, which takes parsimony into account (and not just conditional independence), we can discover the possible existence of hidden factors. This is the basis of much of scientific and everday human reasoning (see e.g. [GT09] for a discussion).

30.2.8.6 Structural EM

One way to perform structural inference in the presence of missing data is to use a standard search procedure (deterministic or stochastic), and to use the methods from Section 30.2.8.1 to estimate the marginal likelihood. However, this approach is not very efficient, because the marginal likelihood does not decompose when we have missing data, and nor do its approximations. For example, if we use the CS approximation or the VBEM approximation, we have to perform inference in every neighboring model, just to evaluate the quality of a single move!

[Fri97; Thi+98] presents a much more efficient approach called the **structural EM** algorithm. The basic idea is this: instead of fitting each candidate neighboring graph and then filling in its data, fill in the data once, and use this filled-in data to evaluate the score of all the neighbors. Although this might be a bad approximation to the marginal likelihood, it can be a good enough approximation of the difference in marginal likelihoods between different models, which is all we need in order to pick the best neighbor.

More precisely, define $\overline{\mathcal{D}}(G_0, \hat{\boldsymbol{\theta}}_0)$ to be the data filled in using model G_0 with MAP parameters $\hat{\boldsymbol{\theta}}_0$. Now define a modified BIC score as follows:

$$\text{BIC}(G, \mathcal{D}) \triangleq \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \frac{\log N}{2} \dim(G) + \log p(G) + \log p(\hat{\boldsymbol{\theta}}|G) \tag{30.41}$$

where we have included the log prior for the graph and parameters. One can show [Fri97] that if we pick a graph G which increases the BIC score relative to G_0 on the expected data, it will also increase the score on the actual data, i.e.,

$$\text{BIC}(G, \overline{\mathcal{D}}) - \text{BIC}(G_0, \overline{\mathcal{D}}) \leq \text{BIC}(G, \mathcal{D}) - \text{BIC}(G_0, \mathcal{D}) \tag{30.42}$$

To convert this into an algorithm, we proceed as follows. First we initialize with some graph G_0 and some set of parameters $\boldsymbol{\theta}_0$. Then we fill-in the data using the current parameters — in practice, this means when we ask for the expected counts for any particular family, we perform inference using our current model. (If we know which counts we will need, we can precompute all of them, which is much faster.) We then evaluate the BIC score of all of our neighbors using the filled-in data, and we pick the best neighbor. We then refit the model parameters, fill-in the data again, and repeat. For increased speed, we may choose to only refit the model every few steps, since small changes to the structure hopefully won't invalidate the parameter estimates and the filled-in data too much.

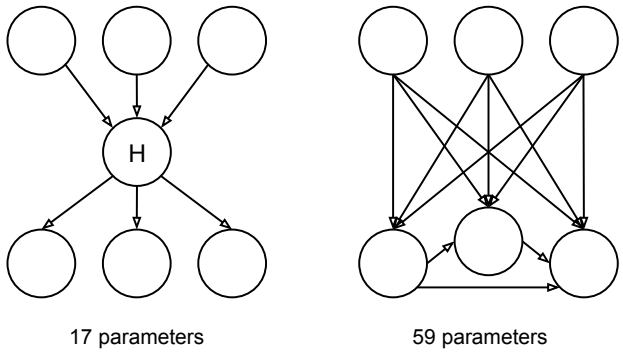


Figure 30.11: A DGM with and without hidden variables. For example, the leaves might represent medical symptoms, the root nodes primary causes (such as smoking, diet and exercise), and the hidden variable can represent mediating factors, such as heart disease. Marginalizing out the hidden variable induces a clique.

One interesting application is to learn a phylogenetic tree structure. Here the observed leaves are the DNA or protein sequences of currently alive species, and the goal is to infer the topology of the tree and the values of the missing internal nodes. There are many classical algorithms for this task (see e.g., [Dur+98]), but one that uses structural EM is discussed in [Fri+02].

Another interesting application of this method is to learn sparse mixture models [BF02]. The idea is that we have one hidden variable C specifying the cluster, and we have to choose whether to add edges $C \to X_t$ for each possible feature X_t. Thus some features will be dependent on the cluster id, and some will be independent. (See also [LFJ04] for a different way to perform this task, using regular EM and a set of bits, one per feature, that are free to change across data cases.)

30.2.8.7 Discovering hidden variables

In Section 30.2.8.5, we introduced a hidden variable "by hand", and then figured out the local topology by fitting a series of different models and computing the one with the best marginal likelihood. How can we automate this process?

Figure 30.11 provides one useful intuition: if there is a hidden variable in the "true model", then its children are likely to be densely connected. This suggest the following heuristic [Eli+00]: perform structure learning in the visible domain, and then look for **structural signatures**, such as sets of densely connected nodes (near-cliques); introduce a hidden variable and connect it to all nodes in this near-clique; and then let structural EM sort out the details. Unfortunately, this technique does not work too well, since structure learning algorithms are biased against fitting models with densely connected cliques.

Another useful intuition comes from clustering. In a flat mixture model, also called a **latent class model**, the discrete latent variable provides a compressed representation of its children. Thus we want to create hidden variables with high mutual information with their children.

One way to do this is to create a tree-structured hierarchy of latent variables, each of which only has to explain a small set of children. [Zha04] calls this a **hierarchical latent class model**. They propose a greedy local search algorithm to learn such structures, based on adding or deleting hidden nodes, adding or deleting edges, etc. (Note that learning the optimal latent tree is NP-hard [Roc06].)

Recently [HW11] proposed a faster greedy algorithm for learning such models based on agglomerative hierarchical clustering. Rather than go into details, we just give an example of what this system can learn. Figure 30.12 shows part of a latent forest learned from the 20-newsgroup data. The algorithm imposes the constraint that each latent node has exactly two children, for speed reasons. Nevertheless, we see interpretable clusters arising. For example, Figure 30.12 shows separate clusters concerning medicine, sports and religion. This provides an alternative to LDA and other topic models (Section 27.1.1), with the added advantage that inference in latent trees is exact and takes time linear in the number of nodes.

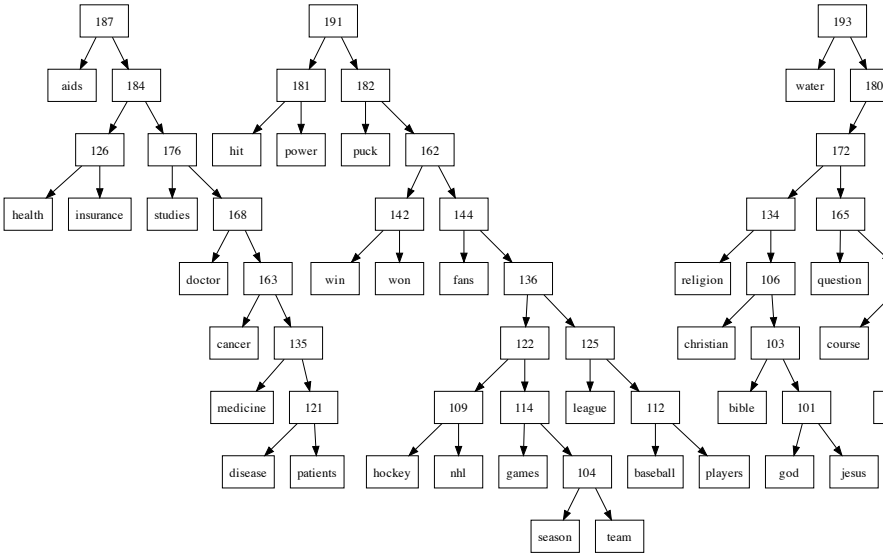


Figure 30.12: Part of a hierarchical latent tree learned from the 20-newsgroup data. From Figure 2 of [HW11]. Used with kind permission of Stefan Harmeling.

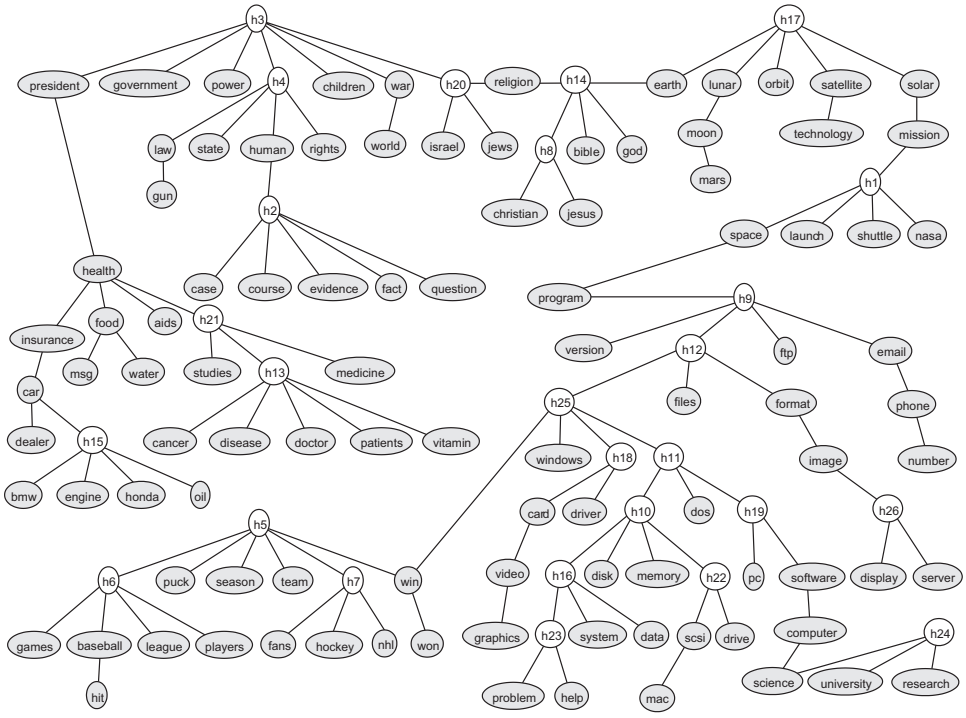


Figure 30.13: A partially latent tree learned from the 20-newsgroup data. Note that some words can have multiple meanings, and get connected to different latent variables, representing different "topics". For example, the word "win" can refer to a sports context (represented by h5) or the Microsoft Windows context (represented by h25). From Figure 12 of [Cho+11]. Used with kind permission of Jin Choi.

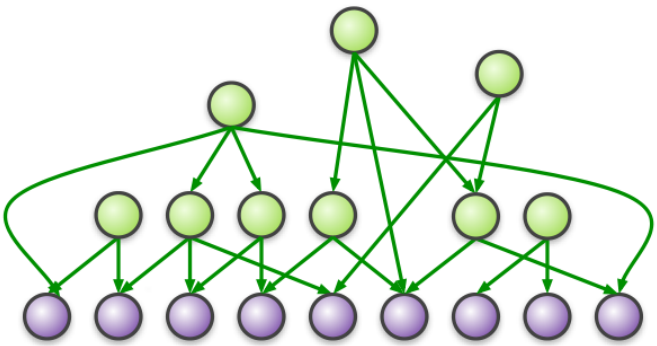


Figure 30.14: Google's rephil model. Leaves represent presence or absence of words. Internal nodes represent clusters of co-occuring words, or "concepts". All nodes are binary, and all CPDs are noisy-OR. The model contains 12 million word nodes, 1 million latent cluster nodes, and 350 million edges. Used with kind permission of Brian Milch.

An alternative approach is proposed in [Cho+11], in which the observed data is not constrained to be at the leaves. This method starts with the Chow-Liu tree on the observed data, and then adds hidden variables to capture higher-order dependencies between internal nodes. This results in much more compact models, as shown in Figure 30.13. This model also has better predictive accuracy than other approaches, such as mixture models, or trees where all the observed data is forced to be at the leaves. Interestingly, one can show that this method can recover the exact latent tree structure, providing the data is generated from a tree. See [Cho+11] for details. Note, however, that this approach, unlike [Zha04; HW11], requires that the cardinality of all the variables, hidden and observed, be the same. Furthermore, if the observed variables are Gaussian, the hidden variables must be Gaussian also.

30.2.8.8 Example: Google's Rephil

In this section, we describe a huge DGM called **Rephil**, which was automatically learned from data.[2] The model is widely used inside Google for various purposes, including their famous AdSense system.[3]

The model structure is shown in Figure 30.14. The leaves are binary nodes, and represent the presence or absence of words or compounds (such as "New York City") in a text document or query. The latent variables are also binary, and represent clusters of co-occuring words. All CPDs are noisy-OR, since some leaf nodes (representing words) can have many parents. This means each edge can be augmented with a hidden variable specifying if the link was activated or not; if the link is not active, then the parent cannot turn the child on. (A very similar model was proposed independently in [SH06].)

Parameter learning is based on EM, where the hidden activation status of each edge needs to be inferred [MH97]. Structure learning is based on the old neuroscience idea that "**nodes that fire together should wire together**". To implement this, we run inference and check for cluster-word and cluster-cluster pairs that frequently turn on together. We then add an edge from parent to child if the link can significantly increase the probability of the child. Links that are not activated very often are pruned out. We initialize with one cluster per "document" (corresponding to a set of semantically related phrases). We then merge clusters A and B if A explains B's top words and vice versa. We can also discard clusters that are used too rarely.

[2] The original system, called "Phil", was developed by Georges Harik and Noam Shazeer,. It has been published as US Patent #8024372, "Method and apparatus for learning a probabilistic generative model for text", filed in 2004. Rephil is a more probabilistically sound version of the method, developed by Uri Lerner et al. The summary below is based on notes by Brian Milch (who also works at Google).

[3] AdSense is Google's system for matching web pages with content-appropriate ads in an automatic way, by extracting semantic keywords from web pages. These keywords play a role analogous to the words that users type in when searching; this latter form of information is used by Google's AdWords system. The details are secret, but [Lev11] gives an overview.

The model was trained on about 100 billion text snippets or search queries; this takes several weeks, even on a parallel distributed computing architecture. The resulting model contains 12 million word nodes and about 1 million latent cluster nodes. There are about 350 million links in the model, including many cluster-cluster dependencies. The longest path in the graph has length 555, so the model is quite deep.

Exact inference in this model is obviously infeasible. However note that most leaves will be off, since most words do not occur in a given query; such leaves can be analytically removed. We can also prune out unlikely hidden nodes by following the strongest links from the words that are on up to their parents to get a candidate set of concepts. We then perform iterative conditional modes (ICM) to form approximate inference. (ICM is a deterministic version of Gibbs sampling that sets each node to its most probable state given the values of its neighbors in its Markov blanket.) This continues until it reaches a local maximum. We can repeat this process a few times from random starting configurations. At Google, this can be made to run in 15 milliseconds!

30.2.8.9 Spectral methods

Recently, various methods have been developed that can recover the exact structure of the DAG, even in the presence of (a known number of) latent variables, under certain assumptions. In particular, identifiability results have been obtained for the following cases:

- If $\boldsymbol{x}$ contains 3 of more indepenent views of z [Goo74; AMR09; AHK12; HKZ12], sometimes called the **triad constraint**.
- If z is categorical, and $\boldsymbol{x}$ is a GMM with mixture components which depend on z [Ana+14].
- If $\boldsymbol{z}$ is composed of binary variables, and $\boldsymbol{x}$ is a set of noisy-OR CPDs [JHS13; Aro+16].

In terms of algorithms, most of these methods are not based on maximum likelihood, but instead use the method of moments and spectral methods. For details, see [Ana+14].

30.2.8.10 Constraint-based methods for learning ADMGs

An alternative to explicitly modeling latent variables is to marginalize them out, and work with acyclic directed mixed graphs (**??**). It is possible to perform Bayesian model selection for ADMGs, although the method is somewhat slow and complicated [SG09]. Alternatively, one can modify the PC/IC algorithm to learn an ADMG. This method is known as the **IC*** algorithm [Pea09, p52]; one can speed it up to get the **FCI** algorithm (FCI stands for "fast causal inference") [SGS00, p144].

Since there will inevitably be some uncertainty about edge orientations, due to Markov equivalence, the output of IC*/ FCI is not actually an ADMG, but is a closely related structured called a **partially oriented inducing path graph** [SGS00, p135] or a **marked pattern** [Pea09, p52]. Such a graph has 4 kinds of edges:

- A marked arrow $a \overset{*}{\rightarrow} b$ signifying a directed path from a to b.
- An unmarked arrow $a \rightarrow b$ signifying a directed path from a to b or a latent common cause $a \leftarrow L \rightarrow b$.
- A bidirected arrow $a \leftrightarrow b$ signifying a latent common causes $a \leftarrow L \rightarrow b$.
- An undirected edge $a - b$ signifying $a \leftarrow b$ or $a \rightarrow b$ or a latent common causes $a \leftarrow L \rightarrow b$.

IC*/ FCI is faster than Bayesian inference, but suffers from the same problems as the original IC/PC algorithm (namely, the need for a CI testing oracle, problems due to multiple testing, no probabilistic representation of uncertainty, etc.) Furthermore, by not explicitly representing the latent variables, the resulting model cannot be used for inference and prediction.

30.3 Learning undirected graph structures

In this section, we discuss how to learn the structure of undirected graphical models. On the one hand, this is easier than learning DAG structure because we don't need to worry about acyclicity. On the other hand, it

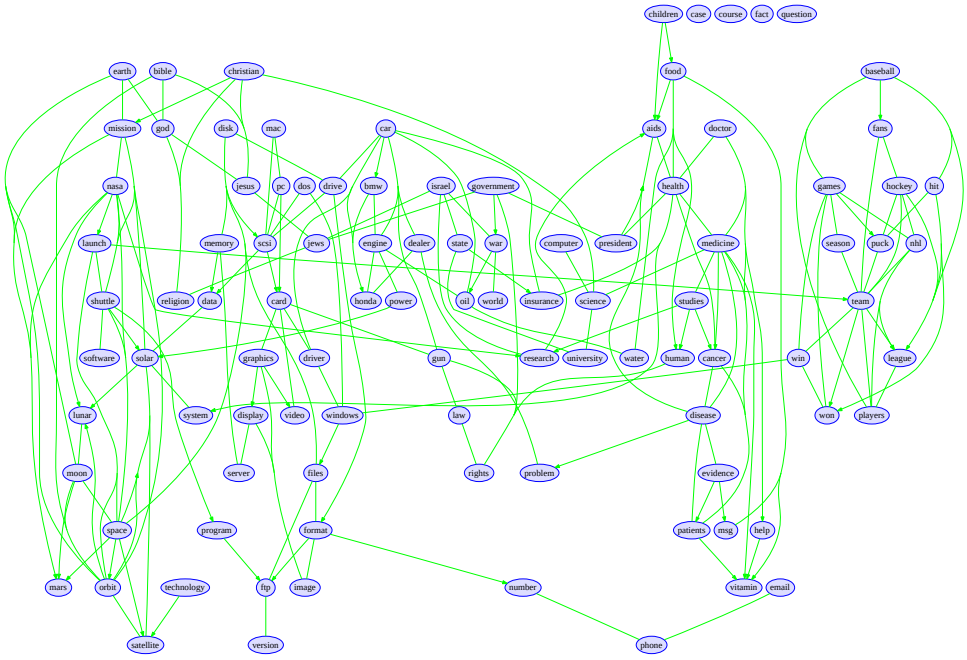


Figure 30.15: A dependency network constructed from the 20 newsgroup data. We show all edges with regression weight above 0.5 in the Markov blankets estimated by ℓ_1 penalized logistic regression. Undirected edges represent cases where a directed edge was found in both directions. From Figure 4.9 of [Sch10a]. Used with kind permission of Mark Schmidt.

is harder than learning DAG structure since the likelihood does not decompose (see **??**). This precludes the kind of local search methods (both greedy search and MCMC sampling) we used to learn DAG structures, because the cost of evaluating each neighboring graph is too high, since we have to refit each model from scratch (there is no way to incrementally update the score of a model). In this section, we discuss several solutions to this problem.

30.3.1 Dependency networks

A simple way to learn the structure of a UGM is to represent it is as a product of full conditionals:

$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_{d=1}^{D} p(x_d | \boldsymbol{x}_{-d}) \tag{30.43}$$

This expression is called the **pseudolikelihood**.

Such a collection of local distributions defines a model called a **dependency network** [Hec+00]. Unfortunately, a product of full conditionals which are independently estimated is not guaranteed to be consistent with any valid joint distribution. However, we can still use the model inside of a Gibbs sampler to approximate a joint distribution. This approach is sometimes used for data imputation [GR01].

However, the main use advantage of dependency networks is that we can use sparse regression techniques for each distribution $p(x_d|\boldsymbol{x}_{-d})$ to induce a sparse graph structure. For example, [Hec+00] use classification/regression trees, [MB06] use ℓ_1-regularized linear regression, [WRL06; WSD19] use ℓ_1-regularized logistic regression, [Dob09] uses Bayesian variable selection, etc.

Figure 30.15 shows a dependency network that was learned from the 20-newsgroup data using ℓ_1 regularized logistic regression, where the penalty parameter λ was chosen by BIC. Many of the words present in these estimated Markov blankets represent fairly natural associations (aids:disease, baseball:fans, bible:god, bmw:car, cancer:patients, etc.). However, some of the estimated statistical dependencies seem less intuitive, such as

baseball:windows and bmw:christian. We can gain more insight if we look not only at the sparsity pattern, but also the values of the regression weights. For example, here are the incoming weights for the first 5 words:

- **aids**: children (0.53), disease (0.84), fact (0.47), health (0.77), president (0.50), research (0.53)
- **baseball**: *christian* (-0.98), *drive* (-0.49), games (0.81), *god* (-0.46), *government* (-0.69), hit (0.62), *memory* (-1.29), players (1.16), season (0.31), *software* (-0.68), *windows* (-1.45)
- **bible**: *car* (-0.72), *card* (-0.88), christian (0.49), fact (0.21), god (1.01), jesus (0.68), orbit (0.83), *program* (-0.56), religion (0.24), version (0.49)
- **bmw**: car (0.60), *christian* (-11.54), engine (0.69), *god* (-0.74), *government* (-1.01), *help* (-0.50), *windows* (-1.43)
- **cancer**: disease (0.62), medicine (0.58), patients (0.90), research (0.49), studies (0.70)

Words in italic red have negative weights, which represents a dissociative relationship. For example, the model reflects that baseball:windows is an unlikely combination. It turns out that most of the weights are negative (1173 negative, 286 positive, 8541 zero) in this model.

[MB06] discuss theoretical conditions under which dependency networks using ℓ_1-regularized linear regression can recover the true graph structure, assuming the data was generated from a sparse Gaussian graphical model. We discuss a more general solution in Section 30.3.2.

30.3.2 Graphical lasso for GGMs

In this section, we consider the problem of learning the structure of undirected Gaussian graphical models (GGM)s. These models are useful, since there is a 1:1 mapping between sparse parameters and sparse graph structures. This allows us to extend the efficient techniques of ℓ_1 regularized estimation in **??** to the graph case; the resulting method is called the **graphical lasso** or **Glasso** [FHT08; MH12].

30.3.2.1 MLE for a GGM

Before discussing structure learning, we need to discuss parameter estimation. The task of computing the MLE for a (non-decomposable) GGM is called **covariance selection** [Dem72].

The log likelihood can be written as

$$\ell(\boldsymbol{\Omega}) = \log\det\boldsymbol{\Omega} - \mathrm{tr}(\mathbf{S}\boldsymbol{\Omega}) \tag{30.44}$$

where $\boldsymbol{\Omega} = \boldsymbol{\Sigma}^{-1}$ is the precision matrix, and $\mathbf{S} = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^{\mathsf{T}}$ is the empirical covariance matrix. (For notational simplicity, we assume we have already estimated $\hat{\boldsymbol{\mu}} = \overline{\boldsymbol{x}}$.) One can show that the gradient of this is given by

$$\nabla\ell(\boldsymbol{\Omega}) = \boldsymbol{\Omega}^{-1} - \mathbf{S} \tag{30.45}$$

However, we have to enforce the constraints that $\Omega_{st} = 0$ if $G_{st} = 0$ (structural zeros), and that $\boldsymbol{\Omega}$ is positive definite. The former constraint is easy to enforce, but the latter is somewhat challenging (albeit still a convex constraint). One approach is to add a penalty term to the objective if $\boldsymbol{\Omega}$ leaves the positive definite cone; this is the approach used in [DVR08]. Another approach is to use a coordinate descent method, described in [HTF09, p633].

Interestingly, one can show that the MLE must satisfy the following property: $\Sigma_{st} = S_{st}$ if $G_{st} = 1$ or $s = t$, i.e., the covariance of a pair that are connected by an edge must match the empirical covariance. In addition, we have $\Omega_{st} = 0$ if $G_{st} = 0$, by definition of a GGM, i.e., the precision of a pair that are not connected must be 0. We say that $\boldsymbol{\Sigma}$ is a positive definite **matrix completion** of $\mathbf{S}$, since it retains as many of the entries in $\mathbf{S}$ as possible, corresponding to the edges in the graph, subject to the required sparsity pattern on $\boldsymbol{\Sigma}^{-1}$, corresponding to the absent edges; the remaining entries in $\boldsymbol{\Sigma}$ are filled in so as to maximize the likelihood.

Let us consider a worked example from [HTF09, p652]. We will use the following adjacency matrix, representing the cyclic structure, $X_1 - X_2 - X_3 - X_4 - X_1$, and the following empirical covariance matrix:

$$\mathbf{G} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}, \quad \mathbf{S} = \begin{pmatrix} 10 & 1 & 5 & 4 \\ 1 & 10 & 2 & 6 \\ 5 & 2 & 10 & 3 \\ 4 & 6 & 3 & 10 \end{pmatrix} \tag{30.46}$$

The MLE is given by

$$\boldsymbol{\Sigma} = \begin{pmatrix} 10.00 & 1.00 & \mathbf{1.31} & 4.00 \\ 1.00 & 10.00 & 2.00 & \mathbf{0.87} \\ \mathbf{1.31} & 2.00 & 10.00 & 3.00 \\ 4.00 & \mathbf{0.87} & 3.00 & 10.00 \end{pmatrix}, \quad \boldsymbol{\Omega} = \begin{pmatrix} 0.12 & -0.01 & \mathbf{0} & -0.05 \\ -0.01 & 0.11 & -0.02 & \mathbf{0} \\ \mathbf{0} & -0.02 & 0.11 & -0.03 \\ -0.05 & \mathbf{0} & -0.03 & 0.13 \end{pmatrix} \tag{30.47}$$

(See ggmFitDemo.py for the code to reproduce these numbers, using the coordinate descent algorithm from [FHT08].) The constrained elements in $\boldsymbol{\Omega}$, and the free elements in $\boldsymbol{\Sigma}$, both of which correspond to absent edges, have been highlighted.

30.3.2.2 Promoting sparsity

We now discuss one way to learn a sparse Gaussian MRF structure, which exploits the fact that there is a 1:1 correspondence between zeros in the precision matrix and absent edges in the graph. This suggests that we can learn a sparse graph structure by using an objective that encourages zeros in the precision matrix. By analogy to lasso (see **??**), one can define the following ℓ_1 penalized NLL:

$$J(\boldsymbol{\Omega}) = -\log\det\boldsymbol{\Omega} + \mathrm{tr}(\mathbf{S}\boldsymbol{\Omega}) + \lambda||\boldsymbol{\Omega}||_1 \tag{30.48}$$

where $||\boldsymbol{\Omega}||_1 = \sum_{j,k} |\omega_{jk}|$ is the 1-norm of the matrix. This is called the **graphical lasso** or **Glasso**.

Although the objective is convex, it is non-smooth (because of the non-differentiable ℓ_1 penalty) and is constrained (because $\boldsymbol{\Omega}$ must be a positive definite matrix). Several algorithms have been proposed for optimizing this objective [YL07; BGd08; DGK08], although arguably the simplest is the one in [FHT08], which uses a coordinate descent algorithm similar to the shooting algorithm for lasso. An even faster method, based on soft thresholding, is described in [FS18; FZS18].

As an example, let us apply the method to the flow cytometry dataset from [Sac+05]. A discretized version of the data is shown in Figure 30.19(a). Here we use the original continuous data. However, we are ignoring the fact that the data was sampled under intervention. In **??**, we illustrate the graph structures that are learned as we sweep λ from 0 to a large value. These represent a range of plausible hypotheses about the connectivity of these proteins.

It is worth comparing this with the DAG that was learned in Figure 30.19(b). The DAG has the advantage that it can easily model the interventional nature of the data, but the disadvantage that it cannot model the feedback loops that are known to exist in this biological pathway (see the discussion in [SM09]). Note that the fact that we show many UGMs and only one DAG is incidental: we could easily use BIC to pick the "best" UGM, and conversely, we could easily display several DAG structures, sampled from the posterior.

30.3.3 Graphical lasso for discrete MRFs/CRFs

It is possible to extend the graphical lasso idea to the discrete MRF and CRF case. However, now there is a set of parameters associated with each edge in the graph, so we have to use the graph analog of group lasso (see **??**). For example, consider a pairwise CRF with ternary nodes, and node and edge potentials given by

$$\psi_t(y_t, \boldsymbol{x}) = \begin{pmatrix} \boldsymbol{v}_{t1}^\mathsf{T}\boldsymbol{x} \\ \boldsymbol{v}_{t2}^\mathsf{T}\boldsymbol{x} \\ \boldsymbol{v}_{t3}^\mathsf{T}\boldsymbol{x} \end{pmatrix}, \quad \psi_{st}(y_s, y_t, \boldsymbol{x}) = \begin{pmatrix} \boldsymbol{w}_{t11}^\mathsf{T}\boldsymbol{x} & \boldsymbol{w}_{st12}^\mathsf{T}\boldsymbol{x} & \boldsymbol{w}_{st13}^\mathsf{T}\boldsymbol{x} \\ \boldsymbol{w}_{st21}^\mathsf{T}\boldsymbol{x} & \boldsymbol{w}_{st22}^\mathsf{T}\boldsymbol{x} & \boldsymbol{w}_{st23}^\mathsf{T}\boldsymbol{x} \\ \boldsymbol{w}_{st31}^\mathsf{T}\boldsymbol{x} & \boldsymbol{w}_{st32}^\mathsf{T}\boldsymbol{x} & \boldsymbol{w}_{st33}^\mathsf{T}\boldsymbol{x} \end{pmatrix} \tag{30.49}$$

where we assume $\boldsymbol{x}$ begins with a constant 1 term, to account for the offset. (If $\boldsymbol{x}$ only contains 1, the CRF reduces to an MRF.) Note that we may choose to set some of the $\boldsymbol{v}_{tk}$ and $\boldsymbol{w}_{stjk}$ weights to 0, to ensure identifiability, although this can also be taken care of by the prior.

To learn sparse structure, we can minimize the following objective:

$$J = -\sum_{i=1}^{N}\left[\sum_{t}\log\psi_t(y_{it},\boldsymbol{x}_i,\boldsymbol{v}_t)+\sum_{s=1}^{V}\sum_{t=s+1}^{V}\log\psi_{st}(y_{is},y_{it},\boldsymbol{x}_i,\boldsymbol{w}_{st})\right] + \lambda_1\sum_{s=1}^{V}\sum_{t=s+1}^{V}||\boldsymbol{w}_{st}||_p+\lambda_2\sum_{t=1}^{V}||\boldsymbol{v}_t||_2^2 \tag{30.50}$$

where $||\boldsymbol{w}_{st}||_p$ is the p-norm; common choices are $p=2$ or $p=\infty$, as explained in **??**. This method of CRF structure learning was first suggested in [Sch+08]. (The use of ℓ_1 regularization for learning the structure of binary MRFs was proposed in [LGK06].)

Although this objective is convex, it can be costly to evaluate, since we need to perform inference to compute its gradient, as explained in **??** (this is true also for MRFs), due to the global partition function. We should therefore use an optimizer that does not make too many calls to the objective function or its gradient, such as the projected quasi-Newton method in [Sch+09]. In addition, we can use approximate inference, such as loopy belief propagation (**??**), to compute an approximate objective and gradient more quickly, although this is not necessarily theoretically sound.

Another approach is to apply the group lasso penalty to the pseudo-likelihood discussed in **??**. This is much faster, since inference is no longer required [HT09]. Figure 30.16 shows the result of applying this procedure to the 20-newsgroup data, where y_{it} indicates the presence of word t in document i, and $\boldsymbol{x}_i = 1$ (so the model is an MRF).

For a more recent approach to learning sparse discrete PGM-U structures, based on sparse full conditionals, see the GRISE (Generalized Regularized Interaction Screening Estimator) method of [VML19], which takes polynomial time, yet its sample complexity is close to the information-theoretic lower bounds [Lok+18].

30.3.4 Bayesian inference for undirected graph structures

Although the graphical lasso is reasonably fast, it only gives a point estimate of the structure. Furthermore, it is not model-selection consistent [Mei05], meaning it cannot recover the true graph even as $N \to \infty$. It would be preferable to integrate out the parameters, and perform posterior inference in the space of graphs, i.e., to compute $p(G|\mathcal{D})$. We can then extract summaries of the posterior, such as posterior edge marginals, $p(G_{ij}=1|\mathcal{D})$, just as we did for DAGs. In this section, we discuss how to do this.

If the graph is decomposable, and if we use conjugate priors, we can compute the marginal likelihood in closed form [DL93]. Furthermore, we can efficiently identify the decomposable neighbors of a graph [TG09], i.e., the set of legal edge additions and removals. This means that we can perform relatively efficient stochastic local search to approximate the posterior (see e.g. [GG99; Arm+08; SC08]).

However, the restriction to decomposable graphs is rather limiting if one's goal is knowledge discovery, since the number of decomposable graphs is much less than the number of general undirected graphs.[4]

A few authors have looked at Bayesian inference for GGM structure in the non-decomposable case (e.g., [DGR03; WCK03; Jon+05]), but such methods cannot scale to large models because they use an expensive Monte Carlo approximation to the marginal likelihood [AKM05]. [LD08] suggested using a Laplace approximation. This requires computing the MAP estimate of the parameters for $\boldsymbol{\Omega}$ under a G-Wishart prior [Rov02]. In [LD08], they used the iterative proportional scaling algorithm [SK86; HT08] to find the mode. However, this is very slow, since it requires knowing the maximal cliques of the graph, which is NP-hard in general.

[4] The number of decomposable graphs on V nodes, for $V = 2,\ldots,8$, is as follows ([Arm05, p158]): 2; 8; 61; 822; 18,154; 61,7675; 30,888,596. If we divide these numbers by the number of undirected graphs, which is $2^{V(V-1)/2}$, we find the ratios are: 1, 1, 0.95, 0.8, 0.55, 0.29, 0.12. So we see that decomposable graphs form a vanishing fraction of the total hypothesis space.

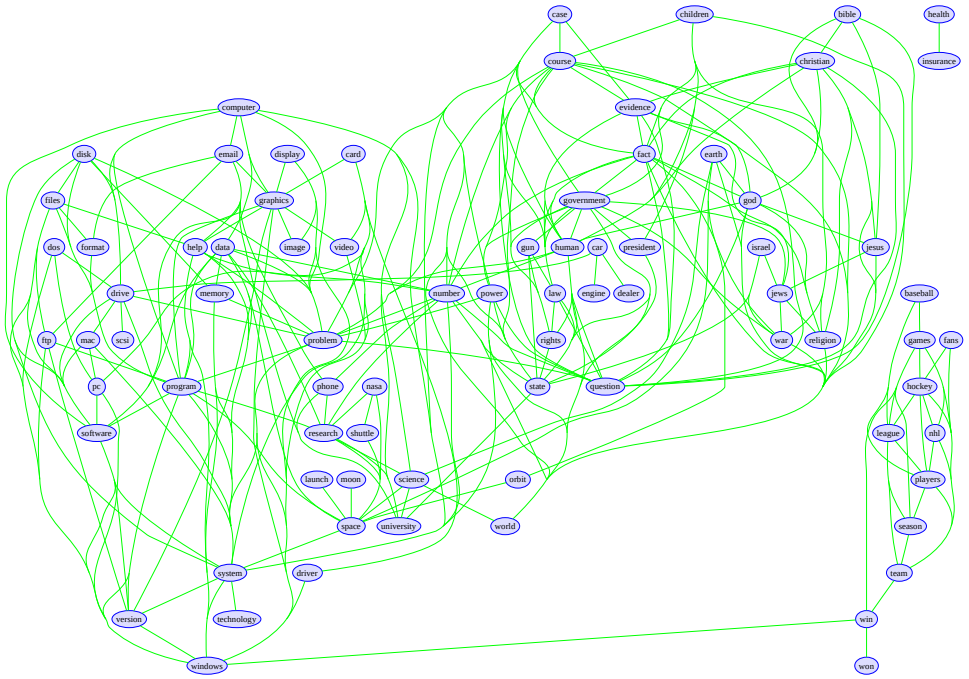


Figure 30.16: An MRF estimated from the 20-newsgroup data using group ℓ_1 regularization with $\lambda = 256$. Isolated nodes are not plotted. From Figure 5.9 of [Sch10a]. Used with kind permission of Mark Schmidt.

In [Mog+09], a much faster method is proposed. In particular, they modify the gradient-based methods from Section 30.3.2.1 to find the MAP estimate; these algorithms do not need to know the cliques of the graph. A further speedup is obtained by just using a diagonal Laplace approximation, which is more accurate than BIC, but has essentially the same cost. This, plus the lack of restriction to decomposable graphs, enables fairly fast stochastic search methods to be used to approximate $p(G|\mathcal{D})$ and its mode. This approach significantly outperfomed graphical lasso, both in terms of predictive accuracy and structural recovery, for a comparable computational cost.

30.4 Learning causal DAGs

Causal reasoning (which we discuss in more detail in **??**) relies on knowing the underlying structure of the DAG (although [JZB19] shows how to answer some queries if we just know the graph up to Markov equivalence). Learning this structure is called **causal discovery** (see e.g., [GZS19]).

If we just have two variables, we need to know if the causal model should be written as $X \to Y$ or $X \leftarrow Y$. both of these models are Markov equivalent (Section 30.2.2), meaning they cannot be distinguished from observational data, yet they make very different causal predictions. We discuss how to learn cause-effect pairs in Section 30.4.1.

When we have more than 2 variables, we need to consider more general techniques. In Section 30.2, we discuss how to learn a DAG structure from observational data using likelihood based methods, and hypothesis testing methods. However, these approaches cannot distinguish between models that are Markov equivalent, so we need to perform interventions to reduce the size of the equivalence class [Sol19]. We discuss some suitable methods in Section 30.4.2.

The above techniques assume that the causal variables of interest (e.g., cancer rates, smoking rates) can be measured directly. However, in many ML problems, the data is much more "low level". For example, consider trying to learn a causal model of the world from raw pixels. We briefly disuss this topic in in Section 30.4.3.

For more details on causal discovery methods, see e.g., [Ebe17; PJS17; HDMM18; Guo+21].

30.4.1 Learning cause-effect pairs

If we only observe a pair of variables, we cannot use methods discussed in Section 30.2 to learn graph structure, since such methods are based on conditional independence tests, which need at least 3 variables. However, intuitively, we should still be able to learn causal relationships in this case. For example, we know that altitude X causes temperature Y and not vice versa. For example, suppose we measure X and Y in two different countries, say the Netherlands (low altitude) and Switzerland (high altitude). If we represent the joint distribution as $p(X, Y) = p(X)p(Y|X)$, we find that the $p(Y|X)$ distribution is stable across the two populations, while $p(X)$ will change. However, if we represent the joint distribution as $p(X, Y) = p(Y)p(X|Y)$, we find that *both* $p(Y)$ and $p(X|Y)$ need to change across populations, so both of the corresponding distributions will be more "complicated" to capture this non-stationarity in the data. In this section, we discuss some approaches that exploit this idea. Our presentation is based on [PJS17]. (See [Moo+16] for more details.)

30.4.1.1 Algorithmic information theory

Suppose $X \in \{0, 1\}$ and $Y \in \mathbb{R}$ and we represent the joint $p(x, y)$ using

$$p(x, y) = p(x)p(y|x) = \text{Ber}(x|\theta)\mathcal{N}(y|\mu_x, 1) \tag{30.51}$$

We can equally well write this in the following form [Daw02, p165]:

$$p(x, y) = p(y)p(x|y) = [\theta\mathcal{N}(y|\mu_1, 1) + (1 - \theta)\mathcal{N}(y|\mu_2, 1)]\text{Ber}(x|\sigma(\alpha + \beta y)) \tag{30.52}$$

where $\alpha = \text{logit}(\theta) + \mu_2^2 - \mu_1^2$ and $\beta = \mu_1 - \mu_2$. We can plausibly argue that the first model, which corresponds to $X \to Y$, is more likely to be correct, since it is consists of two simple distributions that seem to be rather generic. By contrast, in Equation (30.52), the distribution of $p(Y)$ is more complex, and seems to be dependent on the specific form of $p(X|Y)$.

[JS10] show how to formalize this intuition using **algorithmic information theory**. In particular, they say that X causes Y if the *distributions* P_X and $P_{Y|X}$ (not the random variables X and Y) are **algorithmically independent**. To define this, let $P_X(X)$ be the distribution induced by $f_x(X, U_X)$, where U_X is a bit string, and f_X is represented by a **Turing machine**. Define $P_{Y|X}$ analogously. Finally, let $K(s)$ be the **Kolmogorov complexity** of bit string s, i.e., the length of the shortest program that would generate s using a universal Turing machine. We say that P_X and $P_{Y|X}$ are algorithmically independent if

$$K(P_{X,Y}) = K(P_X) + K(P_{Y|X}) \tag{30.53}$$

Unfortunately, there is no algorithm to compute the Kolmogorov complexity, so this approach is purely conceptual. In the sections below, we discuss some more practical metrics.

30.4.1.2 Additive noise models

A generic two-variable SCM of the form $X \to Y$ requires specifying the function $X = f_X(U_X)$, the distribution of U_X, the function $Y = f_Y(X, U_Y)$, and the distribution of U_Y. We can simplify our notation by letting $X = U_x$ and defining $p(X)$ directly, and defining $Y = f_Y(X, U_Y) = f(X, U)$, where U is a noise term.

In general, such a model is not identifiable from a finite dataset. For example, we can imagine that the value of U can be used to select between different functional mappings, $Y = f(X, U = u) = f_u(X)$. Since U is not observed, the induced distribution will be a mixture of different mappings, and it will generally be impossible to disentangle. For example, consider the case where X and U are Bernoulli random variables, and U selects between the functions $Y = f_{\text{id}}(X) = \mathbb{I}(Y = X)$ and $Y = f_{\text{neg}}(X) = \mathbb{I}(Y \neq X)$. In this case, the induced distribution $p(Y)$ is uniform, independent of X, even though we have the structure $X \to Y$.

The above concerns motivate the desire to restrict the flexibility of the functions at each node. One natural family is **additive noise models** (**ANM**), where we assume each variable has the following dependence on its parents [Hoy+09]:

$$X_i = f_i(X_{\text{pa}_i}) + U_i \tag{30.54}$$

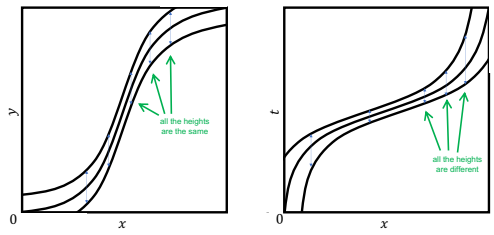


Figure 30.17: Signature of X causing Y. Left: If we try to predict Y from X, the residual error (noise term, shown by vertical arrows) is independent of X. Right: If we try to predict X from Y, the residual error is not constant. From Figure 8.8 of [Var21]. Used with kind permission of Kush Varshney.

In the case of two variables, we have $Y = f(X) + U$. If X and U are both Gaussian, and f is linear, the system defines a jointly Gaussian distribution $p(X, Y)$, as we discussed in **??**. This is symmetric, and prevents us distinguishing $X \to Y$ from $Y \to X$. However, if we let f be nonlinear, and/or let X or U be non-Gaussian, we can distinguish $X \to Y$ from $Y \to X$, as we discuss below.

30.4.1.3 Nonlinear additive noise models

Suppose $p_{Y|X}$ is an additive noise model (possibly Gaussian noise) where f is a nonlinear function. In this case, we will not, in general, be able to create an ANM for $p_{X|Y}$. Thus we can determine whether $X \to Y$ or vice versa as follows: we fit a (nonlinear) regression model for $X \to Y$, and then check if the residual error $Y - \hat{f}_Y(X)$ is independent of X; we then repeat the procedure swapping the roles of X and Y. The theory [PJS17] says that the independence test will only pass for the causal direction. See Figure 30.17 for an illustration.

30.4.1.4 Linear models with non-Gaussian noise

If the function mapping from X to Y is linear, we cannot tell if $X \to Y$ or $Y \to X$ if we assume Gaussian noise. This is apparent from the symmetry of Figure 30.17 in the linear case. However, by combining linear models with non-Gaussian noise, we can recover identifiability.

For example, consider the ICA model from **??**. This is a simple linear model of the form $\boldsymbol{y} = \mathbf{A}\boldsymbol{x}$, where $p(\boldsymbol{x})$ has a non-Gaussian distribution, and $p(\boldsymbol{y}|\boldsymbol{x})$ is a degenerate distribution, since we assume the observation model is deterministic (noise-free). In the ICA case, we can uniquely identify the parameters $\mathbf{A}$ and the corresponding latent source $\boldsymbol{x}$. This lets us distinguish the $X \to Y$ model from the $Y \to X$ model. (The intuition behind this method is that linear combinations of random variables tend towards a Gaussian distribution (by the central limit theorem), so if $X \to Y$, then $p(Y)$ will "look more Gaussian" than $p(X)$.)

Another setting in which we can distinguish the direction of the arrow is when we have non-Gaussian observation noise, i.e., $\boldsymbol{y} = \mathbf{A}\boldsymbol{x} + U_Y$, where U_Y is non-Gaussian. This is an example of a "linear non-Gaussian acyclic model" (**LiNGAM**) [Shi+06]. The non-Gaussian additive noise results in the induced distributions $p(X, Y)$ being different depending on whether $X \to Y$ or $Y \to X$.

30.4.1.5 Information-geometric causal inference

An alternative approach, known as **information-geometric causal inference**, or **IGCI**, was proposed in [Dan+10; Jan+12]. In this method, we assume f is a deterministic strictly monotonic function on $[0, 1]$, with $f(0) = 0$ and $f(1) = 1$, and there is no observation noise, so $Y = f(X)$. If X has the distribution $p(X)$, then the shape of the induced distribution $p(Y)$ will depend on the form of the function f, as illustrated in Figure 30.18. Intuitively, the peaks of $p(Y)$ will occur in regions where f has small slope, and thus f^{-1} has

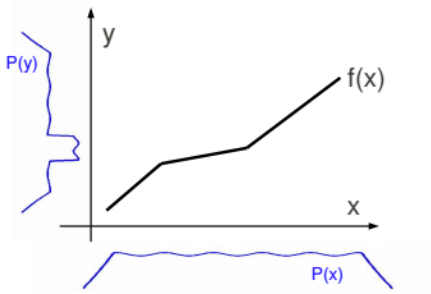


Figure 30.18: Illustration of information-geometric causal inference for $Y = f(X)$. The density of the effect $p(Y)$ tends to be high in regions where f is flat (and hence f^{-1} is steep). From Figure 4 of [Jan+12].

large slope. Thus $p_Y(Y)$ and $f^{-1}(Y)$ will depend on each other, whereas $p_X(X)$ and $f(X)$ do not (since we assume the distribution of causes is independent of the causal mechanism).

More precisely, let the functions $\log f'$ (the log of the derivative function) and p_X be viewed as random variables on the probability space $[0, 1]$ with a uniform distrbution. We say $p_{X,Y}$ satisfies an IGCI model if f is a mapping as above, and the following independence criterion holds: $\text{Cov}\left[\log f', p_X\right] = 0$, where

$$\text{Cov}\left[\log f', p_X\right] = \int_0^1 \log f'(x) p_X(x) dx - \int_0^1 \log f'(x) dx \int_0^1 p_X(x) dx \tag{30.55}$$

where $\int_0^1 p_X(x)dx = 1$. One can show that the inverse function f^{-1} satisfies $\text{Cov}\left[\log f^{-1'}, p_Y\right] \geq 0$, with equality iff f is linear.

This can be turned into an empirical test as follows. Define

$$C_{X \to Y} = \int_0^1 \log f'(x) p(x) dx \approx \frac{1}{N-1} \sum_{j=1}^{N-1} \log \frac{|y_{j+1} - y_j|}{|x_{j+1} - x_j|} \tag{30.56}$$

where $x_1 < x_2 \cdots x_N$ are the observed x-values in increasing order. The quantity $C_{Y \to X}$ is defined analogously. We then choose $X \to Y$ as the model whenever $\hat{C}_{X \to Y} < \hat{C}_{Y \to X}$. This is called the **slope based approach** to IGCI.

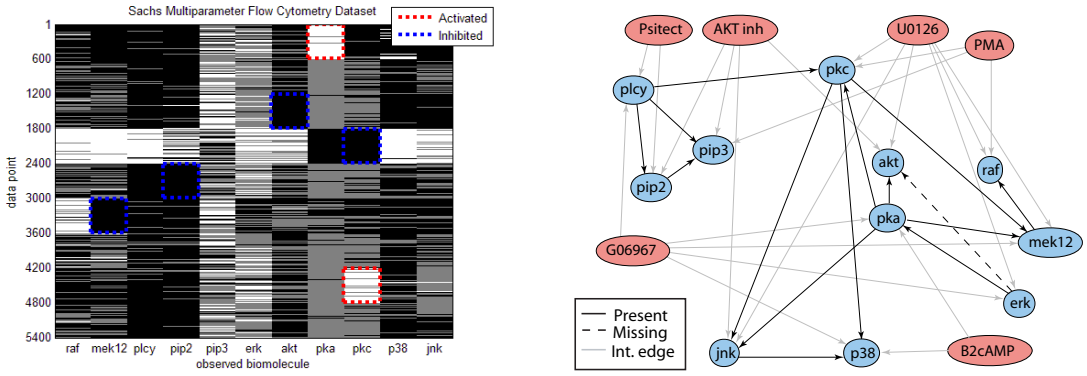
One can also show that an IGCI model satisfies the property that $H(X) \leq H(Y)$, where $H()$ is the differential entropy. Intuitively, the reason is that applying a nonlinear function f to p_X can introduce additional irregularities, thus making p_Y less uniform that p_X. This is illustrated in Figure 30.18. We can then choose between $X \to Y$ and $X \leftarrow Y$ based on the difference in estimated entropies.

An empirical comparison of the slope-based and entropy-based approaches to IGCI can be found in [Moo+16].

30.4.2 Learning causal DAGs from interventional data

In Section 30.2, we discuss how to learn a DAG structure from observational data, using either likelihood-based (Bayesian) methods of model selection, or constraint-based (frequentist) methods. (See [Tu+19] for a recent empirical comparison of such methods applied to a medical simulator.) However, such approaches cannot distinguish between models that are Markov equivalent, and thus the output may not be sufficient to answer all causal queries of interest.

To distinguish DAGs within the same Markov equivalence class, we can use **interventional data**, where certain variables have been set, and the consequences have been measured. In particular, we can modify the standard likelihood-based DAG learning method discussed in Section 30.2 to take into account the fact that



(a) *(b)*

Figure 30.19: (a) A design matrix consisting of 5400 data points (rows) measuring the status (using flow cytometry) of 11 proteins (columns) under different experimental conditions. The data has been discretized into 3 states: low (black), medium (grey) and high (white). Some proteins were explicitly controlled using activating or inhibiting chemicals. (b) A directed graphical model representing dependencies between various proteins (blue circles) and various experimental interventions (pink ovals), which was inferred from this data. We plot all edges for which $p(G_{st} = 1|\mathcal{D}) > 0.5$. Dotted edges are believed to exist in nature but were not discovered by the algorithm (1 false negative). Solid edges are true positives. The light colored edges represent the effects of intervention. From Figure 6d of [EM07].

the data generating mechanism has been changed. For example, if $\theta_{ijk} = p(X_i = j|X_{\text{pa}(i)} = k)$ is a CPT for node i, then when we compute the sufficient statistics $N_{ijk} = \sum_n \mathbb{I}\left(X_{ni} = j, X_{n,\text{pa}(i)} = k\right)$, we exclude cases n where X_i was set externally by intervention, rather than sampled from θ_{ijk}. This technique was first proposed in [CY99], and corresponds to Bayesian parameter inference from a set of mutiliated models with shared parameters.

The preceding method assumes that we use **perfect interventions**, where we deterministically set a variable to a chosen value. In reality, experimenters can rarely control the state of individual variables. Instead, they can perform actions which may affect many variables at the same time. (This is sometimes called a "**fat hand intervention**", by analogy to an experiment where someone tries to change a single component of some system (e.g., electronic circuit), but accidently touching multiple components and thereby causing various side effects.) We can model this by adding the intervention nodes to the DAG (**??**), and then learning a larger augmented DAG structure, with the constraint that there are no edges between the intervention nodes, and no edges from the "regular" nodes back to the intervention nodes.

For example, suppose we perturb various proteins in a cellular signalling pathway, and measure the resulting phosphorylation status using a technique such as flow cytometry, as in [Sac+05]. An example of such a dataset is shown in Figure 30.19(a). Figure 30.19(b) shows the augmented DAG that was learned from the interventional flow cytometry data depicted in Figure 30.19(a). In particular, we plot the median graph, which includes all edges for which $p(G_{ij} = 1|\mathcal{D}) > 0.5$. These were computed using the exact algorithm of [Koi06]. See [EM07] for details.

Since interventional data can help to uniquely identify the DAG, it is natural to try to choose the optimal set of interventions so as to discover the graph structure with as little data as possible. This is a form of active learning or experiment design, and is similar to what scientists do. See e.g., [Mur01; HG09; KB14; HB14; Mue+17] for some approaches to this problem.

30.4.3 Learning from low-level inputs

In many problems, the available data is quite "low level", such as pixels in an image, and is believed to be generated by some "higher level" latent causal factors, such as objects interacting in a scene. Learning causal models of this type is known as **causal representation learning**, and combines the causal discovery methods discussed in this section with techniques from latent variable modeling (e.g., VAEs, **??**) and representation

learning (**??**). For more details, see e.g., [CEP17; Sch+21].

Chapter 31

Non-parametric Bayesian models

Chapter 32

Representation learning

Chapter 33

Interpretability

Part VI

Decision making

Chapter 34

Multi-step decision problems

Chapter 35

Reinforcement learning

Chapter 36

Causality

Bibliography

[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.

[AEM18] Ö. D. Akyildiz, V. Elvira, and J. Miguez. "The Incremental Proximal Method: A Probabilistic Perspective". In: *ICASSP*. 2018.

[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. Kakade. "A method of moments for mixture models and hidden Markov models". In: *COLT*. 2012.

[AKM05] A. Atay-Kayis and H. Massam. "A Monte Carlo method for computing the marginal likelihood in nondecomposable Gaussian graphical models". In: *Biometrika* 92 (2005), pp. 317–335.

[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).

[AKZK19] B. Amos, V. Koltun, and J Zico Kolter. "The Limited Multi-Label Projection Layer". In: (June 2019). arXiv: 1906.08707 [cs.LG].

[ALK06] C. Albers, M. Leisink, and H. Kappen. "The Cluster Variation Method for Efficient Linkage Analysis on Extended Pedigrees". In: *BMC Bioinformatics* 7 (2006).

[AMR09] E. S. Allman, C. Matias, and J. A. Rhodes. "Identifiability of parameters in latent structure models with many observed variables". en. In: *Ann. Stat.* 37.6A (Dec. 2009), pp. 3099–3132.

[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.

[AQ+20] M. A. A. Al-Qaness, A. A. Ewees, H. Fan, and M. Abd El Aziz. "Optimization Method for Forecasting Confirmed Cases of COVID-19 in China". en. In: *J. Clinical Medicine* 9.3 (Mar. 2020).

[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.

[Arm+08] H. Armstrong, C. Carter, K. Wong, and R. Kohn. "Bayesian Covariance Matrix Estimation using a Mixture of Decomposable Graphical Models". In: *Statistics and Computing* (2008), pp. 1573–1375.

[Aro+13] S. Arora, R. Ge, Y. Halpern, D. Mimno, A. Moitra, D. Sontag, Y. Wu, and M. Zhu. "A Practical Algorithm for Topic Modeling with Provable Guarantees". In: *ICML*. 2013.

[Aro+16] S. Arora, R. Ge, T. Ma, and A. Risteski. "Provable learning of Noisy-or Networks". In: (2016). arXiv: 1612.08795 [cs.LG].

[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).

[AY19] B. Amos and D. Yarats. "The Differentiable Cross-Entropy Method". In: (Sept. 2019). arXiv: 1909.12830 [cs.LG].

[Bac+15] S. H. Bach, M. Broecheler, B. Huang, and L. Getoor. "Hinge-Loss Markov Random Fields and Probabilistic Soft Logic". In: (2015). arXiv: 1505.04406 [cs.LG].

[Bal17] S. Baluja. "Learning deep models of optimization landscapes". In: *IEEE Symposium Series on Computational Intelligence (SSCI)* (2017).

[BB12] J. Bergstra and Y. Bengio. "Random Search for Hyper-Parameter Optimization". In: *JMLR* 13 (2012), pp. 281–305.

[BBZ17] T. Bartz-Beielstein and M. Zaefferer. "Model-based Methods for Continuous and Discrete Global Optimization". In: *Appl. Soft Comput.* 55.C (June 2017), pp. 154–167.

[BC95] S. Baluja and R. Caruana. "Removing the Genetics from the Standard Genetic Algorithm". In: *ICML*. 1995, pp. 38–46.

[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].

[BD03] S. G. Bottcher and C. Dethlefsen. "deal: A Package for Learning Bayesian Networks". In: *J. of Statistical Software* 8.20 (2003).

[BD97] S. Baluja and S. Davies. "Using Optimal Dependency-Trees for Combinatorial Optimization: Learning the Structure of the Search Space". In: *ICML*. 1997.

[Ber15] D. P. Bertsekas. "Incremental Gradient, Subgradient, and Proximal Methods for Convex Optimization: A Survey". In: (July 2015). arXiv: 1507.01030 [cs.SY].

[BF02] Y. Barash and N. Friedman. "Context-specific Bayesian clustering for gene expression data". In: *J. Comp. Bio.* 9 (2002), pp. 169–191.

[BG06] M. Beal and Z. Ghahramani. "Variational Bayesian Learning of Directed Graphical Models with Hidden Variables". In: *Bayesian Analysis* 1.4 (2006).

[BGd08] O. Banerjee, L. E. Ghaoui, and A. d'Aspremont. "Model selection through sparse maximum likelihood estimation for multivariate Gaussian or binary data". In: *JMLR* 9 (2008), pp. 485–516.

[BGHM17] J. Boyd-Graber, Y. Hu, and D. Mimno. "Applications of Topic Models". In: *Foundations and Trends® in Information Retrieval* 11.2-3 (2017), pp. 143–296.

[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.

[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.

[BJV97] J. S. D. Bonet, C. L. I. Jr., and P. A. Viola. "MIMIC: Finding Optima by Estimating Probability Densities". In: *NIPS*. MIT Press, 1997, pp. 424–430.

[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.

[BK10] R. Bardenet and B. Kegl. "Surrogating the surrogate: accelerating Gaussian-process-based global optimization with a mixture cross-entropy algorithm". In: *ICML*. 2010.

[BKR11] A. Blake, P. Kohli, and C. Rother, eds. *Advances in Markov Random Fields for Vision and Image Processing*. MIT Press, 2011.

[BL06a] D. Blei and J. Lafferty. "Dynamic topic models". In: *ICML*. 2006, pp. 113–120.

[BL06b] K. Bryan and T. Leise. "The $25,000,000,000 Eigenvector: The Linear Algebra behind Google". In: *SIAM Review* 48.3 (2006).

[BL07] D. Blei and J. Lafferty. "A Correlated Topic Model of "Science"". In: *Annals of Applied Stat.* 1.1 (2007), pp. 17–35.

[Ble12] D. M. Blei. "Probabilistic topic models". In: *Commun. ACM* 55.4 (2012), pp. 77–84.

[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.

[Boe+05] P.-T. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. "A Tutorial on the Cross-Entropy Method". en. In: *Ann. Oper. Res.* 134.1 (Feb. 2005), pp. 19–67.

[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.

[Bro+20] D. Brookes, A. Busia, C. Fannjiang, K. Murphy, and J. Listgarten. "A view of estimation of distribution algorithms through the lens of expectation-maximization". In: *GECCO*. GECCO '20. Cancún, Mexico: Association for Computing Machinery, July 2020, pp. 189–190.

[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.

[BVZ01] Y. Boykov, O. Veksler, and R. Zabih. "Fast Approximate Energy Minimization via Graph Cuts". In: *IEEE PAMI* 23.11 (2001).

[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.

[CEP17] K. Chalupka, F. Eberhardt, and P. Perona. "Causal feature learning: an overview". In: *Behaviormetrika* 44.1 (Jan. 2017), pp. 137–164.

[CGJ17] Y. Cherapanamjeri, K. Gupta, and P. Jain. "Nearly-optimal Robust Matrix Completion". In: *ICML*. 2017.

[CH92] G. Cooper and E. Herskovits. "A Bayesian method for the induction of probabilistic networks from data". In: *Machine Learning* 9 (1992), pp. 309–347.

[CH97] D. Chickering and D. Heckerman. "Efficient approximations for the marginal likelihood of incomplete data given a Bayesian network". In: *Machine Learning* 29 (1997), pp. 181–212.

[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].

[Chi02] D. M. Chickering. "Optimal structure identification with greedy search". In: *Journal of Machine Learning Research* 3 (2002), pp. 507–554.

[Chi96] D. Chickering. "Learning Bayesian networks is NP-Complete". In: *AI/Stats V*. 1996.

[CHM97] D. M. Chickering, D. Heckerman, and C. Meek. "A Bayesian Approach to Learning Bayesian Networks with Local Structure". In: *UAI*. UAI'97. San Francisco, CA, USA, 1997, pp. 80–89.

[Cho+11] M. Choi, V. Tan, A. Anandkumar, and A. Willsky. "Learning Latent Tree Graphical Models". In: *JMLR* (2011).

[CL68] C. K. Chow and C. N. Liu. "Approximating discrete probability distributions with dependence trees". In: *IEEE Trans. on Info. Theory* 14 (1968), pp. 462–67.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *An Introduction to Algorithms*. MIT Press, 1990.

[CS96] P. Cheeseman and J. Stutz. "Bayesian Classification (AutoClass): Theory and Results". In: *Advances in Knowledge Discovery and Data Mining*. Ed. by Fayyad, Pratetsky-Shapiro, Smyth, and Uthurasamy. MIT Press, 1996.

[CSF16] A. W. Churchill, S. Sigtia, and C. Fernando. "Learning to Generate Genotypes with Neural Networks". In: (Apr. 2016). arXiv: 1604.04153 [cs.NE].

[CY99] G. Cooper and C. Yoo. "Causal Discovery from a Mixture of Experimental and Observational Data". In: *UAI*. 1999.

[Dan+10] P. Daniusis, D. Janzing, J. Mooij, J. Zscheischler, B. Steudel, K. Zhang, and B. Schoelkopf. "Inferring deterministic causal relations". In: *UAI*. 2010.

[Daw02] A. P. Dawid. "Influence diagrams for causal modelling and inference". In: *Intl. Stat. Review* 70 (2002). Corrections p437, pp. 161–189.

[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.

[DDL97] S. DellaPietra, V. DellaPietra, and J. Lafferty. "Inducing features of random fields". In: *IEEE PAMI* 19.4 (1997).

[Dem72] A. Dempster. "Covariance selection". In: *Biometrics* 28.1 (1972).

[DGK08] J. Duchi, S. Gould, and D. Koller. "Projected Subgradient Methods for Learning Sparse Gaussians". In: *UAI*. 2008.

[DGR03] P. Dellaportas, P. Giudici, and G. Roberts. "Bayesian inference for nondecomposable graphical Gaussian models". In: *Sankhya, Ser. A* 65 (2003), pp. 43–55.

[DHS11] J. Duchi, E. Hazan, and Y. Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *JMLR* 12 (2011), pp. 2121–2159.

[Die+17] A. B. Dieng, C. Wang, J. Gao, and J. Paisley. "TopicRNN: A Recurrent Neural Network with Long-Range Semantic Dependency". In: *ICLR*. 2017.

[DL09] P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for AI*. Morgan & Claypool, 2009.

[DL93] A. P. Dawid and S. L. Lauritzen. "Hyper-Markov laws in the statistical analysis of decomposable graphical models". In: *The Annals of Statistics* 3 (1993), pp. 1272–1317.

[Dob09] A. Dobra. *Dependency networks for genome-wide data*. Tech. rep. U. Washington, 2009.

[Dom+06] P. Domingos, S. Kok, H. Poon, M. Richardson, and P. Singla. "Unifying Logical and Statistical AI". In: *IJCAI*. 2006.

[Don95] D. L. Donoho. "De-noising by soft-thresholding". In: *IEEE Trans. Inf. Theory* 41.3 (May 1995), pp. 613–627.

[DRB19] A. B. Dieng, F. J. R. Ruiz, and D. M. Blei. "The Dynamic Embedded Topic Model". In: (July 2019). arXiv: 1907.05545 [cs.CL].

[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.

[DVR08] J. Dahl, L. Vandenberghe, and V. Roychowdhury. "Covariance selection for non-chordal graphs via chordal embedding". In: *Optimization Methods and Software* 23.4 (2008), pp. 501–502.

[EAL10] D. Edwards, G. de Abreu, and R. Labouriau. "Selecting high-dimensional mixed graphical models using minimal AIC or BIC forests". In: *BMC Bioinformatics* 11.18 (2010).

[Ebe17] F. Eberhardt. "Introduction to the Foundations of Causal Discovery". In: *International Journal of Data Science and Analytics* 3.2 (2017), pp. 81–91.

[Eke+13] M. Ekeberg, C. Lövkvist, Y. Lan, M. Weigt, and E. Aurell. "Improved contact prediction in proteins: using pseudolikelihoods to infer Potts models". en. In: *Phys. Rev. E Stat. Nonlin. Soft Matter Phys.* 87.1 (Jan. 2013), p. 012707.

[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.

[Eli+00] G. Elidan, N. Lotner, N. Friedman, and D. Koller. "Discovering Hidden Variables: A Structure-Based Approach". In: *NIPS*. 2000.

[EM07] D. Eaton and K. Murphy. "Exact Bayesian structure learning from uncertain interventions". In: *AI/Statistics*. 2007.

[Eva+18] R. Evans et al. "De novo structure prediction with deep-learning based scoring". In: (2018).

[Eve09] G. Evensen. *Data Assimilation: The Ensemble Kalman Filter*. en. 2nd ed. 2009 edition. Springer, Aug. 2009.

[EW08] B. Ellis and W. H. Wong. "Learning Causal Bayesian Network Structures From Experimental Data". In: *JASA* 103.482 (2008), pp. 778–789.

[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.

[FHT08] J. Friedman, T. Hastie, and R. Tibshirani. "Sparse inverse covariance estimation the graphical lasso". In: *Biostatistics* 9.3 (2008), pp. 432–441.

[FK03] N. Friedman and D. Koller. "Being Bayesian about Network Structure: A Bayesian Approach to Structure Discovery in Bayesian Networks". In: *Machine Learning* 50 (2003), pp. 95–126.

[FK13] M Frei and H. R. Künsch. "Bridging the ensemble Kalman and particle filters". In: *Biometrika* 100.4 (Dec. 2013), pp. 781–800.

[Fri+02] N. Friedman, M. Ninion, I. Pe'er, and T. Pupko. "A Structural EM Algorithm for Phylogenetic Inference". In: *J. Comp. Bio.* 9 (2002), pp. 331–353.

[Fri97] N. Friedman. “Learning Bayesian Networks in the Presence of Missing Values and Hidden Variables”. In: *UAI*. 1997.

[FS18] S. Fattahi and S. Sojoudi. “Graphical Lasso and Thresholding: Equivalence and Closed-form Solutions”. In: *JMLR* (2018).

[FZS18] S. Fattahi, R. Y. Zhang, and S. Sojoudi. “Linear-Time Algorithm for Learning Large-Scale Sparse Graphical Models”. In: *IEEE Access* (2018).

[GG99] P. Giudici and P. Green. “Decomposable graphical Gaussian model determination”. In: *Biometrika* 86.4 (1999), pp. 785–801.

[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].

[GGS84] H. Gabow, Z. Galil, and T. Spencer. “Efficient implementation of graph algorithms using contraction”. In: *FOCS*. 1984.

[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.

[GH94] D. Geiger and D. Heckerman. “Learning Gaussian Networks”. In: *UAI*. Vol. 10. 1994, pp. 235–243.

[GH97] D. Geiger and D. Heckerman. “A characterization of Dirchlet distributions through local and global independence”. In: *Annals of Statistics* 25 (1997), pp. 1344–1368.

[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.

[GL97] F. Glover and M. Laguna. Kluwer Academic Publishers, 1997.

[GM04] A. Goldenberg and A. Moore. “Tractable Learning of Large Bayes Net Structures from Sparse Data”. In: *ICML*. 2004.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[Goo74] L. A. Goodman. “Exploratory latent structure analysis using both identifiable and unidentifiable models”. In: *Biometrika* 61.2 (1974), pp. 215–231.

[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.

[GR01] A. Gelman and T. Raghunathan. “Using conditional distributions for missing-data imputation”. In: *Statistical Science* (2001).

[Gri+04] T. Griffiths, M. Steyvers, D. Blei, and J. Tenenbaum. “Integrating Topics and Syntax”. In: *NIPS*. 2004.

[GS04] T. Griffiths and M. Steyvers. “Finding scientific topics”. In: *PNAS* 101 (2004), pp. 5228–5235.

[GS15] F. Glover and K. Sorensen. “Metaheuristics”. In: *Scholarpedia J.* 10.4 (2015), p. 6532.

[GSM18] U. Garciarena, R. Santana, and A. Mendiburu. “Expanding Variational Autoencoders for Learning and Exploiting Latent Representations in Search Distributions”. In: *Proc. of the Conf. on Genetic and Evolutionary Computation*. 2018, pp. 849–856.

[GT06] T. Griffiths and J. Tenenbaum. “Optimal predictions in everyday cognition”. In: *Psychological Science* 17.9 (2006), pp. 767–773.

[GT09] T. Griffiths and J. Tenenbaum. "Theory-Based Causal Induction". In: *Psychological Review* 116.4 (2009), pp. 661–716.

[Gu+20] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Re. "HiPPO: Recurrent Memory with Optimal Polynomial Projections". In: *NIPS* 33 (2020).

[Guo+21] R. Guo, L. Cheng, J. Li, P Richard Hahn, and H. Liu. "A Survey of Learning Causality with Data: Problems and Methods". In: *ACM Computing Surveys* 53.4 (2021).

[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.

[GZS19] C. Glymour, K. Zhang, and P. Spirtes. "Review of Causal Discovery Methods Based on Graphical Models". en. In: *Front. Genet.* 10 (June 2019), p. 524.

[Han16] N. Hansen. "The CMA Evolution Strategy: A Tutorial". In: (Apr. 2016). arXiv: 1604.00772 [cs.LG].

[Hau+11] M. Hauschild, M. Pelikan, M. Hauschild, and M. Pelikan. "An introduction and survey of estimation of distribution algorithms". In: *Swarm and Evolutionary Computation*. 2011.

[HB14] A. Hauser and P. Bühlmann. "Two optimal strategies for active learning of causal models from interventional data". In: *Int. J. Approx. Reason.* 55.4 (June 2014), pp. 926–939.

[HBB10] M. Hoffman, D. Blei, and F. Bach. "Online learning for latent Dirichlet allocation". In: *NIPS*. 2010.

[HDMM18] C. Heinze-Deml, M. H. Maathuis, and N. Meinshausen. "Causal Structure Learning". In: *Annu. Rev. Stat. Appl.* 5.1 (Mar. 2018), pp. 371–391.

[Hec+00] D. Heckerman, D. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. "Dependency Networks for Density Estimation, Collaborative Filtering, and Data Visualization". In: *JMLR* 1 (2000), pp. 49–75.

[HG09] Y.-B. He and Z. Geng. "Active learning of causal networks with intervention experiments and optimal designs". In: *JMLR* 10 (2009), pp. 2523–2547.

[HGC95] D. Heckerman, D. Geiger, and M. Chickering. "Learning Bayesian networks: the combination of knowledge and statistical data". In: *Machine Learning* 20.3 (1995), pp. 197–243.

[HKP91] J. Hertz, A. Krogh, and R. G. Palmer. *An Introduction to the Theory of Neural Comptuation*. Addison-Wesley, 1991.

[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.

[HMC97] D. Heckerman, C. Meek, and G. Cooper. *A Bayesian approach to Causal Discovery*. Tech. rep. MSR-TR-97-05. Microsoft Research, 1997.

[Hoe+99] J. Hoeting, D. Madigan, A. Raftery, and C. Volinsky. "Bayesian Model Averaging: A Tutorial". In: *Statistical Science* 4.4 (1999).

[Hof99] T. Hofmann. "Probabilistic latent semantic indexing". In: *Research and Development in Information Retrieval* (1999), pp. 50–57.

[Hol92] J. H. Holland. *Adaptation in Natural and Artificial Systems*. https://mitpress.mit.edu/books/adaptation-natural-and-artificial-systems. Accessed: 2017-11-26. Apr. 1992.

[Hop82] J. J. Hopfield. "Neural networks and physical systems with emergent collective computational abilities". In: *PNAS* 79.8 (1982), 2554–2558.

[Hoy+09] P. O. Hoyer, D. Janzing, J. M. Mooij, J. Peters, and P. B. Schölkopf. "Nonlinear causal discovery with additive noise models". In: *NIPS*. 2009, pp. 689–696.

[HS05] H. Hoos and T. Stutzle. *Stochastic local search: Foundations and applications*. Morgan Kauffman, 2005.

[HS08] T. Hazan and A. Shashua. "Convergent message-passing algorithms for inference over general graphs with convex free energy". In: *UAI*. 2008.

[HSDK12] C. Hillar, J. Sohl-Dickstein, and K. Koepsell. *Efficient and Optimal Binary Hopfield Associative Memory Storage Using Minimum Probability Flow*. Tech. rep. Apr. 2012. arXiv: 1204.2916.

[HT08] H. Hara and A. Takimura. "A Localization Approach to Improve Iterative Proportional Scaling in Gaussian Graphical Models". In: *Communications in Statistics - Theory and Method* (2008). to appear.

[HT09] H. Hoefling and R. Tibshirani. "Estimation of Sparse Binary Pairwise Markov Networks using Pseudo-likelihoods". In: *JMLR* 10 (2009).

[HTF09] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. 2nd edition. Springer, 2009.

[Hu+12] J. Hu, Y. Wang, E. Zhou, M. C. Fu, and S. I. Marcus. "A Survey of Some Model-Based Methods for Global Optimization". en. In: *Optimization, Control, and Applications of Stochastic Systems*. Systems & Control: Foundations & Applications. Birkhäuser, Boston, 2012, pp. 157–179.

[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.

[HW11] S. Harmeling and C. K. I. Williams. "Greedy Learning of Binary Latent Trees". In: *IEEE PAMI* 33.6 (2011), pp. 1087–1097.

[IM17] J. Ingraham and D. Marks. "Bayesian Sparsity for Intractable Undirected Models". In: *ICML*. 2017.

[IX00] K Ito and K Xiong. "Gaussian filters for nonlinear filtering problems". In: *IEEE Trans. Automat. Contr.* 45.5 (May 2000), pp. 910–927.

[Jan+12] D. Janzing, J. Mooij, K. Zhang, J. Lemeire, J. Zscheischler, P. Daniušis, B. Steudel, and B. Schölkopf. "Information-geometric approach to inferring causal directions". In: *AIJ* 182 (2012), pp. 1–31.

[Jay03] E. T. Jaynes. *Probability theory: the logic of science*. Cambridge university press, 2003.

[JBB09] D. Jian, A. Barthels, and M. Beetz. "Adaptive Markov logic networks: Learning statistical relational models with dynamic parameters". In: *9th European Conf. on AI*. 2009, 937–942.

[JHS13] Y. Jernite, Y. Halpern, and D. Sontag. "Discovering hidden variables in noisy-or networks using quartet tests". In: *NIPS*. 2013.

[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.

[Jin11] Y. Jin. "Surrogate-assisted evolutionary computation: Recent advances and future challenges". In: *Swarm and Evolutionary Computation* 1.2 (June 2011), pp. 61–70.

[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.

[Jon+05] B. Jones, A. Dobra, C. Carvalho, C. Hans, C. Carter, and M. West. "Experiments in stochastic computation for high-dimensional graphical models". In: *Statistical Science* 20 (2005), pp. 388–400.

[JS10] D. Janzing and B. Scholkopf. "Causal inference using the algorithmic Markov condition". In: *IEEE Trans. on Information Theory* 56.10 (2010), pp. 5168–5194.

[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.

[JZB19] A. Jaber, J. Zhang, and E. Bareinboim. "Identification of Conditional Causal Effects under Markov Equivalence". In: *NIPS*. 2019, pp. 11512–11520.

[KB07] M. Kalisch and P. Buhlmann. "Estimating high dimensional directed acyclic graphs with the PC algorithm". In: *JMLR* 8 (2007), pp. 613–636.

[KB14] M. Kalisch and P. Bühlmann. "Causal Structure Learning and Inference: A Selective Review". In: *Qual. Technol. Quant. Manag.* 11.1 (Jan. 2014), pp. 3–21.

[KF09] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[KNH11] K. B. Korb, E. P. Nyberg, and L. Hope. "A new causal power theory". In: *Causality in the Sciences*. Oxford University Press, 2011.

[KNP11] K. Kersting, S. Natarajan, and D. Poole. *Statistical Relational AI: Logic, Probability and Computation*. Tech. rep. UBC, 2011.

[Koi06] M. Koivisto. "Advances in exact Bayesian structure discovery in Bayesian networks". In: *UAI*. 2006.

[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.

[Koz92] J. R. Koza. *Genetic Programming*. https://mitpress.mit.edu/books/genetic-programming. Accessed: 2017-11-26. Dec. 1992.

[KS04] M. Koivisto and K. Sood. "Exact Bayesian structure discovery in Bayesian networks". In: *JMLR* 5 (2004), pp. 549–573.

[Lac+20] S. Lachapelle, P. Brouillard, T. Deleu, and S. Lacoste-Julien. "Gradient-Based Neural DAG Learning". In: *ICLR*. 2020.

[Lau92] S. L. Lauritzen. "Propagation of probabilities, means and variances in mixed graphical association models". In: *JASA* 87.420 (1992), pp. 1098–1108.

[LD08] A. Lenkoski and A. Dobra. *Bayesian structural learning and estimation in Gaussian graphical models*. Tech. rep. 545. Department of Statistics, University of Washington, 2008.

[Lev11] S. Levy. *In The Plex: How Google Thinks, Works, and Shapes Our Lives*. Simon & Schuster, 2011.

[LFJ04] M. Law, M. Figueiredo, and A. Jain. "Simultaneous Feature Selection and Clustering Using Mixture Models". In: *IEEE PAMI* 26.4 (2004).

[LGK06] S.-I. Lee, V. Ganapathi, and D. Koller. "Efficient Structure Learning of Markov Networks using L1-Regularization". In: *NIPS*. 2006.

[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.

[LHF17] R. M. Levy, A. Haldane, and W. F. Flynn. "Potts Hamiltonian models of protein co-variation, free energy landscapes, and evolutionary fitness". en. In: *Curr. Opin. Struct. Biol.* 43 (Apr. 2017), pp. 55–62.

[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+18] C. Li, H. Farkhoor, R. Liu, and J. Yosinski. "Measuring the Intrinsic Dimension of Objective Landscapes". In: *ICLR*. 2018.

[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).

[LL02] P. Larranaga and J. A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Norwell, MA, USA: Kluwer Academic Publishers, 2002.

[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].

[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.

[Lok+18] A. Y. Lokhov, M. Vuffray, S. Misra, and M. Chertkov. "Optimal structure and parameter learning of Ising models". en. In: *Science Advances* 4.3 (Mar. 2018), e1700791.

[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.

[Luk13] S. Luke. *Essentials of Metaheuristics*. 2013.

[May79] P. Maybeck. *Stochastic models, estimation, and control*. Academic Press, 1979.

[MB06] N. Meinshausen and P. Buhlmann. "High dimensional graphs and variable selection with the lasso". In: *The Annals of Statistics* 34 (2006), pp. 1436–1462.

[MB16] Y. Miao and P. Blunsom. "Language as a Latent Variable: Discrete Generative Models for Sentence Compression". In: *EMNLP*. 2016.

[MC03] P. Moscato and C. Cotta. "A Gentle Introduction to Memetic Algorithms". en. In: *Handbook of Metaheuristics*. International Series in Operations Research & Management Science. Springer, Boston, MA, 2003, pp. 105–144.

[McE20] R. McElreath. *Statistical Rethinking: A Bayesian Course with Examples in R and Stan (2nd edition)*. en. Chapman and Hall/CRC, 2020.

[McK+04] B. D. McKay, F. E. Oggier, G. F. Royle, N. J. A. Sloane, I. M. Wanless, and H. S. Wilf. " Acyclic digraphs and eigenvalues of (0,1)-matrices". In: *J. Integer Sequences* 7.04.3.3 (2004).

[Mei05] N. Meinshausen. *A note on the Lasso for Gaussian graphical model selection*. Tech. rep. ETH Seminar fur Statistik, 2005.

[MGR18] H. Mania, A. Guy, and B. Recht. "Simple random search of static linear policies is competitive for reinforcement learning". In: *NIPS*. Ed. by S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, and R Garnett. Curran Associates, Inc., 2018, pp. 1800–1809.

[MH12] R. Mazumder and T. Hastie. *The Graphical Lasso: New Insights and Alternatives*. Tech. rep. Stanford Dept. Statistics, 2012.

[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.

[MJ00] M. Meila and M. I. Jordan. "Learning with mixtures of trees". In: *JMLR* 1 (2000), pp. 1–48.

[MJ06] M. Meila and T. Jaakkola. "Tractable Bayesian learning of tree belief networks". In: *Statistics and Computing* 16 (2006), pp. 77–92.

[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.

[Mog+09] B. Moghaddam, B. Marlin, E. Khan, and K. Murphy. "Accelerating Bayesian Structural Inference for Non-Decomposable Gaussian Graphical Models". In: *NIPS*. 2009.

[Mol+18] D. Moldovan, V. Chifu, C. Pop, T. Cioara, I. Anghel, and I. Salomie. "Chicken Swarm Optimization and Deep Learning for Manufacturing Processes". In: *Networking in Education and Research (RoEduNet) conference*. Cluj-Napoca: IEEE, Sept. 2018, pp. 1–6.

[Moo+16] J. M. Mooij, J. Peters, D. Janzing, J. Zscheischler, and B. Schölkopf. "Distinguishing Cause from Effect Using Observational Data: Methods and Benchmarks". In: *JMLR* 17.1 (Jan. 2016), pp. 1103–1204.

[Mor+11] F. Morcos, A. Pagnani, B. Lunt, A. Bertolino, D. S. Marks, C. Sander, R. Zecchina, J. N. Onuchic, T. Hwa, and M. Weigt. "Direct-coupling analysis of residue coevolution captures native contacts across many protein families". en. In: *Proc. Natl. Acad. Sci. U. S. A.* 108.49 (Dec. 2011), E1293–301.

[MR94] D. Madigan and A. Raftery. "Model selection and accounting for model uncertainty in graphical models using Occam's window". In: *JASA* 89 (1994), pp. 1535–1546.

[Mue+17] J. Mueller, D. N. Reshef, G. Du, and T. Jaakkola. "Learning Optimal Interventions". In: *AISTATS*. 2017.

[Mur01] K. Murphy. *Active Learning of Causal Bayes Net Structure*. Tech. rep. Comp. Sci. Div., UC Berkeley, 2001.

[Mur22] K. P. Murphy. *Probabilistic Machine Learning: An introduciton*. MIT Press, 2022.

[Nea92] R. Neal. "Connectionist learning of belief networks". In: *Artificial Intelligence* 56 (1992), pp. 71–113.

[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.

[Oll+17] Y. Ollivier, L. Arnold, A. Auger, and N. Hansen. "Information-Geometric Optimization Algorithms: A Unifying Picture via Invariance Principles". In: *JMLR* 18 (2017), pp. 1–65.

[Oll18] Y. Ollivier. "Online natural gradient as a Kalman filter". en. In: *Electron. J. Stat.* 12.2 (2018), pp. 2930–2961.

[PB+14] N. Parikh, S. Boyd, et al. "Proximal algorithms". In: *Foundations and Trends in Optimization* 1.3 (2014), pp. 127–239.

[Pe'05] D. Pe'er. "Bayesian network analysis of signaling networks: a primer". In: *Science STKE* 281 (2005), p. 14.

[PE08] J.-P. Pellet and A. Elisseeff. "Using Markov blankets for causal structure learning". In: *JMLR* 9 (2008), pp. 1295–1342.

[Pea09] J. Pearl. *Causality: Models, Reasoning and Inference (Second Edition)*. Cambridge Univ. Press, 2009.

[Pel05] M. Pelikan. *Hierarchical Bayesian Optimization Algorithm: Toward a New Generation of Evolutionary Algorithms*. en. Softcover reprint of hardcover 1st ed. 2005 edition. Springer, 2005.

[PGCP00] M Pelikan, D. E. Goldberg, and E Cantú-Paz. "Linkage problem, distribution estimation, and Bayesian networks". en. In: *Evol. Comput.* 8.3 (2000), pp. 311–340.

[PHL12] M. Pelikan, M. Hausschild, and F. Lobo. *Introduction to estimation of distribution algorithms*. Tech. rep. U. Missouri, 2012.

[PJS17] J. Peters, D. Janzing, and B. Schölkopf. *Elements of Causal Inference: Foundations and Learning Algorithms (Adaptive Computation and Machine Learning series)*. The MIT Press, Nov. 2017.

[PK11] P. Parviainen and M. Koivisto. "Ancestor Relations in the Presence of Unobserved Variables". In: *ECML*. 2011.

[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.

[Poo+12] D. Poole, D. Buchman, S. Natarajan, and K. Kersting. "Aggregation and Population Growth: The Relational Logistic Regression and Markov Logic Cases". In: *Statistical Relational AI workshop*. 2012.

[PRG17] M. Probst, F. Rothlauf, and J. Grahl. "Scalability of using Restricted Boltzmann Machines for combinatorial optimization". In: *Eur. J. Oper. Res.* 256.2 (Jan. 2017), pp. 368–383.

[Pri12] S. Prince. *Computer Vision: Models, Learning and Inference*. Cambridge, 2012.

[PSCP06] M. Pelikan, K. Sastry, and E. Cantú-Paz. *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications (Studies in Computational Intelligence)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[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.

[RD06] M. Richardson and P. Domingos. "Markov logic networks". In: *Machine Learning* 62 (2006), pp. 107–136.

[Rea+19] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. "Regularized Evolution for Image Classifier Architecture Search". In: *AAAI*. 2019.

[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.

[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).

[RK04] R. Rubinstein and D. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Springer-Verlag, 2004.

[RM15] G. Raskutti and S. Mukherjee. "The information geometry of mirror descent". In: *IEEE Trans. Info. Theory* 61.3 (2015), pp. 1451–1457.

[RN10] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. 3rd edition. Prentice Hall, 2010.

[Rob73] R. W. Robinson. "Counting labeled acyclic digraphs". In: *New Directions in the Theory of Graphs*. Ed. by F. Harary. Academic Press, 1973, pp. 239–273.

[Roc06] S. Roch. "A short proof that phylogenetic tree reconstrution by maximum likelihood is hard". In: *IEEE/ACM Trans. Comp. Bio. Bioinformatics* 31.1 (2006).

[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.

[Rov02] A. Roverato. "Hyper inverse Wishart distribution for non-decomposable graphs and its application to Bayesian inference for Gaussian graphical models". In: *Scand. J. Statistics* 29 (2002), pp. 391–411.

[RU10] A. Rajaraman and J. Ullman. *Mining of massive datasets*. Self-published, 2010.

[Rub97] R. Y. Rubinstein. "Optimization of computer simulation models with rare events". In: *Eur. J. Oper. Res.* 99.1 (May 1997), pp. 89–112.

[Rus22] S. Rush. *Illustrated S4*. Tech. rep. 2022.

[Sac+05] K. Sachs, O. Perez, D. Pe'er, D. Lauffenburger, and G. Nolan. "Causal Protein-Signaling Networks Derived from Multiparameter Single-Cell Data". In: *Science* 308 (2005).

[Sal+17] T. Salimans, J. Ho, X. Chen, and I. Sutskever. "Evolution Strategies as a Scalable Alternative to Reinforcement Learning". In: (Mar. 2017). arXiv: 1703.03864 [stat.ML].

[San17] R. Santana. "Gray-box optimization and factorized distribution algorithms: where two worlds collide". In: (July 2017). arXiv: 1707.03093 [cs.NE].

[Sar13] S. Sarkka. *Bayesian Filtering and Smoothing*. Cambridge University Press, 2013.

[SC08] J. G. Scott and C. M. Carvalho. "Feature-inclusion Stochastic Search for Gaussian Graphical Models". In: *J. of Computational and Graphical Statistics* 17.4 (2008), pp. 790–808.

[Sch+08] M. Schmidt, K. Murphy, G. Fung, and R. Rosales. "Structure Learning in Random Fields for Heart Motion Abnormality Detection". In: *CVPR*. 2008.

[Sch+09] M. Schmidt, E. van den Berg, M. Friedlander, and K. Murphy. "Optimizing Costly Functions with Simple Constraints: A Limited-Memory Projected Quasi-Newton Algorithm". In: *AI & Statistics*. 2009.

[Sch10a] M. Schmidt. "Graphical model structure learning with L1 regularization". PhD thesis. UBC, 2010.

[Sch10b] 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].

[Sch+21] B. Schölkopf, F. Locatello, S. Bauer, N. R. Ke, N. Kalchbrenner, A. Goyal, and Y. Bengio. "Toward Causal Representation Learning". In: *Proc. IEEE* 109.5 (May 2021), pp. 612–634.

[Seg11] D. Segal. "The dirty little secrets of search". In: *New York Times* (2011).

[Sen+08] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. "Collective Classification in Network Data". en. In: *AI Magazine* 29.3 (Sept. 2008), pp. 93–93.

[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.

[SG09] R. Silva and Z. Ghahramani. "The Hidden Life of Latent Variables: Bayesian Learning with Mixed Graph Models". In: *JMLR* 10 (2009), pp. 1187–1238.

[SG91] P. Spirtes and C. Glymour. "An algorithm for fast recovery of sparse causal graphs". In: *Social Science Computer Review* 9 (1991), pp. 62–72.

[SGS00] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. 2nd edition. MIT Press, 2000.

[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.

[SH06] T. Singliar and M. Hauskrecht. "Noisy-OR Component Analysis and its Application to Link Analysis". In: *JMLR* 7 (2006).

[SH10] R. Salakhutdinov and G. Hinton. "Replicated Softmax: an Undirected Topic Model". In: *NIPS*. 2010.

[Shi+06] S. Shimizu, P. O. Hoyer, A. Hyvärinen, and A. Kerminen. "A Linear Non-Gaussian Acyclic Model for Causal Discovery". In: *JMLR* 7.Oct (2006), pp. 2003–2030.

[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.

[SK86] T. Speed and H. Kiiveri. "Gaussian Markov distributions over finite graphs". In: *Annals of Statistics* 14.1 (1986), pp. 138–150.

[SKM07] T. Silander, P. Kontkanen, and P. Myllymaki. "On Sensitivity of the MAP Bayesian Network Structure to the Equivalent Sample Size Parameter". In: *UAI*. 2007, pp. 360–367.

[SLM92] B. Selman, H. Levesque, and D. Mitchell. "A New Method for Solving Hard Satisfiability Problems". In: *Proceedings of the Tenth National Conference on Artificial Intelligence*. AAAI'92. San Jose, California: AAAI Press, 1992, pp. 440–446.

[SM06] T. Silander and P. Myllymaki. "A simple approach for finding the globally optimal Bayesian network structure". In: *UAI*. 2006.

[SM09] M. Schmidt and K. Murphy. “Modeling Discrete Interventional Data using Directed Cyclic Graphical Models”. In: *UAI*. 2009.

[SMH07] R. R. Salakhutdinov, A. Mnih, and G. E. Hinton. “Restricted Boltzmann machines for collaborative filtering”. In: *ICML*. Vol. 24. 2007, pp. 791–798.

[SNMM07] M. Schmidt, A. Niculescu-Mizil, and K. Murphy. “Learning Graphical Model Structure using L1-Regularization Paths”. In: *AAAI*. 2007.

[Sol19] L. Solus. “Interventional Markov Equivalence for Mixed Graph Models”. In: (Nov. 2019). arXiv: 1911.10114 [math.ST].

[Sör15] K. Sörensen. “Metaheuristics—the metaphor exposed”. In: *Intl. Trans. in Op. Res.* 22.1 (Jan. 2015), pp. 3–18.

[SP18] R. D. Shah and J. Peters. “The Hardness of Conditional Independence Testing and the Generalised Covariance Measure”. In: *Ann. Stat.* (2018).

[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.

[SS17] A. Srivastava and C. Sutton. “Autoencoding Variational Inference For Topic Models”. In: *ICLR*. 2017.

[Sta+19] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen. “Designing neural networks through neuroevolution”. In: *Nature Machine Intelligence* 1.1 (2019).

[SW11] R. Sedgewick and K. Wayne. *Algorithms*. Addison Wesley, 2011.

[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.

[Teh+06] Y.-W. Teh, M. Jordan, M. Beal, and D. Blei. “Hierarchical Dirichlet processes”. In: *JASA* 101.476 (2006), pp. 1566–1581.

[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.

[TG09] A. Thomas and P. Green. “Enumerating the Decomposable Neighbours of a Decomposable Graph Under a Simple Perturbation Scheme”. In: *Comp. Statistics and Data Analysis* 53 (2009), pp. 1232–1238.

[Thi+98] B. Thiesson, C. Meek, D. Chickering, and D. Heckerman. “Learning Mixtures of DAG models”. In: *UAI*. 1998.

[Tse08] P. Tseng. *On accelerated proximal gradient methods for convex-concave optimization*. Unpublished manuscript. 2008.

[Tu+19] R. Tu, K. Zhang, B. C. Bertilson, H. Kjellström, and C. Zhang. “Neuropathic Pain Diagnosis Simulator for Causal Discovery Algorithm Evaluation”. In: *NIPS*. 2019.

[Var21] K. R. Varshney. *Trustworthy Machine Learning*. 2021.

[VML19] M. Vuffray, S. Misra, and A. Y. Lokhov. "Efficient Learning of Discrete Graphical Models". In: (Feb. 2019). arXiv: 1902.00600 [cs.LG].

[VP90] T. Verma and J. Pearl. "Equivalence and synthesis of causal models". In: *UAI*. 1990.

[Wal+09] H. Wallach, I. Murray, R. Salakhutdinov, and D. Mimno. "Evaluation Methods for Topic Models". In: *ICML*. 2009.

[WCK03] F. Wong, C. Carter, and R. Kohn. "Efficient estimation of covariance selection models". In: *Biometrika* 90.4 (2003), pp. 809–830.

[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.

[Wie+14] D Wierstra, T Schaul, J Peters, and J Schmidhuber. "Natural Evolution Strategies". In: *JMLR* 15.1 (2014), pp. 949–980.

[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.

[WP18] H. Wang and H. Poon. "Deep Probabilistic Logic: A Unifying Framework for Indirect Supervision". In: *EMNLP*. 2018.

[WRL06] M. Wainwright, P. Ravikumar, and J. Lafferty. "Inferring Graphical Model Structure using $\ell - 1$-Regularized Pseudo-Likelihood". In: *NIPS*. 2006.

[WSD19] S. Wu, S. Sanghavi, and A. G. Dimakis. "Sparse Logistic Regression Learns All Discrete Pairwise Graphical Models". In: *NIPS*. 2019.

[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).

[Xu18] J. Xu. "Distance-based Protein Folding Powered by Deep Learning". In: (Nov. 2018). arXiv: 1811.03481 [q-bio.BM].

[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.

[YL07] M. Yuan and Y. Lin. "Model Selection and Estimation in the Gaussian Graphical Model". In: *Biometrika* 94.1 (2007), pp. 19–35.

[Yu+19] Y. Yu, J. Chen, T. Gao, and M. Yu. "DAG-GNN: DAG Structure Learning with Graph Neural Networks". In: *ICML*. 2019.

[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.

[Zha04] N. Zhang. "Hierarchical latent class models for cluster analysis". In: *JMLR* (2004), pp. 301–308.

[Zha+20] Y. Zhang, X. Chen, Y. Yang, A. Ramamurthy, B. Li, Y. Qi, and L. Song. "Efficient Probabilistic Logic Reasoning with Graph Neural Networks". In: *ICLR*. 2020.

[Zhe+18] X. Zheng, B. Aragam, P. Ravikumar, and E. P. Xing. "DAGs with NO TEARS: Smooth Optimization for Structure Learning". In: *NIPS*. 2018.

[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.

[ZWM97] C. S. Zhu, N. Y. Wu, and D. Mumford. "Minimax Entropy Principle and Its Application to Texture Modeling". In: *Neural Computation* 9.8 (Nov. 1997).